МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ЛЯЩИНСЬКИЙ ПЕТРО БОРИСОВИЧ

Кваліфікаційна наукова
праця на правах рукопису
УДК 004.89 + 004.02

Синтез біомедичних зображень на основі глибоких нейронних мереж

122 — Комп’ютерні науки

Дисертація на здобуття наукового ступеня
доктора філософії

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

_____ /П.Б. Лящинський/

Науковий керівник
Березький Олег Миколайович
доктор технічних наук, професор

Львів – 2024

АНОТАЦІЯ

Ляцинський П.Б. Синтез біомедичних зображень на основі глибоких нейронних мереж. — Кваліфікаційна наукова праця на правах рукопису. Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки». — Національний університет «Львівська політехніка», Львів, 2024.

Зміст анотації. Дисертаційне дослідження присвячене розробленню, вдосконаленню методів і засобів синтезу та розширення навчальних наборів для підвищення точності класифікації глибокими нейронними мережами біомедичних зображень. Основна частина роботи включає вступ, чотири розділи та висновки.

У вступі розглянуто актуальність теми дослідження, визначено мету та завдання дослідження, виділено наукову новизну та практичну значущість.

У першому розділі «Аналіз нейромережевих методів синтезу зображень» проведено аналіз біомедичних зображень, а саме цитологічних, гістологічних та імуногістохімічних. Також проведено аналіз та порівняння класичних архітектур генеративно-змагальних нейронних мереж для задач синтезу біомедичних зображень, проаналізовано алгоритми навчання генеративно-змагальних мереж, проведено аналіз та порівняння алгоритмів навчання нейронних мереж на основі градієнтного спуску, проаналізовано та здійснено порівняння метрик для оцінки подібності синтетичних і реальних зображень. Результати порівняння показали, що відомі архітектури генеративно-змагальних нейронних мереж синтезують зображення, якість яких не задовільняє згідно метрик задачам навчання класифікаторів.

У другому розділі «Метод автоматичного синтезу архітектур згорткових нейронних мереж» розроблено метод синтезу архітектур згорткових нейронних мереж. Проведено порівняння архітектури, що синтезована розробленим методом, з іншими відомими дослідженнями для класифікації цитологічних зображень, використовуючи набір даних Sipakmed. Розроблений метод для

синтезу архітектур згорткових нейронних мереж складається із етапів мікро- та макропошуку. Етап мікропошуку використовується для оптимізації архітектури комірки, а етап макропошуку — для оптимізації кількості комірок, за допомогою яких будується фінальна архітектура нейронної мережі. Крім того, здійснено порівняння оптимізованої архітектури нейромережі із відомими архітектурами, використовуючи набір цитологічних зображень. Результати показали, що архітектура, створена розробленим методом, значно перевершує відомі архітектури по точності класифікації (99,125%).

У третьому розділі «Метод автоматичного синтезу біомедичних зображень та архітектур генеративно-змагальних нейронних мереж» розроблено метод для синтезу архітектур генеративно-змагальних нейронних мереж, який складається із двох кроків: пошук архітектури генератора із фіксованим дискримінатором та пошук архітектури дискримінатора із найкращим генератором. У розділі також здійснено порівняння якості синтетичних зображень, створених оптимізованою архітектурою, із зображеннями синтезованими іншими відомими архітектурами генеративно-змагальних мереж. Результати показали, що оптимізована архітектура синтезує зображення кращої якості за метрикою FID, що дорівнює 3,39. У розділі також розроблено автоматичний метод синтезу біомедичних зображень.

У четвертому розділі «Програмний засіб синтезу та класифікації біомедичних зображень» описано функціональні та системні вимоги до програмного засобу та його програмну реалізацію. У даному розділі розроблено архітектуру програмного засобу із використанням сучасних технологій в проектуванні та розробці програмного забезпечення. Програмний засіб побудовано з використанням клієнт-серверної модульної архітектури, що дає змогу розгортати компоненти системи або на локальному комп’ютері, або на будь-якій хмарній інфраструктурі (наприклад, GCP, AWS, Microsoft Azure та інші). Такий підхід дає змогу масштабувати систему горизонтально, якщо є потреба. Також в даному розділі розроблено структуру бази даних програмного

засобу використовуючи UML діаграму класів. Основними модулями програми є модуль наборів даних (датасетів), класифікаторів та генераторів. Програмний засіб дає можливість використовувати відомі або власні архітектури нейронних мереж для синтезу та класифікації біомедичних зображень. Також архітектури класифікаторів та генераторів можна синтезувати за допомогою методів, які розроблено у третьому розділі.

Основні наукові результати дисертації опубліковано у 18 працях, зокрема: п’ять статей – у наукових фахових періодичних виданнях України; дві статті – у наукових фахових періодичних виданнях України, що входять до наукометричної бази Web of Science; сім публікацій – у матеріалах міжнародних та всеукраїнських наукових, науково-технічних конференцій (п’ять із них входять до наукометричної бази Scopus); розділ колективної монографії – у наукових періодичних виданнях іншої держави; одна стаття – на іноземному сервісі arXiv.org (Cornell University); одна стаття – у наукових періодичних виданнях іншої держави (Scopus, Q1); авторське право.

Ключові слова: біомедичні зображення, цитологічні зображення, нейромережеві методи, класифікація зображень, згорткові нейронні мережі, генеративно-змагальні мережі, синтез зображень, програмний засіб, автоматичний пошук архітектури нейронної мережі.

ABSTRACT

Liashchynskyi P.B. Synthesis of biomedical images based on deep neural networks. — Qualifying scientific work on manuscript rights. Dissertation for obtaining the scientific degree of Doctor of Philosophy in specialty 122 «Computer Science». — Lviv Polytechnic National University, Lviv, 2024.

Abstract content. The dissertation research is devoted to the development, improvement of methods and tools for synthesizing and expanding training sets to improve the accuracy of classification of biomedical images by deep neural networks. The main part of the work includes an introduction, four chapters and conclusions.

The introduction discusses the relevance of the research topic, defines the purpose and objectives of the study, and highlights the scientific novelty and practical significance.

The first chapter, “Analysis of Neural Network Methods for Image Synthesis,” analyzes biomedical images, namely cytological, histological, and immunohistochemical images. We also analyze and compare the classical architectures of generative-competitive neural networks for biomedical image synthesis tasks, analyze the algorithms for training a generative-competitive network, analyze and compare the algorithms for training neural networks based on gradient descent, and analyze and compare metrics for assessing the similarity of synthetic and real images. The comparison results showed that the known architectures of generative-competitive neural networks synthesize images whose quality does not satisfy the classifier training tasks according to the metrics.

In Chapter 2, “A Method for Automatic Synthesis of Convolutional Neural Network Architectures,” a method for synthesizing convolutional neural network architectures has been developed. The architecture synthesized by the developed method is compared with other known studies for cytological image classification using the Sipakmed dataset. The developed method for the synthesis of convolutional neural network architectures consists of micro- and macro-search stages. The micro-search stage is used to optimize the cell architecture, and the macro-search stage is

used to optimize the number of cells that are used to build the final neural network architecture. In addition, the optimized neural network architecture is compared with known architectures using a set of cytological images. The results show that the architecture created by the developed method significantly outperforms the known architectures in terms of classification accuracy (99.125%).

In the third chapter, “A Method for Automatic Synthesis of Biomedical Images and Architectures of Generative Adversarial Neural Networks,” a method for synthesizing architectures of generative adversarial neural networks has been developed, which consists of two steps: searching for a generator architecture with a fixed discriminator and searching for a discriminator architecture with the best generator. The chapter also compares the quality of synthetic images created by the optimized architecture with images synthesized by other known generative adversarial network architectures. The results show that the optimized architecture synthesizes images of better quality with a FID metric of 3.39. The chapter also develops an automatic method for synthesizing biomedical images.

The fourth chapter, “Software for the synthesis and classification of biomedical images,” describes the functional and system requirements for the software and its software implementation. In this section, the architecture of the software tool is developed using modern technologies in software design and development. The software tool is built using a client-server modular architecture, which allows you to deploy system components either on a local computer or on any cloud infrastructure (for example, GCP, AWS, Microsoft Azure, etc.). This approach allows you to scale the system horizontally, if necessary. Also, in this section, the structure of the software database is developed using a UML class diagram. The main modules of the program are the module of data sets (datasets), classifiers and generators. The software tool makes it possible to use known or proprietary neural network architectures for the synthesis and classification of biomedical images. Also, classifier and generator architectures can be synthesized using the methods developed in Chapter 3.

The main scientific results of the dissertation are published in 18 works, in particular: five articles - in scientific professional periodicals of Ukraine; two articles - in scientific professional periodicals of Ukraine, which are included in the Web of Science; seven publications - in the proceedings of international and all-Ukrainian scientific, scientific and technical conferences (five of them are included in the Scopus scientometric database); a section of a collective monograph - in scientific periodicals of another country; one preprint - on the foreign service arXiv. org (Cornell University); one article - in scientific periodicals of another country (Scopus, Q1); copyright.

Keywords: biomedical images, cytological images, neural network methods, image classification, CNN networks, GAN networks, image synthesis, software tool, automatic search for neural network architecture.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у фахових виданнях України:

1. Інтелектуальна система автоматизованої мікроскопії аналізу гістологічних та цитологічних зображень / О.М. Березький, О.Й. Піцун, П.Б. Лящинський, П.Б. Лящинський, Г.М. Мельник // Штучний інтелект. — 2017. — № 2. — С. 129-141. — Бібліогр.: 9 назв. — укр. [1]
2. Синтез біомедичних зображень на підставі генеративно-змагальних мереж / О. М. Березький, П. Б. Лящинський, П. Б. Лящинський, А. Р. Сухович, Т. М. Долинюк // Український журнал інформаційних технологій. — Львів : Видавництво Львівської політехніки, 2019. — Том 1. — № 1. — С. 35–40. [2]
3. Berezsky O.M.., Liashchynskyi P.B. “Comparison of generative adversarial networks architectures for biomedical images synthesis”. Applied Aspects of Information Technology. 2021; Vol. 4, No. 3: 250–260. DOI:https://doi.org/10.15276/aait.03.2021.4. [3]
4. Liashchynskyi, P., & Liashchynskyi, P. (2023). Analysis of metrics for GAN evaluation. Computer Systems and Information Technologies, (4), 44–51. https://doi.org/10.31891/csit-2023-4-6. [4]
5. Berezsky, O. M., Liashchynskyi, P. B., Pitsun, O. Y., & Melnyk, G. M. (2024). Deep network-based method and software for small sample biomedical image generation and classification . Radio Electronics, Computer Science, Control, (4), 76. https://doi.org/10.15588/1607-3274-2023-4-8 (**Web of Science**) [5]
6. Berezsky, O. M., & Liashchynskyi, P. B. (2024). Method of generative-adversarial networks searching architectures for biomedical images synthesis . Radio Electronics, Computer Science, Control, (1), 104. https://doi.org/10.15588/1607-3274-2024-1-10 (**Web of Science**) [6]
7. Лящинський, П. Б. (2024). Програмний засіб для класифікації та синтезу біомедичних зображень. *Науковий вісник НЛТУ України*, *34*(4), 120-127. [7]

Статті у виданнях інших держав:

8. Liashchynskyi, P.B., & Liashchynskyi, P. (2019). Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. ArXiv, abs/1912.06059. [8]
9. Berezsky O, Liashchynskyi P. Synthesis of Convolutional Neural Network architectures for biomedical image classification, Biomedical Signal Processing and Control, Volume 95, Part B, 2024, 106325, https://doi.org/10.1016/j.bspc.2024.106325. (**Scopus**, **Q1**) [9]

Розділ колективної монографії:

10. Berezsky, O., Pitsun, O., Liashchynskyi, P., Derysh, B., Batryn, N. (2023). Computational Intelligence in Medicine. In: Babichev, S., Lytvynenko, V. (eds) Lecture Notes in Data Engineering, Computational Intelligence, and Decision Making. ISDMCI 2022. Lecture Notes on Data Engineering and Communications Technologies, vol 149. Springer, Cham. https://doi.org/10.1007/978-3-031-16203-9_28 (**Scopus**) [10]

Матеріали конференцій:

11. O. Berezsky, O. Pitsun, L. Dubchak, P. Liashchynskyi and P. Liashchynskyi, "GPU-based biomedical image processing," 2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Ukraine, 2018, pp. 96-99, doi: 10.1109/MEMSTECH.2018.8365710. (**Scopus**) [11]
12. Лящинський П. Порівняння GAN-архітектур для синтезу біомедичних зображень // III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» 2020, 26 листопада 2020р.: тези доп. – Тернопіль, 2020. – С. 31. [12]
13. Berezsky, O., Liashchynskyi, P., Pitsun, O., Liashchynskyi, P., & Berezkyy, M. Comparison of Deep Neural Network Learning Algorithms for Biomedical Image Processing. CEUR Workshop Proceedings. 2022. Vol. 3302: 5th

International Conference on Informatics & Data-Driven Medicine, November 18–20, 2022, Lyon, France. P. 135-145. (**Scopus**) [13]

14. Berezsky, O., Pitsun, O., Melnyk, G., Batko, Y., Derysh, B., & Liashchynskyi, P. Application Of MLOps Practices For Biomedical Image Classification. CEUR Workshop Proceedings. 2022. Vol. 3302: 5th International Conference on Informatics & Data-Driven Medicine, November 18–20, 2022, Lyon, France. P. 69-77. (**Scopus**) [14]
15. Berezsky, O., Liashchynskyi, P., Pitsun, O., Melnyk, G. Method and software tool for generating artificial databases of biomedical images based on deep neural networks. CEUR Workshop Proceedings. 2023. Vol. 3609: 6th International Conference on Informatics & Data-Driven Medicine, November 17 - 19, 2023, Bratislava, Slovakia. P. 15-26. (**Scopus**) [15]
16. Berezsky, O., Pitsun, O., Melnyk, G., Batko, Y., Liashchynskyi, P., & Berezkyy, M. MLOps approach for automatic segmentation of biomedical images. CEUR Workshop Proceedings. 2023. Vol. 3609: 6th International Conference on Informatics & Data-Driven Medicine, November 17 - 19, 2023, Bratislava, Slovakia. P. 241-248. (**Scopus**) [16]
17. Лящинський, П. Б., & Лящинський, П. Б. Використання систем автоматизованої мікроскопії для покращення аналізу біомедичних зображень // Економічний і соціальний розвиток України в XXI столітті: національна візія та виклики глобалізації, с. 729. [17]

Авторське право:

18. BCADCID breast cancer artificial digital cytology image database.: copyright certificate 123607 / O. M. Berezsky, P. B. Liashchynskyi, G. M. Melnyk, O. Y. Pitsun. Application. November 20, 2023, published February 8, 2024. [18]

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS – amazon web services (веб-сервіси Amazon);

AGA – aging genetic algorithm (генетичний алгоритм з еволюцією старіння);

API – application programming interface (прикладний програмний інтерфейс);

BCADCID – breast cancer artificial digital cytology image database (база синтетичних цитологічних зображень раку молочної залози);

CPU – central processing unit (центральний процесор);

CI/CD – continuous integration / continuous delivery (безперервна інтеграція і доставка);

CNN – convolutional neural network (згорткова нейронна мережа);

CRUD – create, read, update, delete (створити, прочитати, оновити, видалити);

DCGAN – deep convolutional generative adversarial network (глибока згорткова генеративно-змагальна мережа);

ELU – exponential linear unit (експоненціальний випрямляч, функція активації);

WGAN – wasserstein generative adversarial network (генеративно-змагальна мережа Васерштейна);

WGAN-GP – wasserstein generative adversarial network with gradient penalty (генеративно-змагальна мережа Васерштейна з штрафуванням градієнта);

BGAN – boundary-seeking generative adversarial network (генеративно-змагальна мережа з пошуком межі);

BEGAN – boundary equilibrium generative adversarial network (генеративно-змагальна мережа граничної рівноваги);

GAN – generative adversarial network (генеративно-змагальна мережа);

GA – genetic algorithm (генетичний алгоритм);

GCP – google cloud platform (платформа хмарних обчислень Google);

H – height (висота);

W – width (ширина);

C – channels (кількість каналів);

HTTP – hyper-text transfer protocol (протокол передачі гіпертексту);

CSV – comma-separated values (значення, розділені комою);

GPU – graphical processing unit (графічний процесор);

IS – inception score (метрика Inception);

FID – frechet inception distance (метрика Фреше);

JSON – javascript object notation (нотація об’єкта мови програмування Javascript);

FPGA – field programmable gate arrays (програмована логічна інтегральна схема);

RAM – random access memory (оперативна пам’ять);

RELU – rectified linear unit (випрямляч, різновид функції активації);

ROC – receiver operating charachteristic (операційна характеристика класифікатора);

RL – reinforcement learning (навчання з підкріпленням);

E – математичне сподівання;

MSE – mean squared error (середньоквадратичне похибка);

RMSE – root mean squared error (коренева середньоквадратична похибка);

SSIM – structure similarity index measure (структурна подібність);

PSNR – peak signal to noise ration (пікове співвідношення сигналу до шуму);

NAS – neural architecture search (пошук архітектури нейромережі);

TPU – tensor processing unit (тензорний процесор Google);

TP – true positive (істинно позитивні випадки);

TN – true negative (істинно негативні випадки);

FP – false positive (хибно позитивні випадки);

FN – false negative (хибно негативні випадки);

URL – unified resource location (уніфікований локатор ресурсів);

ЗНМ – згорткова нейронна мережа;

ГЗМ – генеративно-змагальна нейронна мережа.

ВСТУП

Обґрунтування вибору теми дослідження. Біомедичним зображенням називають структурно-функціональний образ органів людини, призначений для діагностики захворювань, а також вивчення анатомо-фізіологічної картини організму [1], [19]. Біомедичні зображення використовуються для візуалізації роботи органів людини і тварин та постановки діагнозу. Серед цих зображень є підкласи (цитологічні, гістологічні, імуногістохімічні), які використовуються для постановки діагнозу в онкології [20], [21]. Автоматизоване опрацювання цих зображень є трудомістким, тривалим в часі процесом, який вимагає спеціальних знань з комп'ютерного зору лікарів-діагностів. Завдяки розвитку обчислювального інтелекту, глибокого машинного навчання з'явились сучасні автоматичні класифікатори зображень, які проектуються на основі глибоких нейронних мереж [22].

Значний внесок у розвиток методів обчислювального інтелекту, а саме глибокого машинного навчання, згорткових та генеративно-змагальних нейронних мереж, зробили такі: Лекун та Й. Бенджіо [23], А. Крижевський [24], Дж. Хінтон [25], Я. Гудфеллоу [26], [27], [28], О. Івахненко [29], Є. Бодянський [30], [31], С. Субботін [32], [33], [34], Д. Пелешко [35], [36].

Застосування нейронних мереж вимагає розв'язку трьох основних задач для отримання потрібної точності класифікації: проектування архітектури нейронної мережі, збір великих навчальних вибірок і процес навчання нейронної мережі. Завдяки наявності великих навчальних вибірок досягається висока точність класифікації. Проте наявні навчальні вибірки біомедичних зображень є обмеженими. Тому існує суперечність між точністю сучасних класифікаторів і розміром вибірок для їх навчання. Особливо гостро ця проблема стоїть для цитологічних, гістологічних та імуногістохімічних зображень. Тому розширення наборів цих зображень для навчання нейронних мереж є актуальною задачею.

Отже, науковою задачею є розроблення, вдосконалення методів і засобів

синтезу та розширення навчальних наборів для підвищення точності класифікації глибокими нейронними мережами біомедичних зображень.

Зв'язок роботи з науковими програмами, планами і темами. Дисертація виконувалася відповідно до пріоритетних напрямів науково-дослідних робіт Західноукраїнського національного університету (ЗУНУ) та Національного університету «Львівська політехніка» (НУЛП) та відповідно до координаційних планів Міністерства освіти й науки України. Зокрема, наукові дослідження виконувалися в рамках держбюджетних та господарських наукових тем кафедри комп'ютерної інженерії ЗУНУ та автоматизованих систем управління НУЛП: «Гібридна інтелектуальна інформаційна технологія діагностування передракових станів молочної залози на основі аналізу зображень» (держ. номер 0116U002500), «Нейромережеві методи і засоби класифікації зображень ауто- та ксеногенних тканин» (держ. номер 0119U103227), «Методи машинного навчання для кластеризації та класифікації зображень ауто- та ксеногенних тканин» (держ. номер 0120U103514), «Високопродуктивна комп'ютерна система опрацювання біомедичних зображень» (держ. номер 0122U201124), «Методи та засоби нейронечіткого управління групою мобільних робототехнічних платформ» (держ. номер 0123U101688). Довідки про участь у виконанні науково-дослідних робіт ЗУНУ та НУЛП наведено у додатках А та Б.

Мета і завдання дослідження.

Метою дослідження є розроблення, вдосконалення методів і засобів синтезу та розширення навчальних наборів для підвищення точності класифікації глибокими нейронними мережами біомедичних зображень.

Для досягнення поставленої мети необхідно вирішити такі *завдання:*

- провести аналіз нейромережевих методів, засобів синтезу зображень та сформулювати завдання дослідження;
- розробити метод автоматичного синтезу архітектур згорткових нейронних мереж для класифікації біомедичних зображень;

- розробити метод автоматичного синтезу архітектур генеративно-змагальних нейронних мереж для задач генерування біомедичних зображень;
- вдосконалити метод генерування та класифікації біомедичних зображень;
- вдосконалити модель опису архітектур нейронних мереж;
- розробити засоби для автоматичного синтезу архітектур згорткових і генеративно-змагальних нейронних мереж;
- розробити засоби генерування та класифікації біомедичних зображень;
- розробити структуру бази даних для зберігання синтетичних зображень.

Об'єкт дослідження — процеси опрацювання, зберігання, синтезу, класифікації біомедичних зображень на основі глибоких нейронних мереж.

Предмет дослідження — методи, моделі та засоби опрацювання, зберігання, синтезу, класифікації біомедичних зображень на основі глибоких нейронних мереж.

Методи дослідження. Для розв'язання поставлених задач використано основні положення математичного аналізу для задачі опису нейронних мереж, методи теорії імовірностей і математичної статистики для оптимізації структур нейронних мереж і оброблення результатів комп'ютерних експериментів , теорія нейронних мереж, методи глибокого машинного навчання для проектування архітектур згорткових та генеративно-змагальних нейронних мереж, бібліотеки та фреймворки машинного навчання для програмної реалізації системи синтезу та класифікації біомедичних зображень.

Наукова новизна отриманих результатів.

У результаті проведеного дисертаційного дослідження здобувачем *одержано такі наукові результати*:

Вперше розроблено:

- метод автоматичного синтезу архітектур згорткових нейронних мереж для класифікації біомедичних зображень, який за рахунок використання фаз

мікропошуку та макропошуку забезпечує створення архітектур нейромереж з підвищеною точністю класифікації біомедичних зображень;

- метод автоматичного синтезу архітектур генеративно-змагальних нейронних мереж для задач генерування біомедичних зображень, який за рахунок використання механізмів самоуваги в генераторі і дискримінаторі та синтезу зображень за мітками забезпечує підвищення якості синтезованих зображень;

вдосконалено:

- метод генерування та класифікації біомедичних зображень, який за рахунок використання методів автоматичного синтезу архітектур згорткових і генеративно-змагальних нейронних мереж забезпечив розширення та доповнення навчальної вибірки біомедичних зображень для навчання згорткових нейронних мереж;
- модель опису архітектур нейронних мереж, яка за рахунок використання теоретико-множинного підходу, забезпечила формалізацію представлення згорткових і генеративно-змагальних нейронних мереж.

Практичне значення одержаних результатів. Використання розроблених засобів забезпечує автоматичний синтез архітектур згорткових і генеративно-змагальних нейронних для задач класифікації та генерування біомедичних зображень. Розроблені засоби генерування та класифікації біомедичних зображень розширюють та доповнюють навчальні набори біомедичних зображень для навчання згорткових нейронних мереж.

Розроблені структури бази даних забезпечують надійне зберігання синтезованих зображень та архітектур генеративно-змагальних нейронних мереж, використання яких забезпечує генерування нових біомедичних зображень для підвищення точності навчання згорткових нейронних мереж.

Розроблена архітектура програмного засобу з використанням сучасних технології в сфері проектування та розробки програмного забезпечення дає

можливість легко масштабувати систему в майбутньому відповідно до навантаження та обсягів даних, забезпечує високий рівень безпеки інформації.

Розроблений програмний засіб для синтезу та класифікації біомедичних зображень з використанням мов програмування Python і Typescript забезпечує розширення навчальних наборів даних на основі генеративно-змагальних нейронних мереж з подальшим їх використанням для навчання класифікаторів на базі згорткових нейронних мереж.

Результати дисертаційної роботи впроваджено у ТзОВ «Інститут біомедичних технологій» (акт впровадження від 20.05.2024, додаток В).

Результати дисертаційної роботи використано при підготовці таких дисциплін: «Дослідження комп'ютерних систем штучного інтелекту», «Методи розпізнавання зображень і комп'ютерний зір», «Технології машинного навчання», «Теоретичні основи штучного інтелекту» на кафедрі комп'ютерної інженерії Західноукраїнського національного університету (акт впровадження від 05.06.2024, додаток Г).

Особистий внесок здобувача. Основні положення та результати дисертаційної роботи одержані автором самостійно. Особисто здобувачеві належать такі наукові результати: аналіз та порівняння архітектур генеративно-змагальних нейронних мереж для синтезу біомедичних зображень [3], [12]; аналіз та порівняння алгоритмів навчання нейронних мереж [13], методів оптимізації гіперпараметрів нейронної мережі [8] та метрик для оцінки генеративно-змагальних нейронних мереж [4]; розробка методів для синтезу архітектур згорткових та генеративно-змагальних нейронних мереж [6], [9], [15]; розробка алгоритмів та методів класифікації і синтезу біомедичних зображень та архітектур згорткових і генеративно-змагальних нейронних мереж [2], [5], [10], [11].

Апробація результатів дисертації. Результати дисертаційної роботи доповідалися автором особисто на семінарах та конференціях: наукових семінарах кафедри комп'ютерної інженерії Західноукраїнського національного

університету (2020-2023 рр.), кафедри автоматизованих систем управління Національного університету «Львівська політехніка» (2023-2024 рр.), Міжнародних науково-практичних конференціях «Informatics & Data-Driven Medicine IDDM» (Львів, 2022-2023 рр.), «Інтелектуальні комп'ютерні системи та мережі» (Тернопіль, 2020), «Економічний і соціальний розвиток України в XXI столітті: національна візія та виклики глобалізації» (Тернопіль, 2023 р.).

Структура та обсяг дисертації. Дисертація складається із вступу, чотирьох розділів, висновків та списку використаних джерел із 122 найменувань. Повний обсяг дисертації складає 196 сторінок, основний зміст викладено на 141 сторінці, де наведено 83 рисунки та 24 таблиці, 7 додатків.

РОЗДІЛ 1. АНАЛІЗ НЕЙРОМЕРЕЖЕВИХ МЕТОДІВ СИНТЕЗУ ЗОБРАЖЕНЬ

У розділі проведено аналіз біомедичних зображень (цитологічних, гістологічних та імуногістохімічних). Також проведено аналіз та порівняння класичних архітектур ГЗМ для задач синтезу біомедичних зображень, проведено аналіз та порівняння алгоритмів навчання нейронних мереж на основі градієнтного спуску, проаналізовано та здійснено порівняння метрик для оцінки подібності синтетичних і реальних зображень. Сформульовано задачі дисертаційного дослідження.

Результати розділу опубліковано у працях автора [3], [4], [12], [13].

1.1 Аналіз біомедичних зображень

В медицині та біології біомедичні зображення використовуються для візуалізації внутрішніх структур тіла, діагностики захворювань і дослідження біологічних процесів. Багато різних технологій та технічних засобів, таких як комп'ютерна томографія, магнітно-резонансна томографія, ультразвук тощо, можуть бути використані для отримання цих зображень. Вони дозволяють лікарям вивчати анатомію органів, виявляти патології та вирішувати прикладні завдання в медицині та біології. Загалом з поміж всіх біомедичних зображень можна виділити такі класи зображень: цитологічні, гістологічні, імуногістохімічні.

Цитологічні зображення відіграють важливу роль у діагностиці та оцінці різних злоякісних новоутворень. Вони отримуються за допомогою тонкоголкової аспірації або інших методів взяття цитологічних зразків. Цитологічні зображення в основному складаються з окремих клітин або невеликих скупчень клітин. Мікроскопічне дослідження цих зображень дозволяє ідентифікувати аномальні клітини, оцінити клітинну атипію та визначити походження пухлини. Дані зображення використовуються для вивчення структури клітин, їхнього ядра та цитоплазми.

Оцінка цитологічних зображень включає оцінку клітинних характеристик, включаючи розмір клітини, форму, ядерно-цитоплазматичне співвідношення, структуру ядерного хроматину та наявність ядерець. Ці ознаки надають цінну інформацію для розрізнення доброякісних і злоякісних клітин. Наприклад, злоякісні клітини часто демонструють збільшені ядра, неправильні контури, грубий хроматин і помітні ядерця, тоді як доброякісні клітини зазвичай мають однорідні ядра та тонкий хроматин. Також виявлення гормональних рецепторів (рецепторів естрогену та прогестерону) у цитологічних зразках молочної залози може допомогти в діагностиці раку молочної залози та прийняти рішення щодо лікування. Приклад цитологічних зображень показано на рисунку 1.1.

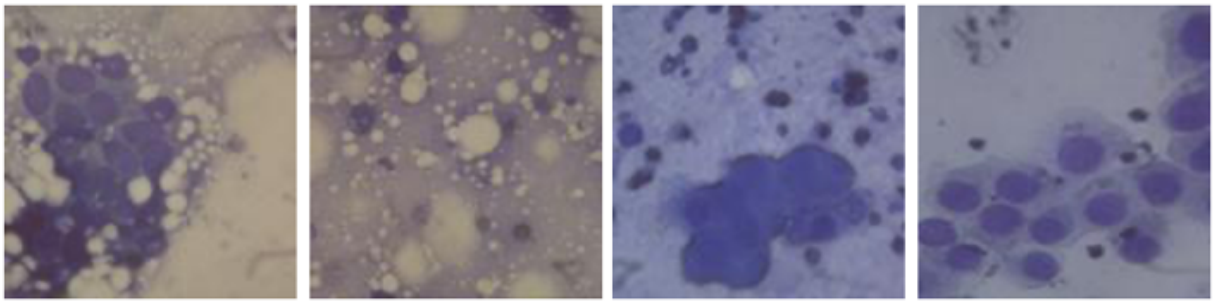


Рисунок 1.1. Приклад цитологічних зображень

Гістологічні зображення — це мікроскопічні зображення тканин, отримані із зрізів самих тканин. Дані зображення дозволяють вивчати структуру тканин, виявляти патологічні зміни або пухлини [37]. Клітинна морфологія відіграє важливу роль у інтерпретації гістологічних зображень [38]. Характеристики ядра, цитоплазми та розташування клітин надають інформацію для діагностики та класифікації різних пухлин. Аномалії розміру, форми, структури хроматину та наявності ядерець можуть вказувати на злоякісність. Розрізнення доброякісних і злоякісних клітин на гістологічних зображеннях має вирішальне значення для точної діагностики та планування лікування. Приклад гістологічних зображень показано на рисунку 1.2.

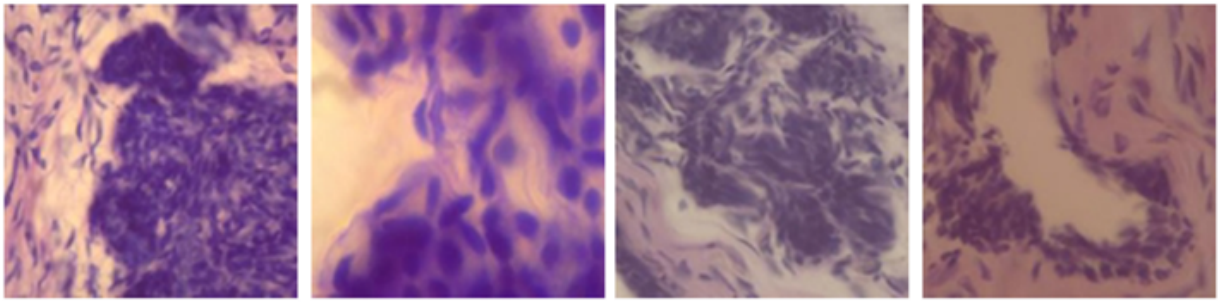


Рисунок 1.2. Приклад гістологічних зображень

Імуногістохімія — потужний сучасний метод, який використовується для ідентифікації специфічних білків або антигенів у гістологічних зразках. Імуногістохімія допомагає визначити наявність або відсутність певних білків у тканинах, що може бути корисним для діагностики пухлин або інших захворювань. Імуногістохімічне фарбування передбачає використання специфічних антитіл, які зв'язуються з антигенами в зрізах тканини. Візуалізація комплексу антитіло-антиген допомагає ідентифікувати специфічні білки або маркери, які допомагають у класифікації та діагностиці пухлини. Окрім класифікації пухлин, імуногістохімія допомагає визначити прогностичні фактори раку. Певні біомаркери, такі як рецептори гормонів (наприклад, рецептори естрогену та прогестерону при раку молочної залози), рецептор людського епідермального фактора росту 2 (HER2) при раку молочної залози та шлунка, мають значний вплив на рішення щодо лікування.

Імуногістохімія дозволяє розрізнити різні підтипи пухлин, наприклад розрізнити аденокарциному легені та плоскоклітинний рак. Аналізуючи моделі поведінки специфічних маркерів, лікарі можуть поставити точніші діагнози та підібрати відповідні варіанти лікування. Приклад імуногістохімічних зображень показано на рисунку 1.3.

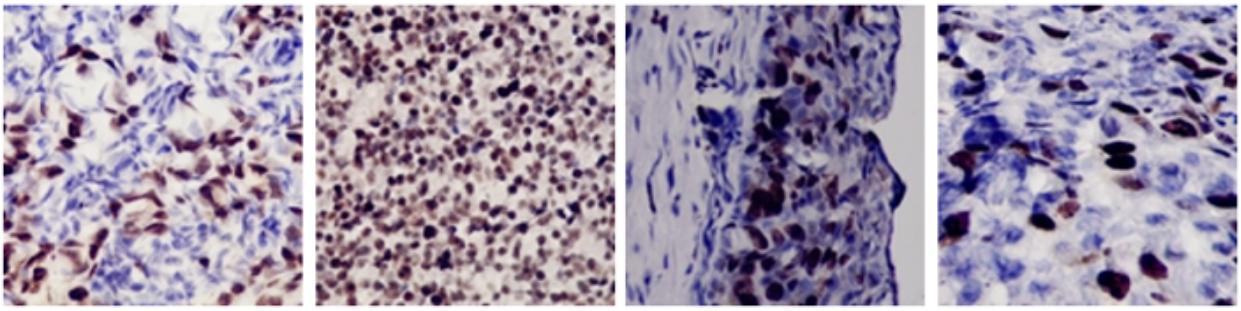


Рисунок 1.3. Приклад імуногістохімічних зображень

Таким чином, імуногістохімічний аналіз гістологічних зображень значно підвищує точність діагностики в онкології. Завдяки цьому аналізу можна отримати інформацію щодо класифікації пухлин, визначення їх походження, прогнозування розвитку хвороби та визначення методів лікування. Поєднання гістологічної оцінки з імуногістохімічним фарбуванням допомагає онкологам приймати обґрунтовані рішення щодо ведення пацієнтів і стратегії їх лікування.

Усі три типи зображень важливі для медичної діагностики, дослідження тканин та виявлення патологічних змін. Вони допомагають лікарям отримати детальну інформацію про клітини та тканини, що дозволяє вчасно виявляти та лікувати захворювання.

1.2 Аналіз програмних засобів для синтезу зображень

Програми для синтезу зображень відносяться до області генеративного штучного інтелекту — це підгалузь штучного інтелекту, яка використовує машинне навчання для створення нового контенту, такого як текст, зображення, музика чи відео. Генеративний штучний інтелект використовує великі набори даних для виявлення певних стилів, структур або шаблонів у вхідних даних. Потім ці знання використовуються для створення нового контенту, який повторює оригінальний формат чи стиль. Такі системи мають на меті не лише повторення існуючого контенту, але й доповнення його новими елементами чи концепціями.

Серед відомих програм для синтиезу зображень є такі: ChatGPT та Stable Diffusion [39], [40]. ChatGPT, розроблений компанією OpenAI, — це вдосконалена мовна модель призначена для розуміння і генерування людського тексту на основі отриманих даних, що робить її корисною для створення контенту і вирішення проблем на основі людської мови. В свою чергу Stable Diffusion — це сучасна модель синтезу зображень, яка генерує реалістичні зображення на основі текстових описів.

Крім того, генеративний штучний інтелект робить значний внесок у підвищення продуктивності та автоматизацію в різних галузях. Наприклад, генератори коду на основі штучного інтелекту в розробці програмного забезпечення можуть автоматизувати повторювані завдання з програмування, пропонувати фрагменти коду і налагоджувати вже написаний код, що прискорює цикли розробки програм та підвищує продуктивність (наприклад, сервіси Copilot, Adrenaline і т.д). Генеративний штучний інтелект можна використовувати не лише для роботи з текстом і в програмуванні. Наприклад, ChatGPT, також може створювати зображення на основі текстових описів. Однак ці моделі часто є закритими та приватними і доступ до їх функцій може бути обмеженим, що обмежує їх використання в незалежних дослідженнях або проектах з відкритим вихідним кодом. Це обмеження підкреслює, наскільки важливо враховувати приватні технології та доступність при впровадженні штучного інтелекту в дослідницькі проекти.

В межах даної дисертаційної роботи проведено декілька експериментів із технологією Stable Diffusion. Експерименти зосереджені на властивостях програми, що пов'язані із синтезом зображень. Хоча Stable Diffusion добре відомий своєю здатністю генерувати художні зображення на основі текстових описів, результати експериментів показали, що модель працює менш ефективно, коли її використовують для створення біомедичних зображень.

На рисунку 1.4 показано приклади цитологічних зображень з роздільною здатністю 512×512, синтезованих за допомогою моделі Stable Diffusion, яку

було навчено на цитологічних зображеннях. Незважаючи на те, що на зображеннях зберігається кольорова гама, форма самих клітин не відповідає дійсним зображенням.

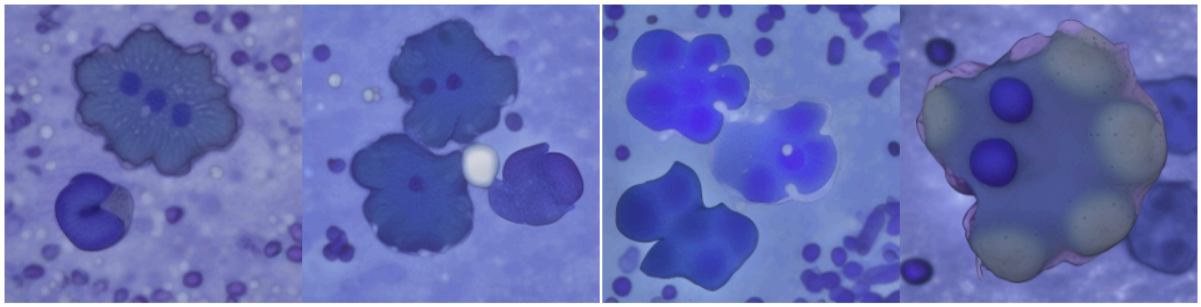


Рисунок 1.4. Зображення синтезовані Stable Diffusion

Модель намагається додати елемент творчості у синтетичні зображення, що є недоцільним у цьому дослідженні. Це підкреслює, що хоча ці моделі мають інноваційний творчий потенціал, їхнє застосування може сильно відрізнятися залежно від ситуації та індивідуальних вимог конкретної задачі.

Проведемо аналіз сучасних засобів, які використовуються для побудови нейронних мереж.

TensorFlow — це широко використовувана бібліотека машинного навчання з відкритим вихідним кодом. Вона пропонує високорівневий інтерфейс під назвою TensorFlow Keras, який спрощує процес побудови та навчання глибоких нейронних мереж. За допомогою TensorFlow можна використовувати попередньо навчені моделі, такі як ГЗМ для задач синтезу зображень. TensorFlow також надає функціональні можливості для доповнення даних та проведення різних маніпуляцій із зображеннями [41].

PyTorch — інша сучасна бібліотека для машинного навчання, яка широко використовується для побудови та навчання нейронних мереж. Дана бібліотека також включає попередньо навчені моделі, такі як ЗНМ, а також утиліти для попередньої обробки зображень. PyTorch відомий своєю простотою та потужною підтримкою спільноти розробників та дослідників [42].

Бібліотека OpenCV в першу чергу відома своїм застосуванням для вирішення задач комп'ютерного зору, але також пропонує можливості для синтезу зображень. OpenCV надає різні функції маніпулювання зображеннями, включаючи геометричні перетворення, змішування та фільтрацію. Ці функції можуть бути використані для створення зображень і їх композицій. Крім того, OpenCV надає інструменти для генерації шуму, синтезу текстур і виявлення контурів, які можуть бути корисними у робочих процесах синтезу зображень [43].

Іншою бібліотекою в області комп’ютерного зору є PIL —потужна бібліотека для обробки зображень та маніпуляцій з ними написана мовою програмування Python. Хоча сама бібліотека не зосереджена безпосередньо на синтезі зображень, вона надає функції для створення нових зображень з нуля, наприклад, малювання фігур, додавання тексту та генерування градієнтів. Ці функції можна комбінувати для синтезу власних зображень [44].

Перспективним рішенням для побудови та реалізації методів синтезу зображень на основі моделей нейронних мереж є платформа PyTorch з відкритим вихідним кодом. Нижче наведено переваги даної платформи над іншими:

– динамічний обчислювальний граф забезпечує більшу гнучкість; на відміну від бібліотек (TensorFlow), де архітектуру нейронної мережі неможливо динамічно змінювати після компіляції, PyTorch дозволяє будувати обчислювальний граф в реальному часі, дозволяючи вносити зміни в архітектуру моделі під час виконання коду програми; ця функція особливо корисна для досліджень та експериментів, оскільки вона спрощує ітеративний процес розробки моделі;
– простий у використанні програмний інтерфейс; інтуїтивно зрозумілий код робить розробку моделей та експерименти більш простими; код, як правило, більш стислий і читабельний у порівнянні з іншими бібліотеками;

– велика спільнота користувачів та екосистема; бібліотека має екосистему з великою колекцією попередньо навчених моделей, користувацьких шарів та утиліт, створених спільнотою;
– бібліотека широко використовується у дослідницькій спільноті завдяки своїй гнучкості та простоті використання; багато передових дослідницьких робіт і моделей випускаються з реалізаціями коду на основі PyTorch, що полегшує відтворення і розвиток існуючих напрацювань;
– інтеграція з іншими бібліотеками Python для наукових обчислень, обробки даних та візуалізації; PyTorch легко інтегрується з бібліотеками, такими як NumPy, SciPy та Matplotlib, дозволяючи використовувати їх функціональні можливості;
– завдяки таким інструментам, як TorchScript та Open Neural Network Exchange, бібліотека надає можливість експортувати навчені моделі нейронних мереж для використання у різних середовищах, включаючи мобільні пристрої та веб-додатки; PyTorch також пропонує оптимізовані бібліотеки, такі як TorchVision, для задач комп'ютерного зору, які полегшують ефективне розгортання моделей для аналізу зображень.

1.3 Архітектури генеративно-змагальних нейронних мереж

Сучасним підходом до синтезу зображень є застосування ГЗМ, які складаються із двох нейронних мереж (генератора та дискримінатора). Одна з них генерує зображення, а інша перевіряє їх достовірність [26]. На даний момент можливості ГЗМ все ще вивчаються багатьма дослідниками [45], оскільки дані мережі вже було застосовано до великої кількості задач, як-от синтез аудіо [46], [47], відео [48], [49], тексту [50] та зображень [51]. Більшість досліджень у цій галузі спрямовані на підвищення якості синтезованих зображень. Дослідники створюють нові архітектури та вдосконалюють уже

відомі алгоритми [27], [52]. Нейронні мережі застосовуються і в медицині. Останні дослідження показали, що штучні нейронні мережі можуть досягати кращих результатів у сегментації та класифікації, ніж люди [53]. Однак сучасні класифікатори вимагають багато навчальних даних. Оскільки процес збору медичних даних може бути складним через вплив багатьох факторів (час, ціна, доступність тощо), ГЗМ в основному використовуються для розширення наборів навчальних даних [54], [55]. У цьому випадку також можна збільшити кількість вибірок у наборі даних за допомогою афінних перетворень [56]. Але цей метод не підходить, якщо необхідно згенерувати велику кількість даних, оскільки створені таким чином зображення не є абсолютно новими, а є лише перетвореними версіями існуючих.

У роботі [57] дослідники використовують ГЗМ для синтезу зображень ядер клітин та їх масок для подальшої сегментації. Вони ділять процес навчання на два різних етапи. На першому кроці вони навчають генератор синтезувати бінарні маски з вхідного вектора шуму. На другому кроці автори використовують умовну генеративну змагальну мережу для синтезу зображень на основі згенерованих масок і вхідного вектора. У підсумку автори отримують два генератори. Один для синтезу маски, інший для синтезу зображення на основі маски.

Автори роботи [58] використовують ГЗМ для синтезу зображень уражених тканин печінки з роздільною здатністю 64×64 пікселі. Експериментальні результати класифікації показали, що точність зросла з 77% до 85% при використанні навчального набору даних, який був розширений синтезованими зображеннями.

Серед переваг використання ГЗМ мереж можна виділити такі: реалістичність (вони можуть генерувати дуже реалістичні зображення, які часто неможливо відрізнити від реальних); різноманітність (вони можуть генерувати високоякісні та унікальні зображення, які не зустрічаються в наборі даних для

навчання); гнучкість (ГЗМ можна використовувати для генерування різних типів зображень, таких як фотографії, портрети, пейзажі тощо).

Основними недоліками є такі: архітектурна складність (архітектури можуть бути складними для навчання та імплементації); стабільність навчання (навчання ГЗМ є досить складним процесом і може бути нестабільним, іноді воно може призвести до небажаних результатів); обчислювальні ресурси (навчання ГЗМ може бути обчислювально витратним, особливо для складних моделей та для зображень високої роздільної здатності).

Типова архітектура ГЗМ мережі складається з двох різних нейронних мереж, які «змагаються» одна проти одної. Перша мережа називається дискримінатором (позначається літерою D). Завданням дискримінатора є перевірка правдоподібності зображень — відрізнити реальні зображення від синтетичних. Реальні зображення є даними з навчальних вибірок. Дискримінатор приймає зображення як вхідні дані, а на виході показує ймовірність того, що зображення реальне. Зазвичай дискримінатором є простий бінарний класифікатор. Друга мережа називається генератором (позначається літерою G). Завданням генератора є синтез нових даних. Генератор бере випадковий вектор певної розмірності з певного латентного простору (зазвичай із нормального розподілу) і намагається перетворити його на реальне зображення [27], [59].

Базуючись на певній функції втрат, модель дискримінатора повинна максимізувати ймовірність реальних даних і мінімізувати ймовірність синтетичних. Аналогічно, генератор повинен синтезувати зображення, які максимально наближені та подібні до реальних, щоб дискримінатор класифікував їх як реальні дані. Як висновок, генеративна модель намагається максимізувати ймовірність синтетичних зображень. Алгоритм навчання ГЗМ виконує оптимізацію параметрів як дискримінатора, так і генератора. Метою навчання є мінімізація наступної функції втрат $V(G, D)$, де E – математичне сподівання:

$$min_G max_D V(G,D) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[\log\Big(1 - D\big(G(z)\big)\Big)]. \quad (1.1)$$

В ідеальному випадку дискримінатор навчається доти, поки не досягне свого оптимального значення, яке залежить від конкретного генератора, а генератор навчається після дискримінатора. Але на практиці дискримінатор навчають тільки певну кількість ітерацій, тоді генератор оновлюється разом із дискримінатором. Крім того, як хороша альтернатива для генератора замість $min_G \log\,(1 - D(G(z)))$ використовується рівняння $max_G \log\, D(G(z))$ [27].

Класичні ГЗМ мають свої недоліки. Перша проблема — затухання градієнтів [61]. Оскільки генератор має зворотній зв'язок із дискримінатором, він може оновлювати свої параметри лише через нього. Проблема затухання проявляється в тому, що градієнти стають настільки малими, що практично не впливають на параметри генератора і процес навчання припиняється. Ця проблема виникає, коли помилка дискримінатора швидко сходиться до нуля. Це також може статися, коли модель дискримінатора набагато потужніша за генератор. Для уникнення цієї проблеми варто при побудові архітектури збалансувати складність моделей дискримінатора та генератора.

Усі архітектури ГЗМ можна розділити на 3 основні типи — повнозв'язні, згорткові та умовні ГЗМ. Поєднуючи ці три типи і використовуючи різні оптимізації, можна отримувати нові архітектури. На даний момент існує багато різних архітектур. Розглянемо найпоширеніші з них.

DCGAN була вперше використана та запропонована в 2014 році [26]. Автори використали ЗНМ як для генератора, так і для дискримінатора. Основна перевага використання ЗНМ перед повнозв'язною архітектурою полягає в тому, що ЗНМ приймає зображення як вхідні дані. Крім того, сама архітектура побудована більш оптимально. Такий підхід дозволяє значно зменшити кількість параметрів, час навчання та підвищити точність таких мереж [62].

Сучасна архітектура ГЗМ мережі може складатися з шарів згортки, субдискретизації (пулінгу) та активації. Найпопулярнішою функцією активації, яка часто використовується в мережах ЗНМ і ГЗМ, є ReLU. Принцип роботи дуже простий і полягає в заміні від'ємних значень на нуль. Генератор і дискримінатор використовують різні функції активації на фінальних шарах: тангенс і сигмоїду відповідно. Загальний приклад DCGAN показаний на рисунку 1.5.

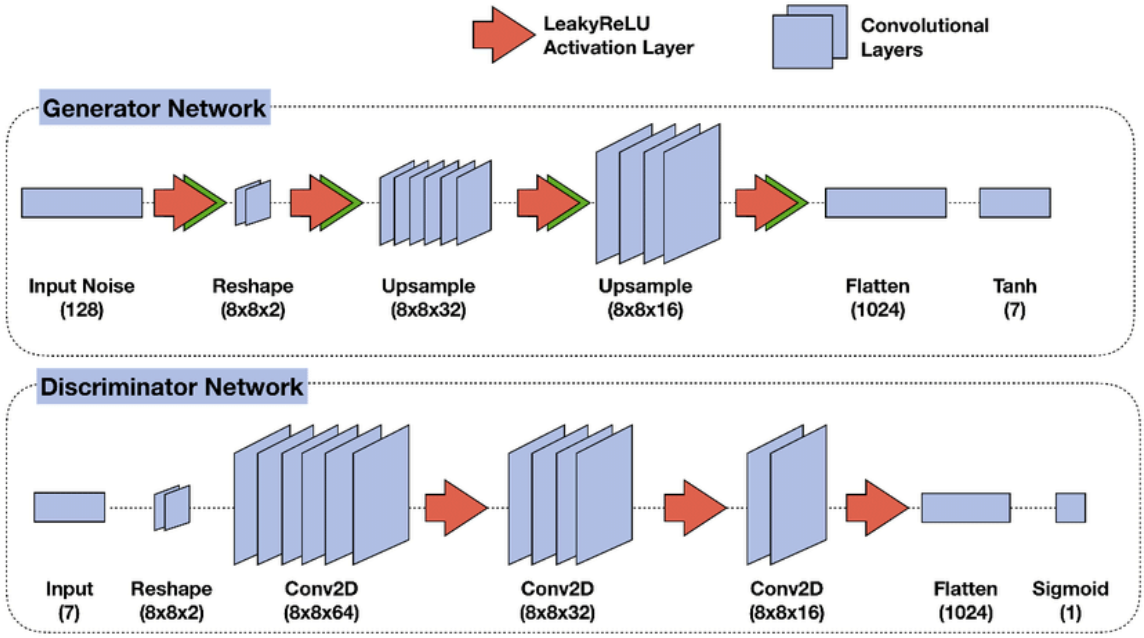


Рисунок 1.5. Загальна архітектура GAN мережі

WGAN і WGAN-GP. Іншою з поширених проблем при навчанні ГЗМ є так званий колапс [63]. Коли це відбувається, генератор починає синтезувати однакові зображення незалежно від вхідних даних. Архітектура WGAN дозволяє частково позбутися цієї проблеми [64]. Нововведенням у цій архітектурі є застосування нової функції втрат. Перевагою такої архітектури є те, що дискримінатор замінений критиком (рисунок 1.6).

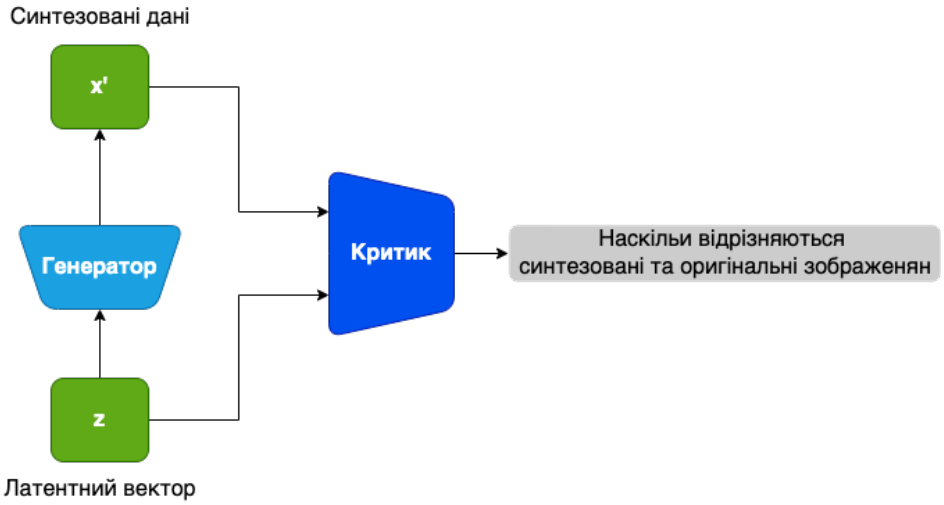
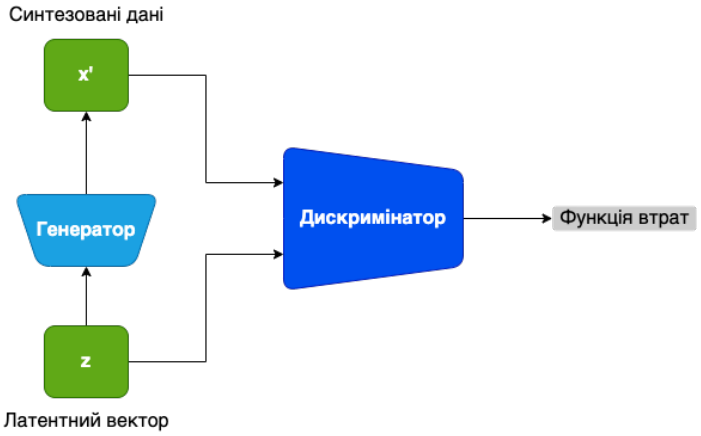


Рисунок 1.6. Архітектура WGAN та WGAN-GP

Таким чином дискримінатор більше не намагається класифікувати зображення як реальне чи синтетичне, а показує певну похибку того, наскільки синтетичне зображення відрізняється від реального. Процес навчання мінімізує цю похибку. Проте ця архітектура також має свої недоліки (досить тривалий час навчання, навіть для зображень з низькою роздільною здатністю). Тому автори запропонували нову модель WGAN-GP, яка є вдосконаленою версією WGAN і використовує градієнтний штраф як метод запобігання перенавчанню [65]. Незважаючи на ці переваги, на практиці WGAN призводить до повільного процесу оптимізації.

BGAN. Для кожного фіксованого генератора існує унікальний і оптимальний дискримінатор [28]. Це показує, що генератор є оптимальним, коли дискримінатор видає ймовірність 0,5 для всіх прикладів, створених із випадкового розподілу. Однак на практиці отримати ідеальний дискримінатор практично неможливо. Тому автори пропонують внести зміни у вихідну функцію втрат генератора так, щоб дискримінатор виводив імовірність 0,5 для всіх синтетичних даних [66].

Архітектурно ця мережа така ж, як оригінальна DCGAN (рисунок 1.7), але з дещо зміненою функцією втрат.



Риснок 1.7. Структура BGAN

BEGAN. У цій мережі дискримінатором є варіаційний автоенкодер, який кодує вхідні зображення у приховані вектори, а потім декодує їх назад у зображення. Таким чином, дискримінатор повертає помилку реконструкції між вхідним і відновленим зображенням замість значення ймовірності [67]. Основна ідея полягає в тому, що однакові значення помилок реконструкції для реальних і синтетичних зображень зрештою призводять до однакового розподілу реальних і синтетичних даних. Генератор використовує ту саму архітектуру, що й декодер дискримінатора. Базова архітектура BEGAN показана на рисунку 1.8.

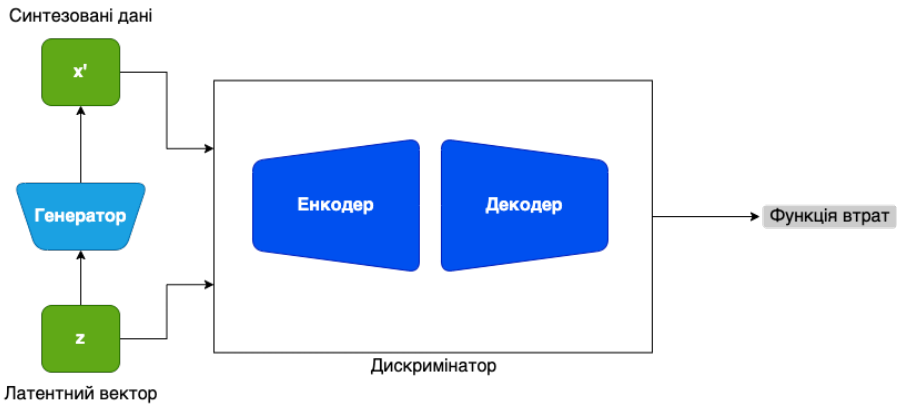


Рисунок 1.8. Структура BEGAN

BigGAN. Дана архітектура розроблена, щоб об'єднати найкращі підходи з попередніх досліджень [68]. Архітектура BigGAN зосереджена на масштабуванні моделі GAN для створення якісніших зображень і зображень із

вищою роздільною здатністю. Для цього використовується більше параметрів моделі (більше згорткових шарів із великою кількістю гіперпараметрів), більший розмір пакету, архітектурні зміни. Як наслідок, BigGAN мережа здатна генерувати зображення, такі як 256×256 і 512×512 пікселів. Незважаючи на всі вдосконалення процесу навчання, які зосереджені на деяких змінах функції втрат, основним недоліком усе ще є час навчання і великі обчислювальні ресурси.

StyleGAN. Ця архітектура пропонує багато змін в моделі генератора [69]. В результаті модель здатна не тільки генерувати високоякісні зображення (роздільна здатність до 1024 пікселів), але й пропонує контроль над стилем зображення на різних рівнях (від деталей до більш загальних рис). Ця модель здебільшого використовувалася для синтезу людських облич.

Порівняльну характеристику описаних архітектур наведено в таблиці 1.1.

Таблиця 1.1. Порівняльна характеристика архітектур ГЗМ

Архітектура	Характеристика	Роздльна здатність
DCGAN	Низька якість зображення, але відносно малий час навчання. Відсутність потреби у великих обчислювальних ресурсах.	64
WGAN	Краща якість зображення порівняно з DCGAN. Дуже повільний тренувальний процес.	64
WGAN-GP	Краща якість зображення, відсутність колапсу. Дуже повільний час навчання (до кількох днів).	будь-яка
BGAN	Краща стабільність навчання порівняно з DCGAN, але зображення низької якості.	64
BEGAN	Вища якість зображення порівняно з DCGAN, але дуже повільне навчання.	будь-яка
BigGAN	Висока якість зображення. Повільний час навчання. Великі обчислювальні ресурси.	128, 256, 512
StyleGAN	Дуже висока якість і розмір зображення. Можливість контролювати стиль створеного зображення від деталей до більш загальних характеристик. Дуже повільний час навчання (до кількох тижнів). Великі обчислювальні ресурси.	1024

1.4 Алгоритми навчання нейронних мереж

Навчання нейронної мережі передбачає оптимізацію параметрів для досягнення мінімального значення помилки. Оптимізацію параметрів нейронної мережі можна здійснювати різними алгоритмами, які називаються алгоритмами навчання. Алгоритми навчання поділяються на алгоритми першого або другого порядку та еволюційні алгоритми. Алгоритми першого порядку засновані на обчисленні першої похідної функції похибок. Тому ці алгоритми ще називають градієнтними алгоритмами. Алгоритми другого порядку використовують другу похідну для вибору напрямку мінімізації помилки. Також є еволюційні алгоритми оптимізації, які будуються на основі генетичних алгоритмів.

Основною метою будь-якого алгоритму навчання є мінімізація помилки навчання та оптимізація параметрів мережі. Сучасні класифікатори [70], [71] вимагають великих обсягів навчальних даних для досягнення високої точності. Сучасні алгоритми навчання нейронних мереж засновані на зворотному поширенні помилок і методі градієнтного спуску. Важливим параметром алгоритмів навчання є швидкість навчання. Цей параметр контролює, наскільки далеко потрібно рухатися в напрямку, протилежному градієнту функції, за один крок. Якщо швидкість навчання низька, час навчання нейронної мережі може значно збільшитися. Якщо швидкість навчання висока, нейронна мережа може не досягти мінімального значення помилки [72]. Формально це можна представити так:

$$\theta = \theta - \alpha \nabla J(\theta) \tag{1.2}$$

де θ відноситься до параметрів нейронної мережі, α – швидкість навчання, $\nabla J(\theta)$ – градієнт оптимізаційної функції (втрат).

Недоліком градієнтного спуску є те, що параметри мережі можна оновити лише після проходження повного навчального набору даних. Серед інших

алгоритмів градієнтного спуску виділяють стохастичний градієнтний спуск і міні-пакетний градієнтний спуск.

Стохастичний градієнтний спуск відрізняється від звичайного тим, що параметри мережі оновлюються після кожної навчальної ітерації. Тому при використанні даного алгоритму навчання параметри нейронної мережі оновлюються набагато частіше.

Міні-пакетний градієнтний спуск використовує пакети даних для оновлення параметрів [73]. Навчальний набір даних розділений на пакети однакового розміру. Потім кожен з пакетів надсилається на вхід мережі, розраховується градієнт і оновлюються параметри:

$$\theta = \theta - \alpha \nabla J(\theta; B(i)), \tag{1.3}$$

де $B(i)$ пакет навчальних прикладів.

Розглянемо різновиди методів на основі градієнтного спуску.

Adagrad. Суть цього алгоритму полягає в тому, що швидкість навчання адаптується відповідно до параметрів мережі [74]. Алгоритм встановлює нижчу швидкість навчання для параметрів, які частіше змінюються. Тоді рівняння з ітераціями матиме такий вигляд:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t, \tag{1.4}$$

де – G діагональна матриця, де кожен з діагональних елементів є сумою квадратів градієнтів за параметрами θ на всіх попередніх ітераціях, включаючи t; ϵ - це параметр з малим значенням, який запобігає діленню на 0 (зазвичай 1e-10); g – градієнт оптимізаційної функції, $\nabla J(\theta)$.

Перевагою цього алгоритму є те, що досліднику не потрібно вручну встановлювати швидкість навчання. Автори використовують значення за замовчуванням для швидкості навчання, яке становить 1,0. Недоліком

алгоритму є накопичення градієнтів від попередніх ітерацій. Це призводить до зниження швидкості навчання та незначного оновлення параметрів мережі.

Adadelta. Цей алгоритм є вдосконаленою версією попереднього алгоритму. Алгоритм Adadelta зменшує розмір матриці накопичених градієнтів до певного фіксованого значення [75].

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \tag{1.5}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t, \tag{1.6}$$

де RMS — середньоквадратичне значення, g – градієнт оптимізаційної функції, $\nabla J(\theta)$.

Перевагою цього алгоритму є відсутність необхідності встановлювати початкове значення швидкості навчання.

RMSProp. Цей алгоритм схожий на алгоритм Ададелька. Його розробив Джеффрі Хінтон. Рівняння, які описують роботу алгоритму, такі:

$$E_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{1.7}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E_t+\epsilon}} g_t. \tag{1.8}$$

RMSprop було розроблено приблизно в той же час, що й Adadelta. Ці алгоритми вирішують проблему монотонного зменшення швидкості навчання в алгоритмі Adagrad.

Adam i Adamax. На відміну від двох попередніх алгоритмів, на додаток до квадратів попередніх градієнтів g^2, алгоритм Adam також зберігає попередні градієнти g:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{1.9}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{1.10}$$

де m, v – оцінки середнього та дисперсії градієнтів відповідно [76].

Автори пропонують такі значення параметрів: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999, \epsilon = 1e-8$.

Для порівняння алгоритмів навчання на основі градієнтного спуску побудовано модель згорткової нейронної мережі. На вхід мережа приймає кольорові цитологічні зображення [77] розміром 64×64 пікселів і виводить мітку класу. Послідовність шарів наведена в таблиці 1.2.

Таблиця 1.2. Структура мережі для експерименту

Назва шару	Вихідна форма	Параметри
Input	64×64×3	
Conv	32×32×64	kernel size = 5
Batch Norm	32×32×64	
Leaky Relu	32×32×64	slope = 0,2
Conv	16×16×128	kernel size = 5
Batch Norm	16×16×128	
Leaky Relu	16×16×128	slope = 0,2
Max Pool	8×8×128	
Dropout	8×8×128	rate = 0,5
Conv	4×4×256	kernel size = 3
Batch Norm	4×4×256	
Leaky Relu	4×4×256	slope = 0,2
Conv	2×2×512	kernel size = 3
Batch Norm	2×2×512	
Leaky Relu	2×2×512	slope = 0,2
Dropout	2×2×512	rate = 0,5
Flatten	2048	
Dense	4	
Softmax	4	

Як видно з таблиці 1.2, мережа складається з кількох повторюваних блоків. Кожен блок складається з послідовності шарів згортки, пакетної нормалізації, шару активації та dropout-шару для запобігання перенавчанню

мережі. Кожен згортковий шар наполовину зменшує роздільну здатність вхідних даних.

Кількість епох навчання становить 30. Набір даних розділений на навчальний та тестовий у співвідношенні 80% до 20%. Розмір пакету дорівнює 64 зображення.

Для побудови моделі та проведення експериментів використано бібліотеку TensorFlow та мову програмування Python. Експерименти проводилися в середовищі Google Colaboratory з використанням графічного процесора Nvidia Tesla K80. Параметри кожного з оптимізаторів встановлено у значення за замовчуванням, указані в бібліотеці TensorFlow.

Порівняння алгоритмів навчання проводилося на основі часу навчання мережі, значення функції втрат і точності класифікації на тестовому наборі даних. Результати експериментів показані в таблиці 1.3 і на рисунках 1.9–1.15.

Таблиця 1.3. Результати експерименту

Назва	Час навчання (хв)	Похибка	Тестова похибка	Точність	Тестова точність
Adam	4,10	0,05	0,77	0,9797	0,7625
SGD	2,50	0,07	1,29	0,9731	0,7462
RMSprop	3,39	0,08	0,45	0,9719	0,895
Nadam	3,55	0,04	2,79	0,9822	0,66
Adadelta	4,35	0,64	0,45	0,7319	0,8425
Adagrad	3,49	0,13	0,12	0,9491	0,9488
Adamax	4,45	0,06	0,30	0,9775	0,9162

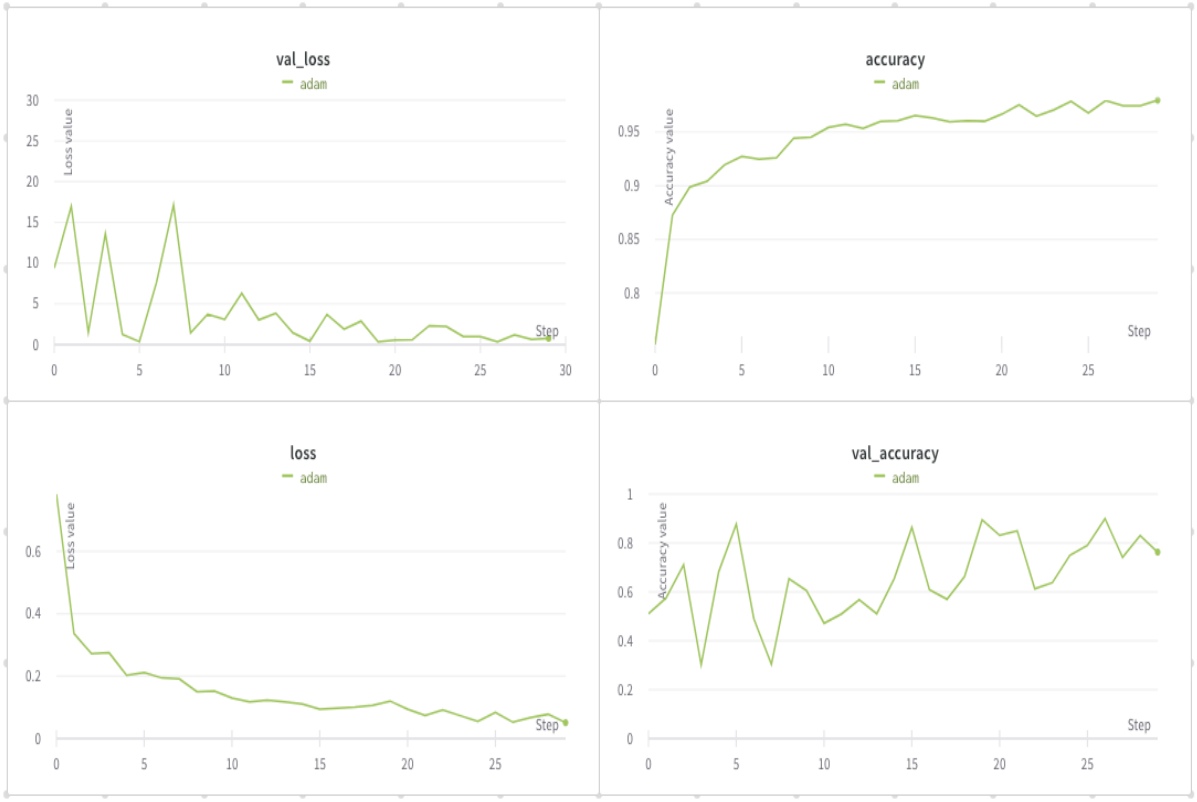


Рисунок 1.9. Результати навчання з оптимізатором Adam

На рисунку 1.9 показано графіки помилок і графіки точності на наборах навчальних і тестових даних для оптимізатора Adam. Криві точності на навчальних і тестових наборах даних показують, що мережа перенавчається. Це відбувається через те, що існує значна різниця в точності між навчальними та тестовими наборами даних. Точність на навчальному наборі даних становить ~98%, а на тестовому — ~76%.

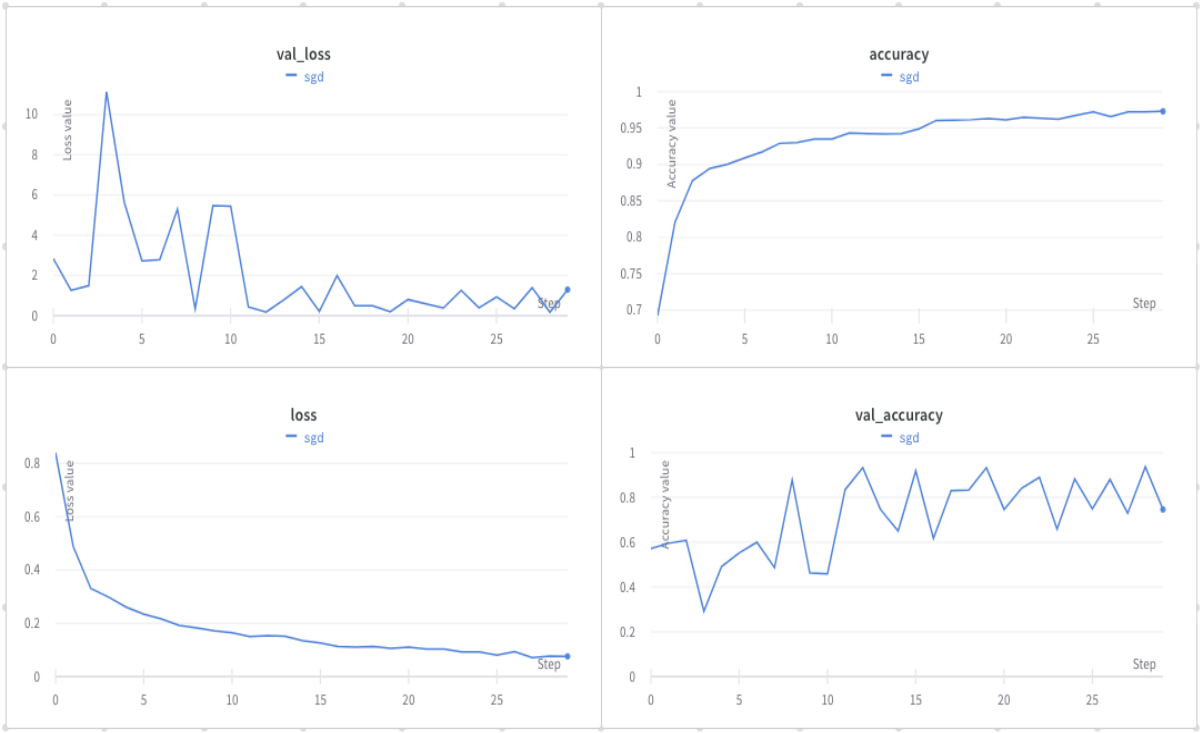


Рисунок 1.10. Результати навчання з оптимізатором SGD

На рисунку 1.10 також показано суттєву різницю між значеннями точності для навчальних і тестових наборів даних: ~97% і ~75% відповідно.

Точність класифікації для моделі з оптимізатором RMSprop на навчальних і тестових наборах даних склала ~97% і ~89% відповідно (рисунок 1.11).

Модель з оптимізатором Nadam також демонструє проблему перенавчання. Точність на навчальному і тестовому наборі даних складає ~98% і 66% відповідно (рисунок 1.12).

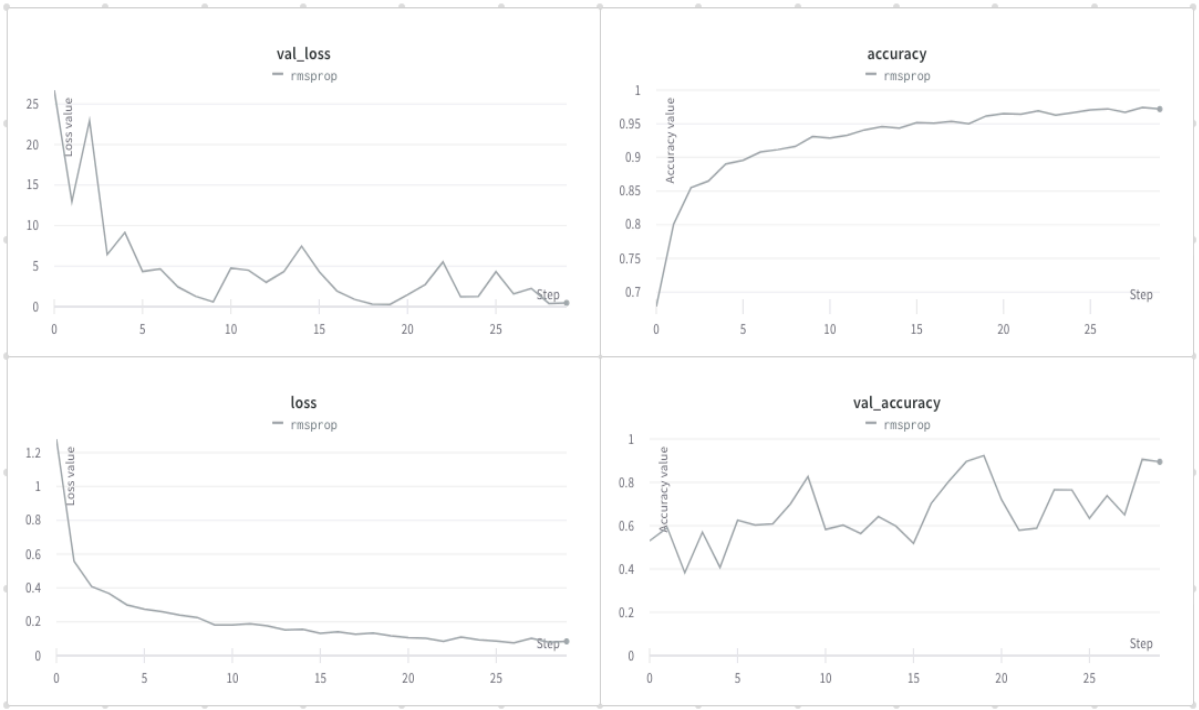


Рисунок 1.11. Результати навчання з оптимізатором RMSprop

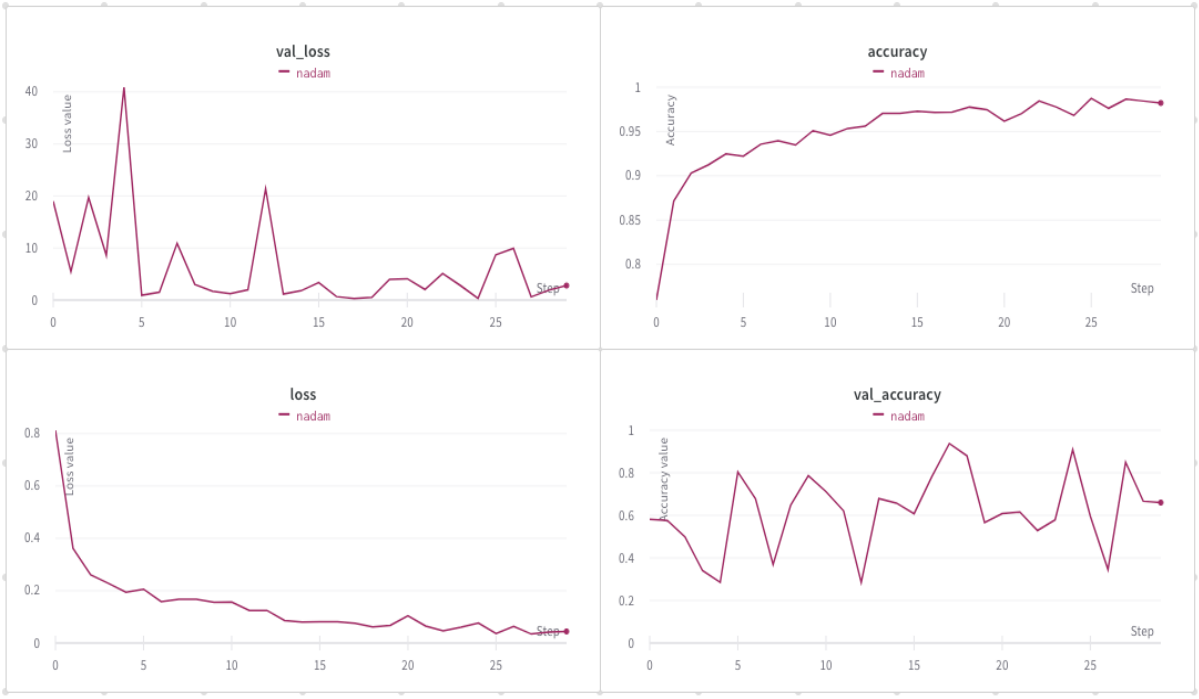


Рисунок 1.12. Результати навчання з оптимізатором Nadam

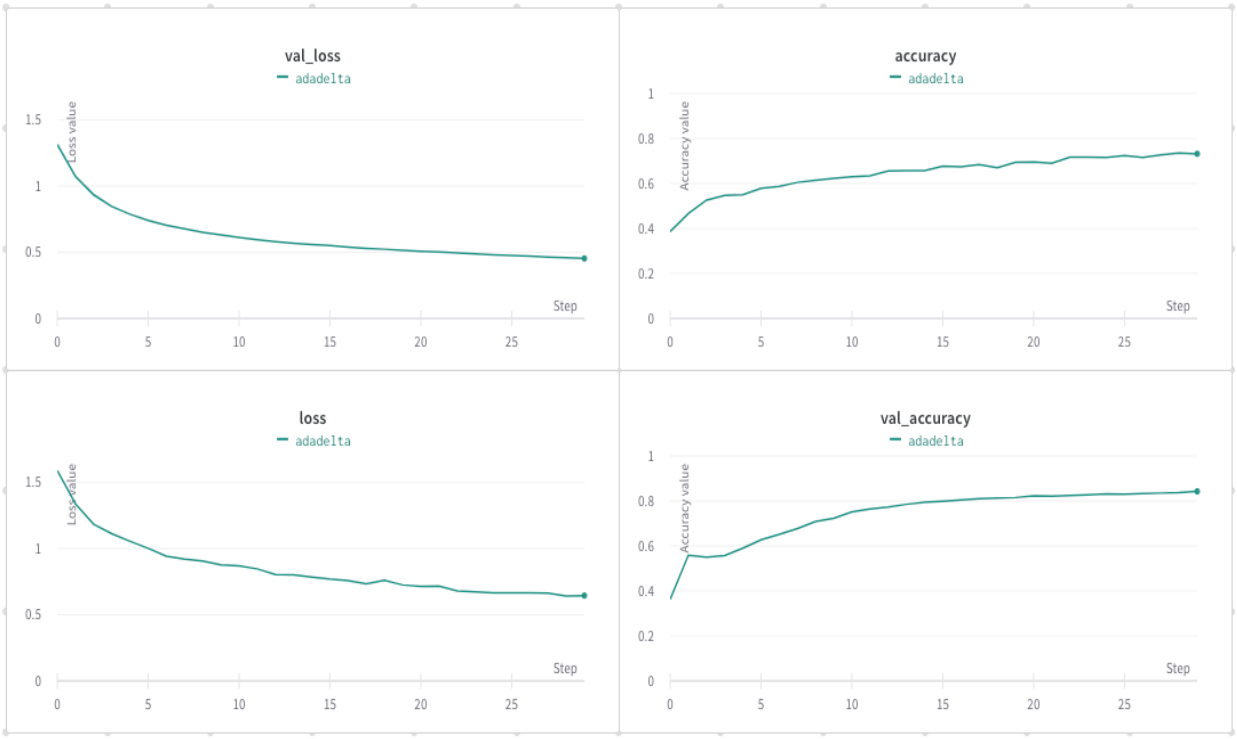


Рисунок 1.13. Результати навчання з оптимізатором Adadelta

Як видно з рисунка 1.13, графіки похибки і точності досить плавні. Однак точність класифікації на навчальному наборі даних становить лише ~73%. Це пояснюється зворотною до перенавчання проблемою. Існує кілька варіантів її вирішення, наприклад збільшення кількості епох навчання або збільшення складності використовуваної моделі.

Значення точності на навчальному і тестовому наборі даних даних майже однакові для моделі з оптимізатором Adagrad і становлять ~95% (рисунок 1.14).

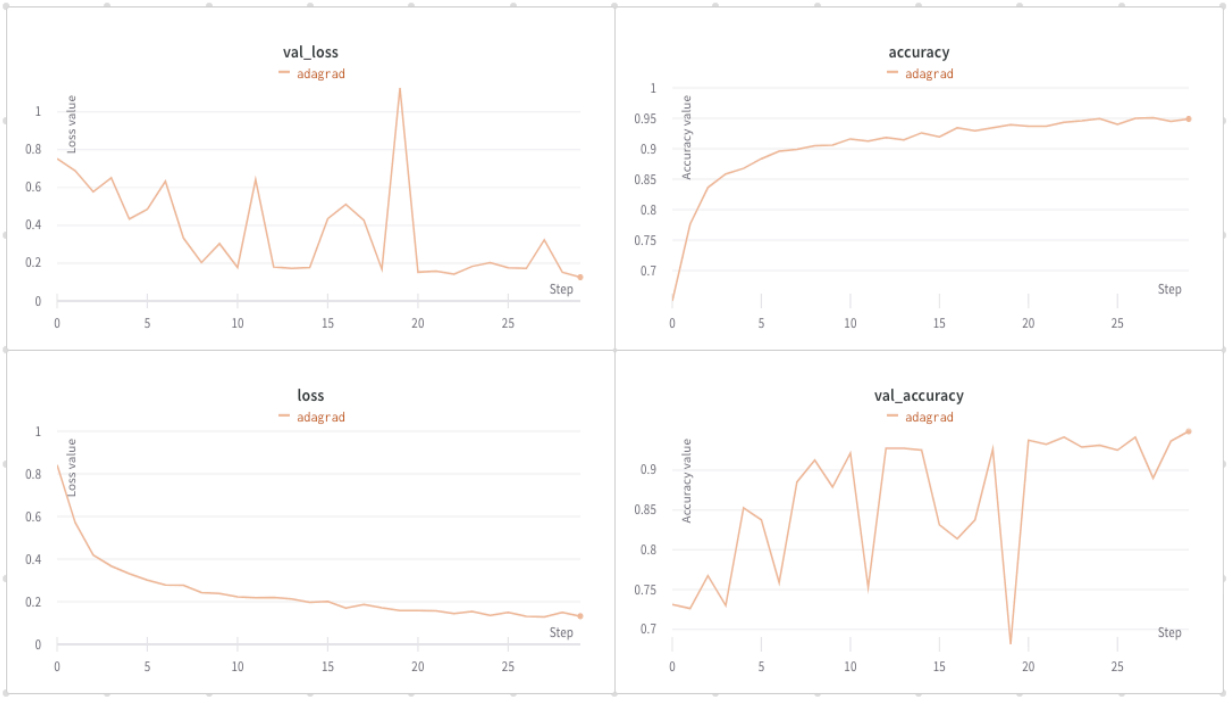


Рисунок 1.14. Результати навчання з оптимізатором Adagrad

На рисунку 1.15 показано, що криві похибок і точності на тестовому наборі даних для майже всіх моделей не гладкі, тому мережі перенавчаються. Про це свідчить значна різниця в значеннях точності на навчальному і тестовому наборі даних. Цю проблему можна вирішити спрощенням моделі нейронної мережі або збільшенням кількості зображень у навчальному наборі даних. З іншого боку, модель з оптимізатором Adadelta має зворотну проблему. Щоб вирішити цю проблему, можна застосувати декілька прийомів, наприклад збільшення складності моделі або збільшення кількості епох навчання.

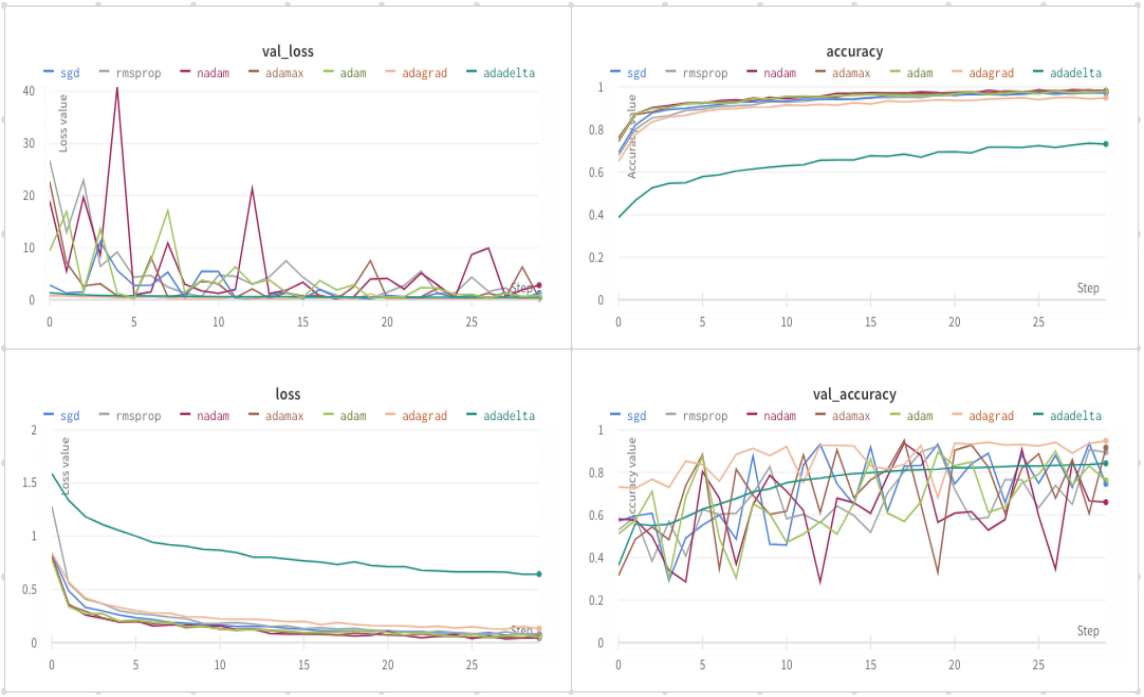


Рисунок 1.15. Візуалізація результатів усіх оптимізаторів

Порівняння різновидів градієнтного спуску проводилося за критеріями часу навчання мережі, значень функції втрат і точності класифікації на наборі даних цитологічного зображення. На основі набору цитологічних зображень і розробленої моделі згорткової нейронної мережі було обрано чотири найкращі оптимізатори за параметром val_accuracy: Adamax, Adadelta, Adagrad і RMSprop.

Графіки val_loss і val_accuracy на тестовому наборі даних для оптимізаторів (окрім Adadelta) не гладкі. На відміну від інших алгоритмів, Adadelta є алгоритмом оптимізації з адаптивною швидкістю навчання. Тому параметри оновлюються з меншим кроком, а під час навчання не виникає проблем з перенавчанням. Як наслідок, криві точності та похибок на тренувальних і тестових наборах даних майже збігаються та є плавними. Оскільки Adadelta є адаптивним алгоритмом, на початку необхідно використовувати більш високу швидкість навчання. Це значно скоротить час навчання та забезпечить збіжність моделі.

1.5 Порівняльний аналіз генеративно-змагальних мереж для синтезу біомедичних зображень

Для проведення експериментів використовувався навчальний набір цитологічних зображень, який опубліковано на платформі Zenodo, яка дозволяє зберігати дослідницькі статті, набори даних, програмне забезпечення на інші цифрові артефакти [77]. В навчальному наборі містяться лише зображення та діагноз, без жодної приватної інформації про пацієнтів. Тому ідентифікувати особу на основі цієї інформації неможливо. Таким чином, забезпечується анонімізація даних пацієнтів, які можуть бути використані в дослідницьких цілях.

Зображення із навчального набору даних конвертовано у роздільну здатність 64×64 пікселі (первинна роздільна здатність становить 3264×2448). Початкова кількість зображень становить близько 100, що є дуже малим значенням. Тому набір даних розширено до 800 зображень шляхом застосування афінних перетворень. Застосувавши цю техніку, набір даних було збалансовано – він містить однакову кількість (200 зображень) на клас. Для розширення навчального набору даних використано власну бібліотеку Rudi [78] з параметрами за замовчуванням. До зображень застосовано випадкове обертання, відображення, масштабування, нахил і спотворення. Всі операції застосовано з ймовірністю 50%.

Для експериментів обрано наступні архітектури: DCGAN, WGAN, WGAN-GP, BGAN, BEGAN. Причина по якій в порівняння не включено мережі BigGAN і StyleGAN полягає в тому, що ці моделі вимагають досить великих обчислювальних ресурсів, процес їх навчання може тривати до кількох днів і навіть тижнів і, що більш важливо, вони потребують великого набору навчальних даних, якого ми поки не маємо. Отже, вибрані архітектури та параметри навчання наведено в таблиці 1.4. У таблиці вказано назву архітектури, використаний оптимізатор (варіант методу градієнтного спуску), кількість ітерацій, розмір пакету та час навчання. Розмір пакету

використовується в міні-пакетному алгоритмі градієнтного спуску та визначає кількість навчальних прикладів в одному пакеті, який вибирається випадковим чином на кожній ітерації. Розрахунок помилки мережі базується на цьому пакеті. Порівняння синтезованих зображень показано на рисунках 1.16–1.20.

Таблиця 1.4. Параметри навчання архітектур GAN

№	Архітектура	Оптимізатор	К-сть ітерацій	Роз. пакету	Ч. навчання
1	DCGAN	Adam	15 000	64	41 хв
2	WGAN	Adam	15 000	64	1 г 52 хв
3	WGAN-GP	Adam	15 000	64	5 г 43 хв
4	BGAN	Adam	15 000	64	3 г 44 хв
5	BEGAN	Adam	15 000	64	6 г 25 хв

Кожна з ГЗМ побудована однаковим способом. У дискримінаторі використано вхідний шар згортки з функцією активації LeakyReLU, три згорткові блоки (шар згортки, пакетна нормалізація та активація LeakyReLU) і шар згортки як вихід. У генераторі використано чотири транспонованих згорткових блоки (транспонований шар згортки, пакетна нормалізація та активація ReLU) і транспоновану згортку як вихід. Розмір ядра, крок і відступ встановлені у значення 4, 2 і 1 відповідно в усіх згорткових блоках. Гіперпараметри згортки, такі як кількість вхідних та вихідних ознак в дискримінаторі, подвоюються в кожному наступному блоці. Початкове значення дорівнює 64. Для генератора початкове значення дорівнює 1024. У кожному наступному блоці це значення зменшується вдвічі.

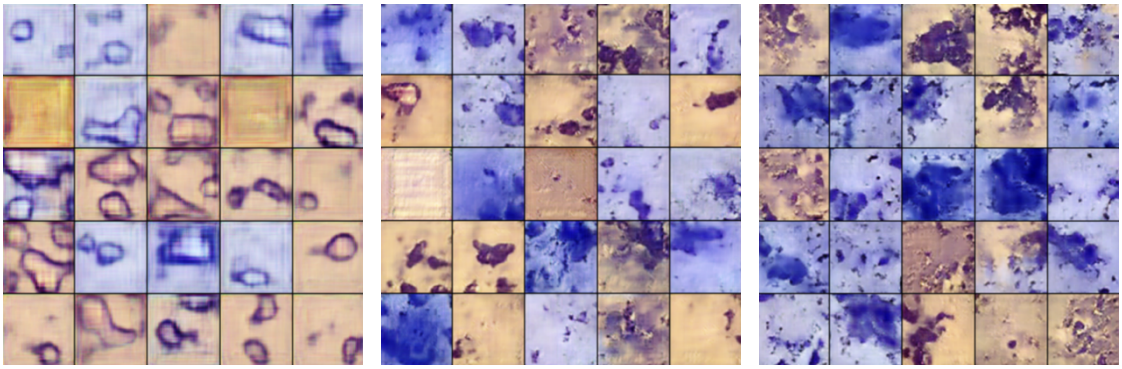


Рисунок 1.16. DCGAN результати після 1000, 7000, і 15000 ітерацій

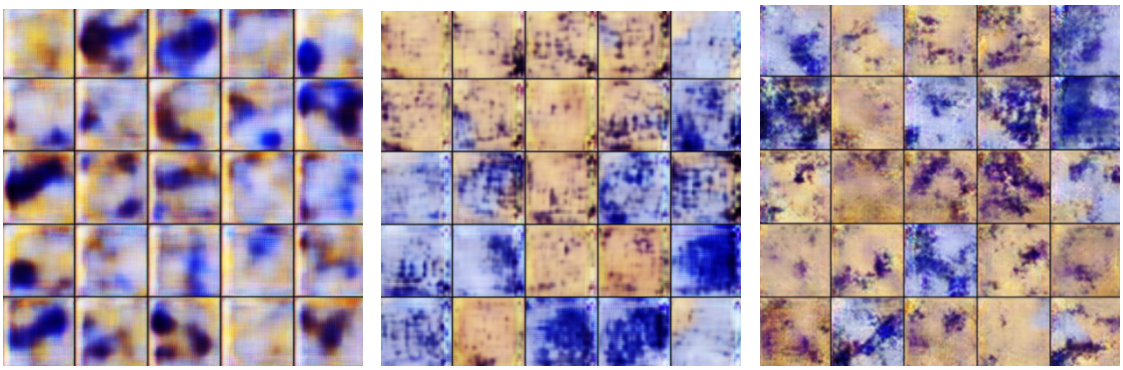


Рисунок 1.17. WGAN результати після 1000, 7000, і 15000 ітерацій

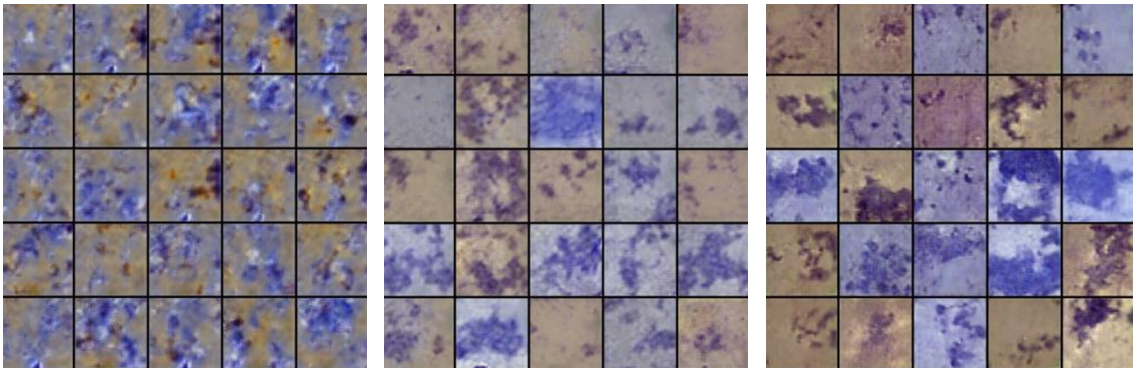


Рисунок 1.18. WGAN-GP результати після 1000, 7000, and 15000 ітерацій

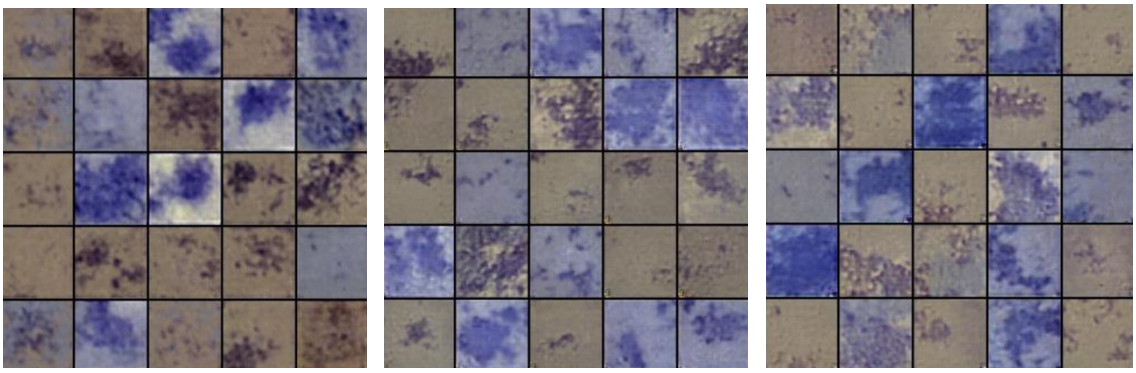


Рисунок 1.19. BGAN результати після 1000, 7000, і 15000 ітерацій

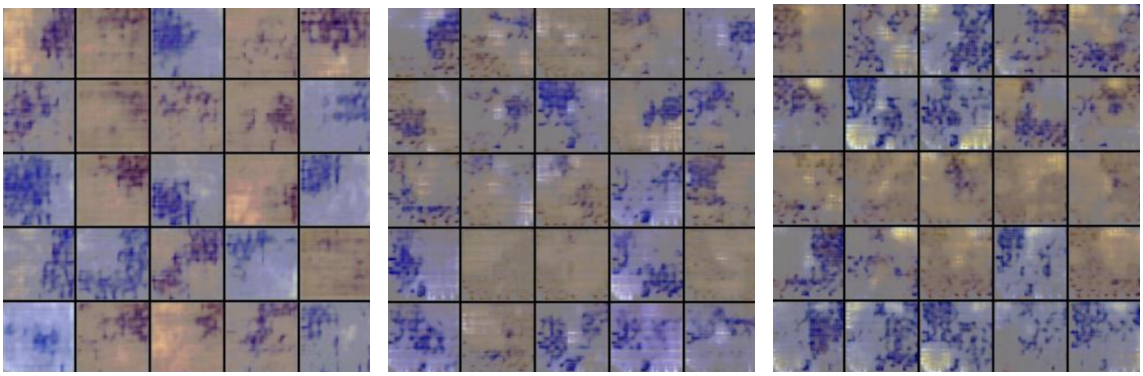


Рисунок 1.20. BEGAN результати після 1000, 7000, і 15000 ітерацій

Кожна з обраних мереж навчалася протягом 15 000 ітерацій. Параметри мережі, такі як швидкість навчання, оптимізатор і розмір пакету (також кінцева функція активації), були встановлені відповідно до значень, наданих авторами в статтях [28], [64], [65], [66], [67]. Експерименти проводилися з використанням графічного процесора Nvidia Tesla K80 за допомогою Google Colaboratory. Для написання коду використовувалася мова програмування Python та бібліотека PyTorch.

Для аналізу якості синтезованих зображень було розраховано метрику FID, що наведена у таблиці 1.5. Порівняльна діаграма наведена на рисунку 1.21.

Таблиця 1.5. Метрика FID кожної з архітектур

Архітектура	К-сть ітерацій (у тисячах)										
	1	2	3	4	5	10	11	12	13	14	15
	FID										
DCGAN	24,58	24,16	22,33	21,12	20,66	16,34	15,97	15,56	13,47	12,86	12,67
WGAN	26,89	25,95	23,14	22,36	20,18	16,02	15,11	14,33	13,01	12,96	12,72
WGAN-GP	34,17	33,86	32,12	31,02	29,35	25,01	24,39	23,11	21,58	20,03	19,09
BGAN	16,89	16,56	16,02	15,76	15,21	13,41	13,05	12,78	12,34	11,24	10,03
BEGAN	21,90	21,45	21,12	20,46	20,04	18,23	17,88	17,03	16,64	15,77	15,32

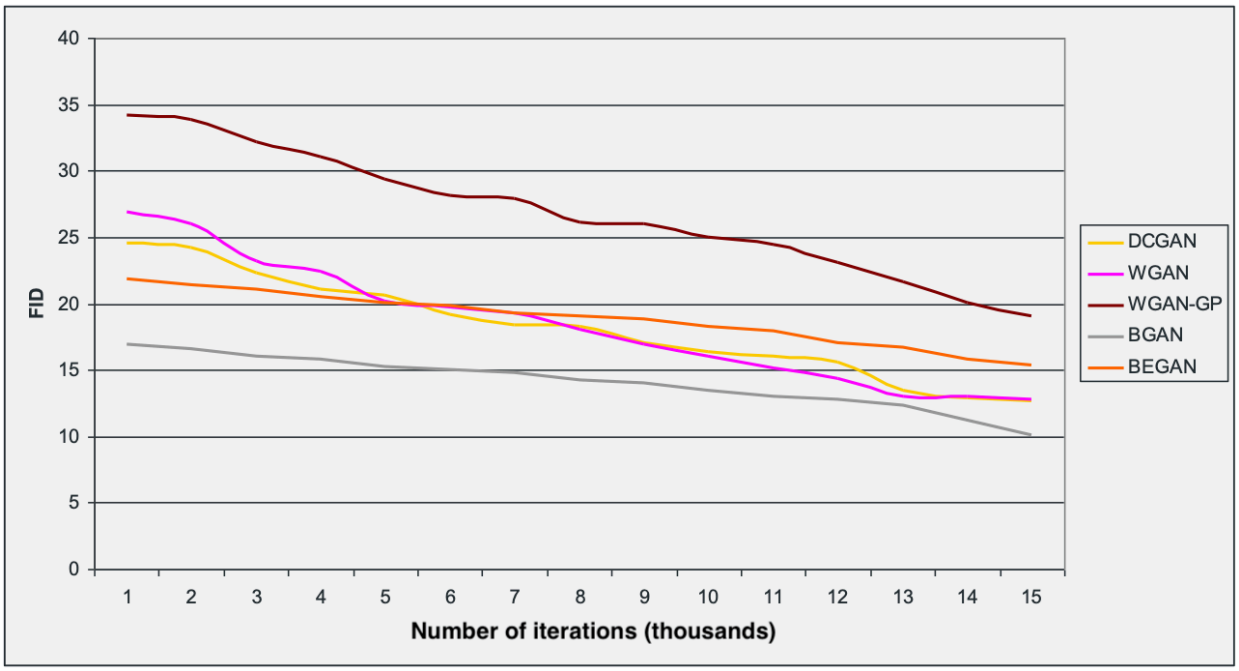


Рисунок 1.21. Значення FID протягом навчання

Як видно із рисунка 1.21, для архітектури BGAN крива FID плавно змінювалася від 17 до 10. Для архітектури DCGAN ця крива більш різка, ніж для BGAN. Метрика змінювалася від 24 до 17. Крива FID для архітектури WGAN падає різкіше, а в кінці навчання стає повільнішою. Значення метрики змінюється від 27 до 17. Для синтезу цитологічних зображень доцільніше використовувати архітектуру BGAN з меншою кількістю ітерацій (приблизно 7000) та часом навчання приблизно 1,5 години.

1.6 Метрики для оцінки подібності зображень

Метрикою називають певну функцію відстані між будь-якою парою елементів деякої множини. Метрична функція повинна задовольняти 3 аксіоми. Метрика повинна бути тотожною, симетричною та задовольняти нерівність трикутника. Метрики діляться на якісні та кількісні. Найпоширенішими метриками, що застосовуються в дослідженнях є саме кількісні метрики [52], [79].

Якісними метриками називають метрики, які не є числовими та часто передбачають суб’єктивну оцінку людини чи оцінку шляхом порівняння.

Найбільш популярними методами є Nearest Neighbor (схожі зображення групуються в кластери) та Rapid Scene Categorization [80]. Останній полягає в тому, що експерти повинні зробити вибір між реальним та синтетичними зображенням за короткий проміжок часу. Основним недоліком підходу, що базується на оцінках експертів є те, що експерти з часом можуть покращувати свої навички. Наприклад, експерти можуть отримувати зворотну оцінку від інших експертів, а також отримувати підказки як краще виявити синтетичне зображення.

Кількісні метрики базуються на обчисленні конкретних числових балів, які використовуються для підсумовування якості синтезованих зображень. До таких метрик відносять MSE, RMSE, SSIM, PSNR, IS, FID.

Мета оцінки подібності зображень полягає в тому, щоб виміряти відстань між синтетичними та реальними зображеннями. Більшість існуючих методів використовують початкову модель Inception для відображення зображень у просторі меншої розмірності. Найпоширенішою метрикою на даний час є IS, яка вимірює відстань за допомогою дивергенції Кульбака-Лейблера [81]. Однак дана метрика базується на імовірності приналежності зображення до одного із класів і не може відобразити перенавчання мережі. В якості кращої альтернативи запропоновано FID [82]. Ця метрика безпосередньо вимірює відстань Фреше на просторі ознак за допомогою апроксимації нормального розподілу.

Оскільки дані метрики базуються на попередньо підготовленій моделі Inception (яка навчалася на датасеті ImageNet), то їх значення можуть погіршуватися при застосуванні до інших датасетів. Серед всіх метрик найбільш популярними та релевантними метриками для оцінки якості синтезованих за допомогою ГЗМ зображень є IS та FID. Дані метрики досить добре себе зарекомендували у багатьох дослідженнях та показали хорошу кореляцію із оцінками експертів.

IS. Дана метрика базується на моделі нейронної мережі для класифікації зображень Google Inception V3. Ця модель призначена для класифікації кольорових зображень. В якості навчального набору даних використано набір даних ImageNet, який включає близько 1,2 мільйона RGB зображень, що поділені на 1000 класів.

Дана метрика показала хорошу кореляцію із оцінками, що зроблені експертами на датасеті CIFAR-10.

$$IS(G) \approx exp(E_{x \sim p_g}[D_{KL}(p(y|x) \,||\, p(y))]) \tag{1.11}$$

де E – математичне сподівання; $x \sim p_g$ показує, що x є зображенням, що синтезоване із розподілу p_g(розподіл генератора); D_{KL} є відстанню Кульбака-Лейблера між розподілом умовної імовірності $p(y|x)$ та маргінальним розподілом $p(y)$. Передбачається, що умовний розподіл даних, який містить значущі об'єкти, повинен мати низьку ентропію, а маргінальний розподіл (синтезовані зображення є різноманітними) — високу.

IS працює так. Для прикладу візьмемо 5000 тисяч синтезованих зображень. Для того, щоб отримати умовний розподіл класів, потрібно класифікувати дані зображення мережею Inception, яка поверне вектор імовірностей $p(y|x)$. Для отримання маргінального розподілу потрібно просумувати умовний розподіл для кожного зображення $p(y) = \frac{1}{5000}\sum_{i=1}^{5000} p(y|x_i)$. Далі потрібно обчислити відстань Кульбака-Лейблера між умовним розподілом кожного синтезованого зображення та загальним маргінальним розподілом. Середнє значення даних відстаней і буде значенням метрики IS [83].

Отже, IS вимірює середню відстань Кульбака-Лейблера між умовним розподілом $p(y|x)$ та маргінальним розподілом класів $p(y)$. Тобто дана метрика взагалі не розглядає розподіл оригінальної вибірки, а тому не може оцінити наскільки добре синтезовані генератором зображення подібні до оригінальної

вибірки. Дана метрика оцінює лише їх різноманітність. Мінусами даної метрики є чутливість до роздільної здатності самих зображень та до змін в самій мережі, яка використовується для класифікації.

Мінімальним значенням даної метрики є 1, а максимальним — кількість класів, яку може класифікувати мережа Inception. В даному випадку — 1000.

Для того, щоб отримати високе значення IS, потрібно щоб синтетичні зображення містили чіткі об'єкти та щоб генератор синтезував різноманітні зображення із всіх класів [82]. Відповідно, якщо хоча б одна із цих умов незадовільна — оцінка буде низькою.

FID порівнює розподіли оригінальних та синтетичних даних. Для того, щоб обчислити FID між реальними та синтезованими зображеннями, дані перетворюються в простір ознак з використанням конкретного шару моделі Inception, а саме — *pool3* layer. Простір ознак використовується для того, щоб представити зображення у просторі меншої розмірності, де схожі зображення представлені у відносно тих самих регіонах. На виході ми отримуємо карти активацій (карти ознак). Далі ці карти ознак апроксимуються із використанням двох розподілів Гауса. Тоді відстань між ними обчислюється наступним чином:

$$d^2((m_r C_r), (m_g C_g)) = \left\| m_r - m_g \right\|^2 + Tr(C_r + C_g - 2(C_r C_g)^{\frac{1}{2}}), \qquad (1.12)$$

де $(m_r C_r)$ та $(m_g C_g)$ — середнє та covariance реального та синтезованого розподілу даних відповідно, Tr — сума діагональних елементів матриці.

Чим нижче значення метрики, тим менша відстань між розподілами, відповідно розподіли більш подібні між собою [84]. FID метрика досить чутлива до спотворень на зображеннях (зсув, шум і т.д). Чим більше спотворень, тим більшим буде значення метрики.

Низьке значення FID свідчить про те, що розподіли реальних та синтетичних зображень подібні між собою. Проте на практиці, якщо модель має низьке значення FID, то це свідчить про те, що зображення мають високу

якість або різноманітність, або й те й інше. Така поведінка може значно ускладнити діагностику моделі.

Автори також показують, що дана метрика більше збігається із людськими оцінками та більш стійка до шуму, ніж IS [85], [86].

Дані метрики є досить популярними в області синтезу зображень за допомогою GAN мереж. Але вони мають свої недоліки.

Метрика IS має такі обмеження:

1) Значення метрики сильно залежить від того, що може класифікувати модель Inception.
2) Синтез зображень іншого набору класів, яких немає в оригінальному датасеті ImageNet, може спричинити низьке значення IS.
3) Якщо класифікатор не може визначити ознаки, які відносяться до навчального датасету, то зображення низької якості можуть отримувати високі оцінки. Мережа Inception натренована на датасеті ImageNet. Якщо обчислювати IS для іншого датасету, то класифікатор може досить погано визначити деякі ознаки, відповідно зображення низької якості будуть отримувати високі оцінки.

FID також базується на моделі Google Inception. Але на відміну від IS, дана метрика може визначити залежності між класами. Тобто якщо модель генерує тільки одне зображення на кожен клас, то IS може бути досить високим, проте FID буде низьким. Також метрика FID погіршується при додаванні різних артефактів до зображення.

Метрика IS дійсно показує кореляцію з якістю та різноманітністю створених зображень, що пояснює широке використання на практиці. Однак дана метрика оцінює лише розподіл синтезованих зображень, проте не бере до уваги те, наскільки синтезовані та оригінальні зображення є подібними. Як наслідок, це може спонукати моделі просто вивчати чіткі та різноманітні зображення (або навіть деякі шуми) замість розподілу оригінальних даних. Ця метрика обмежена тільки вимірюванням того, наскільки синтезовані

зображення є різноманітними, а FID вимірює відстань між розподілом синтезованих та реальних даних.

Оскільки обидві метрики базуються на класифікаторі Inception для отримання умовних імовірностей (метрика IS) та карт активацій (метрика FID), то це може суттєво впливати на результати при обчисленні даних метрик для даних, які не включені в набір даних ImageNet на якому і була навчена мережа Inception.

З метою перевірки різниці між значеннями метрик IS та FID, обчислених за допомогою моделі Inception, від значень метрик обчислених за допомогою іншої моделі розроблено архітектуру класифікатора для біомедичних зображень, що забезпечує отримання більш релевантних умовних імовірностей для метрики IS та карт активацій для метрики FID.

Обидві мережі приймають на вхід кольорові зображення розміром 64×64 пікселі відповідно до роздільної здатності зображень в навчальному наборі даних та називаються BioCNN-1 (рисунок 1.22) та BioCNN-2 (рисунок 1.23).

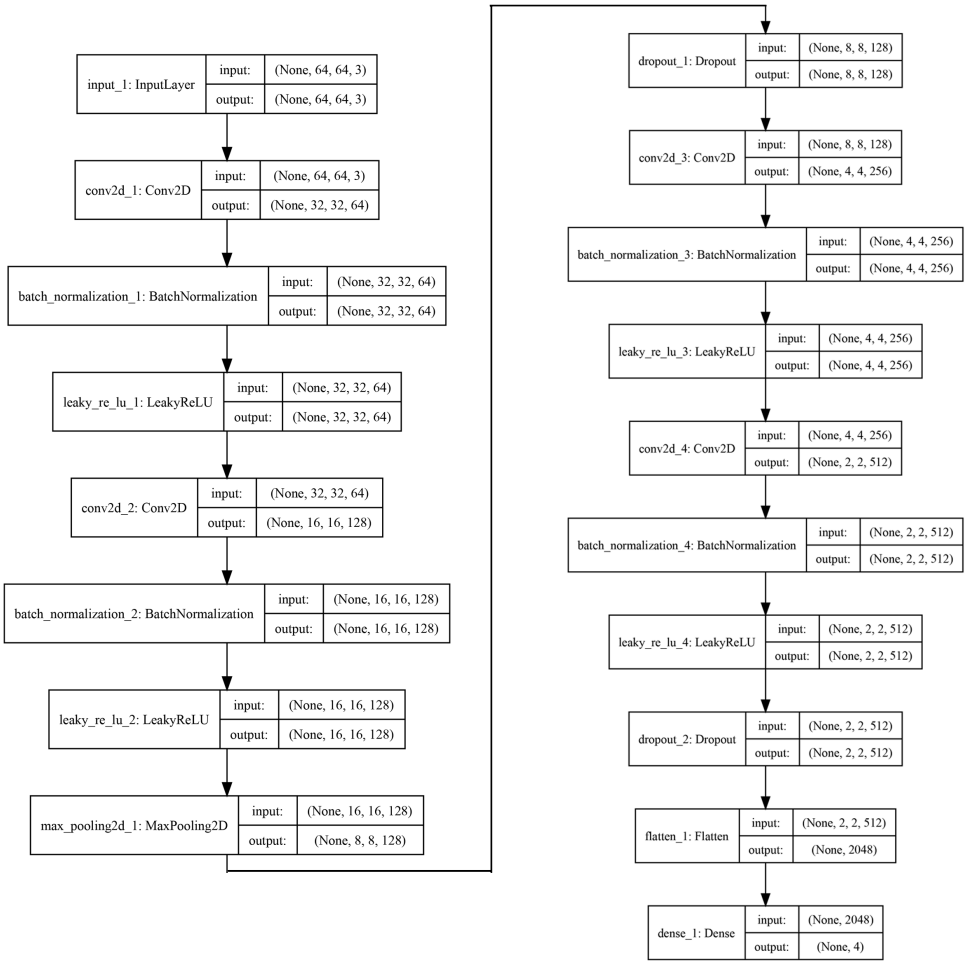


Рисунок 1.22. Архітектура BioCNN-1

Дані мережі є згортковими нейронними мережами. Архітектура BioCNN-1 складається із послідовності шарів згортки, батч нормалізації і активації LeakyReLU. Один набір цих шарів можна назвати блоком згортки. BioCNN-1 складається із чотирьох таких блоків.

Архітектура BioCNN-2 побудована із використанням блоків VGG та ResNet, які чергуються між собою. Дані блоки є окремими елементами архітектури популярних згорткових нейронних мереж VGG та ResNet відповідно [71], [87]

Загалом VGG архітектура складається із послідовності згорткових шарів, що використовують малий розмір вікна згортки (3×3). В кінці такого блоку розміщено шар максимального пулінгу.

ResNet блок складається із двох згорткових шарів з однаковою кількістю фільтрів, де вихід другого шару додано до входу першого.

Також дані архітектури можна покращити шляхом оптимізації гіперпараметрів, що описано у роботі [88].

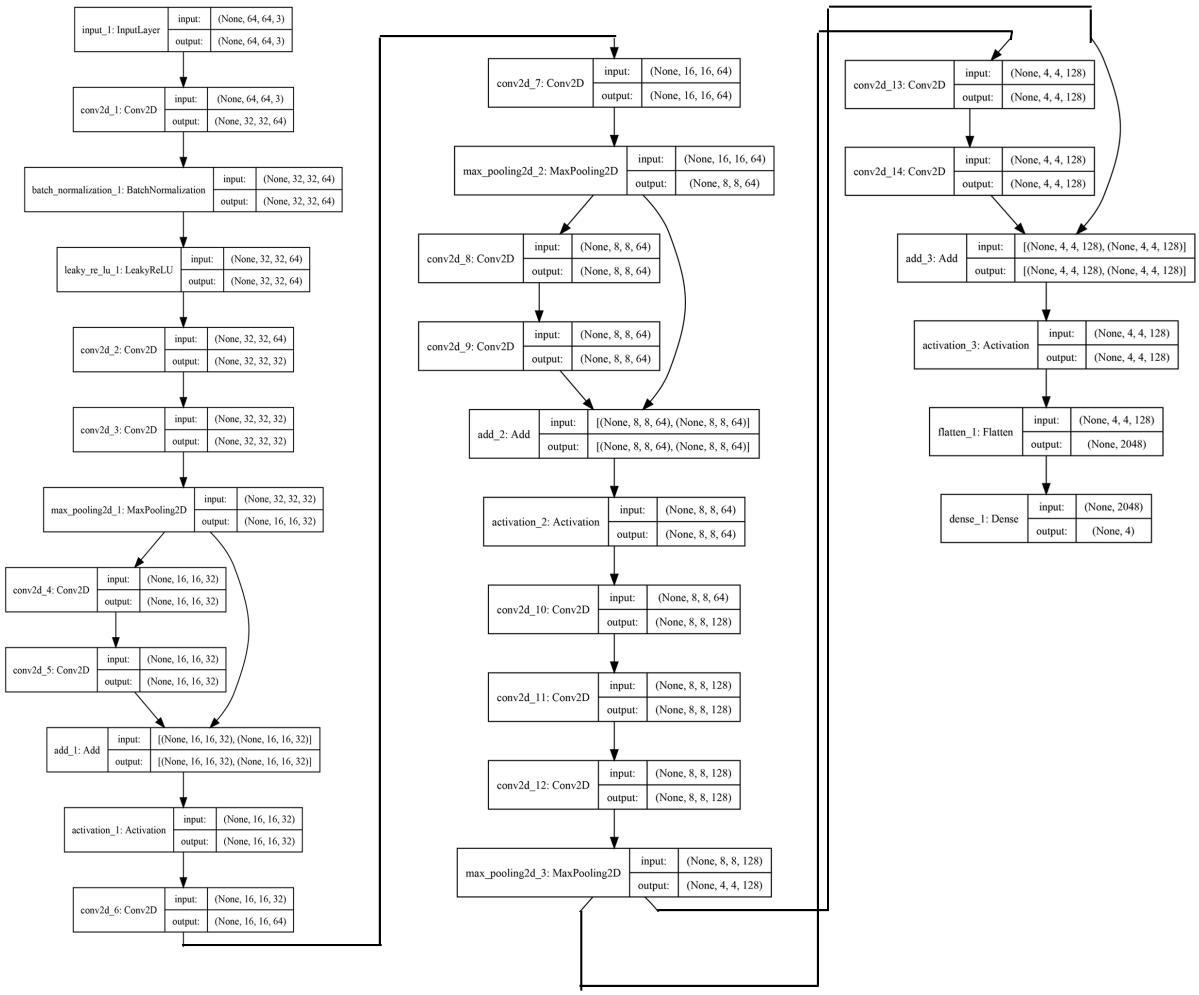


Рисунок 1.23. Архітектура BioCNN-2

Для обчислення метрик на основі власного класифікатора застосовано запропоновані архітектури ЗНМ BioCNN-1 та BioCNN-2. Для побудови моделей, їх навчання та обчислення метрик IS та FID розроблено програмний модуль на мові програмування Python та з використанням фреймворка для машинного навчання Keras. Експерименти проведено на ноутбуці із CPU Intel

Core i7 2.5GHz, RAM 16GB. Гіперпараметри навчання даних мереж наведено у таблиці 1.6.

Таблиця 1.6. Параметри навчання мереж

Назва	Функція втрат	Оптимізатор	Норма навчання	Розмір пакету	К-сть епох
BioCNN-1	перехресна ентропія	Adam	0,003	128	40
BioCNN-2	перехресна ентропія	Adam	0,003	64	100

В якості навчального набору даних використано вибірку із 800 реальних зображень та 800 синтетичних, створених архітектурою BGAN у попередньому розділі. Даний набір даних поділено у співвідношенні 80-10-10 в якості навчального, тестового та валідаційного. Мережа BioCNN-1 досягнула точності класифікації 95% а BioCNN-2 – 96,8%. Час навчання першої мережі склав приблизно 15 хвилин, а другої – 35 хвилин. Друга мережа потребує більше часу для навчання, бо її архітектура є глибшою за першу. ROC криві для обох мереж наведено на рисунках 1.24 і 1.25 відповідно.

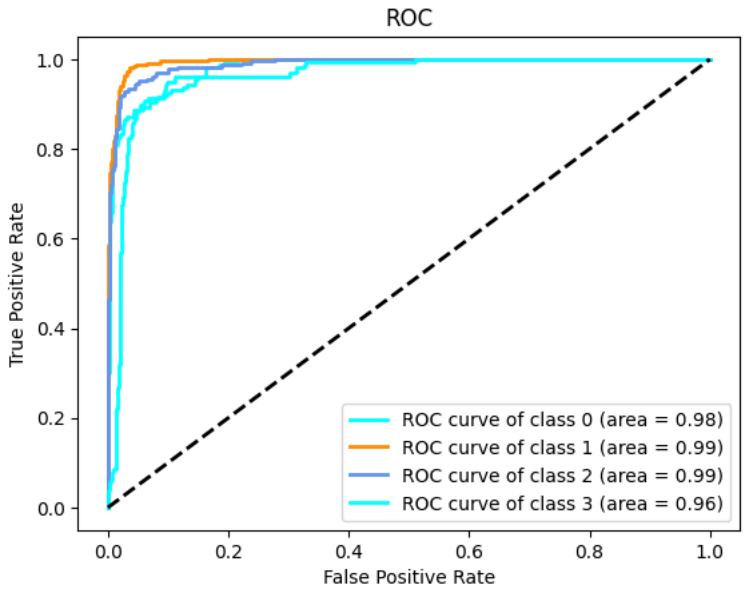


Рисунок 1.24. ROC крива для BioCNN-1

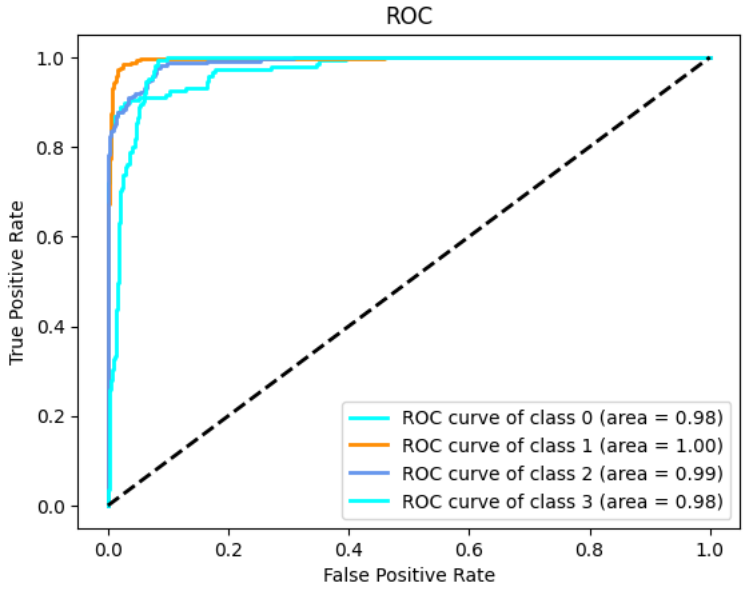


Рисунок 1.25. ROC крива для BioCNN-2

Після навчання обох мереж обчислено значення метрик IS та FID для порівняння валідаційного датасету із синтезованими зображеннями. Для отримання карт активацій, що використовуються в метриці FID, взято четвертий з кінця шар (leaky_re_lu_4) для моделі BioCNN-1 та третій з кінця шар (activation_3) для моделі BioCNN-2. Узагальнені результати наведено у таблиці 1.7.

Таблиця 1.7. Результати експерименту

IS ↑	FID ↓	Модель	Час обчислення
3,22	10,03	Google Inception V3	~ 2 хв.
3,52	7,41	BioCNN-1	~ 8 сек.
3,71	2,34	BioCNN-2	~ 15 сек.

В результаті експериментів показано, що значення метрик покращилося при застосуванні розроблених моделей. Спостерігається незначне покращення метрики IS. Це свідчить про те, що метрика IS практично не залежить від використаної моделі. Причиною цього є те, що дана метрика обчислюється на основі імовірностей приналежності зображення до одного із класів. Теоретичним поясненням є те, що схожі зображення будуть віднесені до одного

класу незалежно від використаної моделі. Проте застосування власних моделей таки покращило значення метрики IS, бо користувацька модель краще класифікує цитологічні зображення, ніж модель Inception.

При використанні моделі BioCNN-1 для обчислення метрики FID порівняно із застосуванням моделі Inception значення FID зменшилося із 10,03 до 7,41. Проте при застосуванні моделі BioCNN-2 значення метрики зменшилося до 2,34. Для обчислення даної метрики використовуються карти ознак, які отримуються із конкретного шару базової моделі. Покращення значень метрики при застосуванні розроблених моделей свідчить про те, що розроблені моделі надають більш релевантні карти ознак для цитологічних зображень, оскільки вони навчалися на зображеннях із даного класу.

Значну різницю між значеннями метрики FID при застосуванні моделей BioCNN-1 та BioCNN-2 можна пояснити спираючись на архітектурні особливості самих мереж. Незважаючи на те, що обидві мережі під час навчання досягнули приблизно однакової точності класифікації на тестовому датасеті, друга мережа є значно глибшою за першу. Під час експериментів також помічено тенденцію до значного збільшення значення FID із наближенням шару, який використовується як екстрактор ознак, до входу мережі. Дана закономірність є менш вираженою для мережі BioCNN-2. Оскільки дані мережі навчалися на однаковому наборі даних, то із значного розкиду значення FID при використанні розроблених мереж слідує висновок, що глибша мережа (BioCNN-2) може набагато краще представити вхідне зображення у просторі низької розмірності, що відповідно продукує більш релевантні та «інформативніші» карти ознак. Відповідно метрика FID досить сильно залежить від мережі, яка використовується в якості екстрактора ознак, порівняно із метрикою IS.

Практичне проведення експериментів показало, що використання розроблених моделей значно зменшує час обчислення даних метрик порівняно із застосуванням мережі Inception. Час обчислення зменшився із 2 хвилин до 15

секунд. Значне прискорення у часі обчислення та покращення значень самих метрик дає змогу розвинути дане дослідження у напрямку застосування метрики FID, як додаткового параметра у функції втрат ГЗМ, що теоретично дало б змогу покращити якість синтезованих зображень.

1.7 Висновки до розділу 1

1. Проведено аналіз біомедичних зображень: цитологічних, гістологічних та імуногістохімічних. Дані зображення є кольоровими, контрастними та містять структури клітин та тканин. Цитологічні зображення вимагають високої точності у розпізнаванні та класифікації клітинних структур, що може бути складним через їхню візуальну схожість між різними типами клітин. Гістологічні та імуногістохімічні зображення, які включають більші та складніші структури тканин, вимагають розуміння глибших шарів та взаємодій між різними клітинними компонентами, що також ускладнює процес їх аналізу.

2. Проаналізовано архітектури класичних ГЗМ мереж та встановлено, що практично кожна із них містить однакові комбінації шарів в генераторі та дискримінаторі. Деякі архітектури є ідентичними та відрізняються лише кінцевою функцією втрат. Досліджено, що практично будь-яка архітектура ГЗМ мережі має два недоліки: затухання градієнтів та колапс, які можна частково вирішити такими техніками, як зміна функції активації та штрафування градієнту. Проаналізовано алгоритми навчання нейронних мереж та здійснено порівняння алгоритмів на основі градієнтного спуску. Алгоритми порівняно за такими критеріями: час навчання ЗНМ, похибки на навчальному та тестовому наборі даних, точність класифікації. Досліджено, що не всі алгоритми показують однакові результати при однаковій кількості ітерацій навчання. Оскільки експерименти проведено з використанням тільки однієї архітектури ЗНМ, зроблено висновок, що застосування таких самих алгоритмів для більш глибоких і складних мереж може призвести до інших результатів, які

вимагатимуть додаткових досліджень та налаштувань для оптимізації процесів навчання та аналізу відповідних параметрів цих мереж.

3. Проведено порівняння класичних архітектур ГЗМ (DCGAN, WGAN, WGAN-GP, BGAN, BEGAN) для синтезу цитологічних зображень. Хоча найкращим результатом за метрикою FID є архітектура BGAN, встановлено, що класичні архітектури синтезують зображення не найкращої якості (зображення є розмитими, неможливо розрізнити окремі клітини, як в наборі реальних даних). Відповідно актуальним завданням є розробка власних архітектур ГЗМ для синтезу біомедичних зображень.

4. Проведено аналіз метрик (IS та FID) для оцінки подібності зображень та розроблено базові моделі, що використовуються при обчисленні цих метрик. Здійснено порівняння значень розглянутих метрик при застосуванні моделі Inception та розроблених моделей. Встановлено, що значення метрики IS практично не залежить від використаної моделі, на відміну від метрики FID. Експерименти показали, що використання розроблених моделей в якості базової моделі для метрик IS та FID значно скоротило час їх обчислення (який зменшився із 2 хвилин до 15 секунд). Це дасть змогу в майбутньому розвинути дане дослідження в напрямку використання метрики FID як параметра у функції втрат при навчанні ГЗМ, що теоретично повинно покращити якість синтетичних зображень.

5. Сформульовано задачі дисертаційного дослідження: провести аналіз алгоритмів, методів та програмних засобів для синтезу зображень; провести аналіз і порівняння класичних архітектур нейронних мереж для синтезу біомедичних зображень; розробити метод автоматичного пошуку архітектур нейронних мереж для класифікації біомедичних зображень; розробити метод автоматичного пошуку архітектур нейронних мереж для синтезу біомедичних зображень; розробити метод синтезу та класифікації біомедичних зображень для отримання синтетичних зображень; спроектувати та реалізувати програмну

систему синтезу та класифікації біомедичних зображень; виконати комп’ютерні експерименти із синтезу та класифікації біомедичних зображень.

РОЗДІЛ 2. МЕТОД АВТОМАТИЧНОГО СИНТЕЗУ АРХІТЕКТУР ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

У даному розділі розроблено метод синтезу архітектур ЗНМ. Проведено порівняння архітектур, що синтезовані розробленим методом, з іншими відомими дослідженнями для класифікації цитологічних зображень.

Результати розділу опубліковано у працях автора [8], [9].

2.1 Модель опису архітектур нейронних мереж

Для опису архітектур спочатку введемо такі умовні позначення:

I_{inp} – множина вхідних зображень;

I_l – множина навчальних зображень;

I_t – множина тестових зображень;

L – множина шарів ЗНМ;

A_{CNN} – архітектура ЗНМ;

A_{CL} – архітектура комірки ЗНМ;

O_{CL} – множина операцій в комірці;

O_{CLB} – множина операцій між комірками;

P_{CNN} – множина параметрів ЗНМ;

P_{CL} – множина параметрів комірки ЗНМ;

$G_{\max}$ – функція пулінгу з визначенням максимального елементу у вікні сканування;

G_{avg} – функція пулінгу з визначенням середнього елементу у вікні сканування;

G_{ad_avg} – функція пулінгу з адаптивним визначенням середнього елементу у вікні сканування;

$C_{o\times o}$ – функція згортки з вікном сканування розміром $o \times o$;

j – індекс класу;

i – індекс шару;

k– індекс комірки;

l – індекс операції між комірками;

m – кількість класів;

r – індекс вузла;

n – кількість шарів ЗНМ;

d – кількість вузлів в комірці;

q – кількість комірок в архітектурі ЗНМ;

t – кількість операцій між комірками;

o – розмір вікна сканування;

AC – міра точності класифікації ЗНМ на етапі навчання;

AC^* – міра точності класифікації ЗНМ на етапі тестування;

F – функція класів.

Нехай задана множина зображень I_{inp}. Розділимо дану множину на дві підмножини: навчальну I_l та тестову I_t, тобто $I_{inp} = I_l \cup I_t$. Крім того задана множина операцій ЗНМ:

$$O_{CNN} = \{\langle C\rangle; \langle P\rangle; \langle K\rangle; S; V; \langle B\rangle\}, \tag{2.1}$$

де $\langle C\rangle$ – множина функцій згортки; $\langle G\rangle$ – множина функцій пулінгу; $\langle K\rangle$ – множина функцій активації; S – функція *Skip connection*; V – операція batch normalization; $\langle B\rangle$ – множина функцій повнозв'язної мережі.

Кожен шар ЗНМ є окремою операцією із параметрами. Тому архітектуру CNN можна представити через множину шарів:

$$A_{CNN} = \left\{L_i, i = \overline{1,n}\right\}. \tag{2.2}$$

Оскільки ряд операцій в архітектурі ЗНМ повторюється, то виділимо їх в окрему множину. Цю множину повторюваних операцій назвемо коміркою. Тоді

архітектуру ЗНМ можна також представити як сукупність архітектури комірок і операцій між ними:

$$A_{CNN} = \left\{ \left\langle A_{CL} \right\rangle ; \left\langle O_{CLB} \right\rangle \right\}. \tag{2.3}$$

Архітектура комірки – це сукупність вузлів і операцій між ними. Архітектуру комірки представимо так:

$$A_{CL} = \left\{ \left\langle O_{CL_1} \right\rangle ; \left\langle O_{CL_2} \right\rangle ; ... ; \left\langle O_{CL_r} \right\rangle ; ... ; \left\langle O_{CL_d} \right\rangle \right\}, \tag{2.4}$$

де $\left\langle O_{CL_r} \right\rangle$ – кортеж операцій для *r*-го вузла, $r = \overline{1,d}$, $O_{CLB} \in O_{CNN}$.

Деталізуємо множину операцій згортки:

$$C = \left\{ \left\langle c_1 \right\rangle \left\langle v_{c_1} \right\rangle , ... , \left\langle c_i \right\rangle \left\langle v_{c_i} \right\rangle , ... , \left\langle c_n \right\rangle \left\langle v_{c_n} \right\rangle \right\}, \tag{2.5}$$

де c_i – функція активації в *i*-му шарі, v_{c_i} – параметри функції активації в *i*-му шарі.

Аналогічно визначаються інші операції перетворення.

2.2 Постановка задачі оптимізації архітектури нейронної мережі

Розв’язок задачі оптимізації архітектури нейронної мережі має такі етапи:

1) побудова математичної моделі архітектури нейронної мережі;
2) визначення цільової функції і параметрів нейронної мережі, які необхідно оптимізувати;
3) мінімізація функції міри точності класифікації нейронної мережі;
4) отримання архітектури нейронної мережі з максимальною мірою точності класифікації.

Цільовою функцією оптимізації архітектури нейронної мережі є міра точності класифікації. Для оцінки міри точності класифікації використаємо міру Accuracy:

$$AC = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.6}$$

де TP – true positive, TN – true negative, FP – false positive, FN – false negative.

Функцію міри точності класифікації представимо у такому вигляді:

$$AC = f(A_{CL}, O_{CL}, O_{CLB}). \tag{2.7}$$

Дворівнева оптимізація буде мати такий вигляд:

1) $P_{CL} = \arg\max_{d, O_{CL}} AC(A_{CL}, O_{CL}, I_l)$

2) $P_{CNN} = \arg\max_{n, O_{CLB}} AC(n, O_{CLB}, I_l)$

Дійсна міра точності синтезованої мережі є функцією від таких параметрів:

$$AC^* = f(A_{CL}, O_{CL}, O_{CLB}, n, d, I_t) \tag{2.8}$$

Причому, $AC^* \geq AC$.

Результатом роботи знайденої архітектури ЗНМ над заданим вхідним зображенням класу є функція F_j, $F_j \in [0;1]$.

2.3 Метод автоматичного пошуку архітектури згорткової нейронної мережі

Для класифікації біомедичних зображень використовуються автоматизовані та автоматичні діагностичні системи. На основі результатів

класифікації ставиться попередній діагноз. Наприклад, можна віднести конкретне зображення до певного класу і таким чином визначити тип або підтип раку. Віднесення зображень до певного типу використовується в медицині для діагностики, прогнозування перебігу захворювання та планування лікування.

В автоматичних діагностичних системах для класифікації зображень часто використовують згорткові нейронні мережі, які показали вищу точність та ефективність, ніж традиційні класифікатори [71].

Для ефективного використання штучних нейронних мереж необхідно вирішити такі завдання: розробити архітектуру нейронної мережі, знайти набори даних для навчання нейронної мережі та навчити нейронну мережу.

Процес розробки архітектури нейромережі є складним, трудомістким і вимагає спеціальних знань [89]. Підхід до синтезу архітектур нейромереж, що називається NAS, забезпечує автоматичний пошук оптимальної архітектури нейронної мережі [90] і використовує алгоритми оптимізації, методи навчання з підкріпленням і генетичні алгоритми для пошуку оптимальної архітектури для конкретного завдання [91], [92]. NAS став популярною темою досліджень у галузі глибокого машинного навчання, оскільки ручне проектування архітектури нейронних мереж має низьку швидкість і продуктивність. Крім того, ручна розробка ефективних архітектур нейромереж є надзвичайно складним завданням через величезний простір пошуку та важливі архітектурні рішення, які впливають на точність та швидкість моделі.

Використання NAS потенційно може призвести до розробки більш оптимальних архітектур і допомогти дослідникам подолати власні обмеження в знаннях і мисленні. Основні публікації в області NAS загалом діляться на наступні три категорії: еволюційні алгоритми [93], градієнтні методи і методи на основі навчання з підкріпленням [94], [95], [96], [97].

RL-методи розглядають пошук архітектури як послідовну задачу прийняття рішень. Архітектура нейронної мережі оновлюється на кожному

кроці на основі функції винагороди, яка вимірює її продуктивність на валідаційному наборі даних [98]. Цей підхід виявився ефективним для задач класифікації зображень.

Однією з ключових перешкод для впровадження методів на основі навчання з підкріпленням, є висока обчислювальна вартість, особливо для організацій з обмеженими обчислювальними ресурсами. Більш широке впровадження обмежується складністю і глибокими знаннями топологій нейронних мереж і алгоритмів машинного навчання, які необхідні для реалізації алгоритмів RL для NAS. При застосуванні RL також існує ймовірність перенавчання мережі, особливо в ситуаціях з великими просторами пошуку і менш надійними метриками оцінювання.

Градієнтні методи використовують інформацію про градієнт параметрів мережі для керування пошуком. Таким чином, основним принципом є підхід до пошуку архітектури як до оптимізаційної задачі. Ці методи використовують диференційований простір пошуку, де кожна архітектура представлена у вигляді диференційованої параметризованої моделі, замість прямого перебору та оцінки всіх можливих архітектур [99].

Еволюційні алгоритми використовують принципи природної еволюції для пошуку оптимальних архітектур, які змінюються і вибираються на основі їхньої продуктивності. Основна ідея еволюційних методів полягає в тому, щоб закодувати архітектуру нейронної мережі в деяку послідовність символів або чисел, а потім "змінити" цю послідовність, використовуючи такі поняття, як селекція, кросовер і мутація.

В роботі [8] представлено детальне порівняння трьох популярних алгоритмів, що використовуються для оптимізації гіперпараметрів при пошуку архітектури нейромережі, зокрема для побудови згорткових нейронних мереж. Порівнюються пошук по ґратці, випадковий пошук та генетичний алгоритм. Експерименти проводяться з використанням набору даних CIFAR-10, а ефективність цих алгоритмів оцінюється на основі часу пошуку і точності

згенерованих моделей. В якості оптимізатора використано алгоритм Adamax. Експерименти демонструють різницю в продуктивності між трьома алгоритмами, показуючи, що хоча всі вони займають багато часу, генетичний алгоритм може бути швидшим, коли мережа має досить багато параметрів. Дослідження показує, що пошук по ґратці, як правило, занадто повільний, а випадковий пошук обмежений розподілом самого простору пошуку. Найбільш перспективним алгоритмом виявився генетичний алгоритм, особливо коли модель має багато шарів. У роботі зроблено висновок, що вибір алгоритму залежить від конкретних вимог моделі та простору пошуку. Для невеликих моделей і просторів пошуку може бути достатньо пошуку по ґратці або випадкового пошуку, але для більших і складніших сценаріїв рекомендується використовувати генетичний алгоритм завдяки його ефективності в обробці великої кількості параметрів.

Загалом існує два підходи до пошуку архітектури: мікросинтез та макросинтез [100]. Мікросинтез використовується для пошуку комірки. Коміркою можна назвати послідовність шарів, що повторюються. Наприклад, у сучасних архітектурах нейромереж підтримується повторюваність таких операцій, як активація, згортка та пакетна (батч) нормалізація [101], [102], [103]. Ці три операції виділено в окрему комірку. Макросинтез використовується для визначення кількості комірок та їх послідовності. Таким чином будується остаточна архітектура мережі [104]. Проблемою існуючих методів на основі мікро- та макросинтезу є те, що дані методи застосовано для пошуку архітектури рекурентних нейронних мереж і кількість підтримуваних операцій є досить обмеженою, що не гарантує отримання оптимальної архітектури [95].

В даному розділі представлено розроблений двоетапний метод автоматичного пошуку оптимальної архітектури згорткової нейронної мережі на основі генетичних алгоритмів з використанням стадій мікропошуку та макропошуку. Стадія мікропошуку передбачає пошук оптимальної архітектури

комірки. Це здійснюється за допомогою конкретного набору операцій для вибору кількості та типу дій у кожній комірці. На стадії макропошуку проводиться пошук оптимальної кількості послідовно з'єднаних комірок для побудови кінцевої архітектури мережі.

Множина можливих архітектур визначається простором пошуку, який визначено в термінах комірок. Кожна комірка складається з декількох вузлів. Простір пошуку також включає в себе кількість вузлів у комірці. Кожен вузол у комірці представляє операцію додавання, а зв'язки (операції) між вузлами вибираються з набору наперед визначених операцій. Визначаючи простір пошуку у вигляді комірок зменшується складність пошуку архітектури.

Перший етап пошуку оптимальної архітектури зосереджений на мікропошуку, який базується на пошуку оптимальної архітектури комірки. На цьому етапі використано генетичний алгоритм для оптимізації кількості вузлів у комірці та типів операцій між вузлами. Генетичний алгоритм оцінює кожну архітектуру-кандидата (комірку). Результатом цього етапу є оптимальна архітектура комірки. Критерієм зупинки є виконання певної кількості циклів генетичного алгоритму, яка задається користувачем.

Другий етап — макропошук, що орієнтований на пошук оптимальної кількості послідовно з'єднаних комірок для формування кінцевої архітектури CNN. Результатом цього етапу є повністю готова до використання архітектура нейронної мережі. Критерій зупинки такий самий, як і на першому етапі.

Для оцінки продуктивності кожного потенційного рішення використано набір валідаційних даних на кожному з двох етапів. Показником ефективності є точність класифікації.

На рисунку 2.1 показано узагальнену схему реалізації розробленого методу.

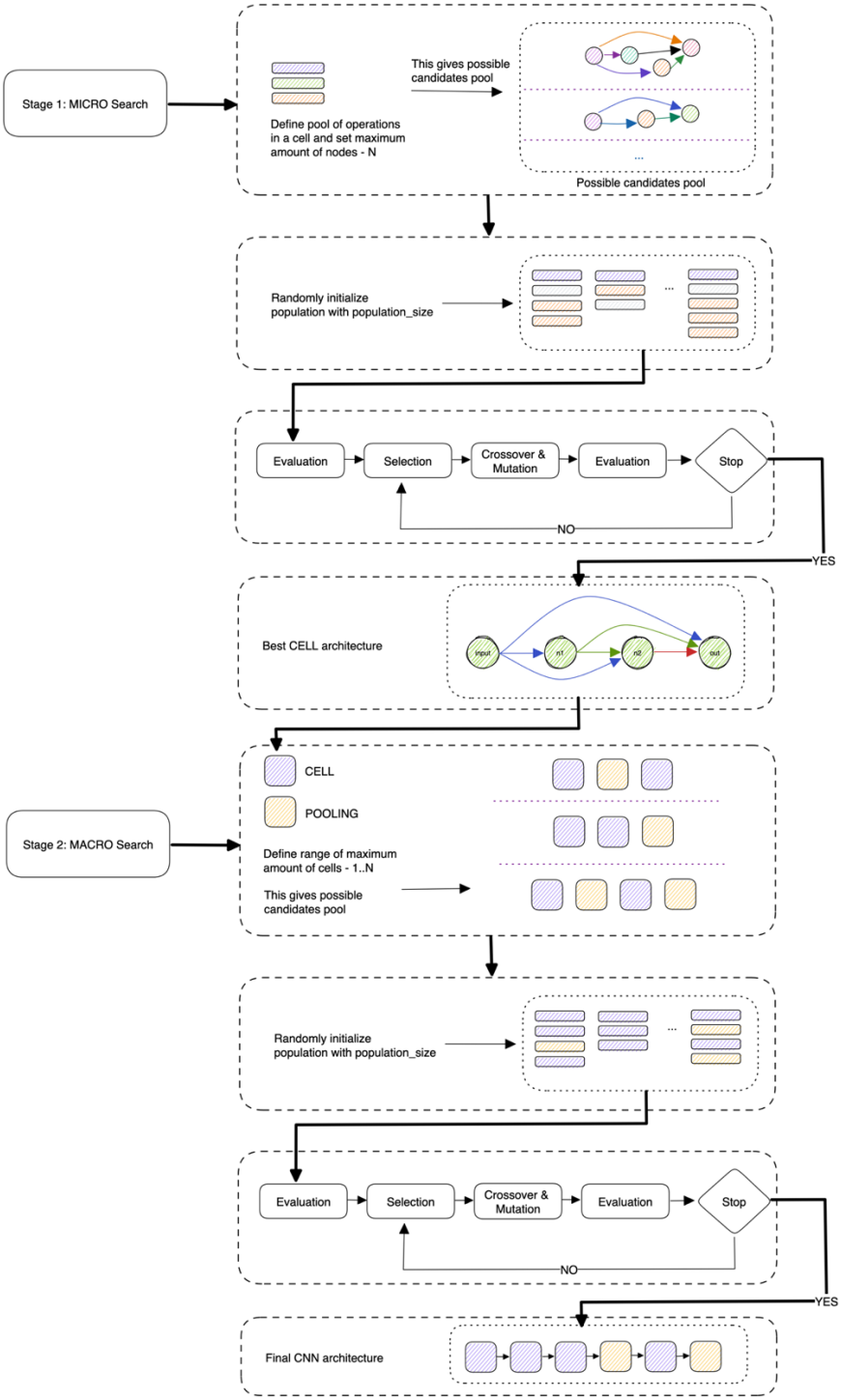


Рисунок 2.1. Схема реалізації розробленого методу

Основним блоком у просторі мікропошуку є комірка, яка складається з вузлів, кожен з яких пов'язаний з усіма іншими. Дві крайні вершини є входом і виходом комірки відповідно.

Стрілками показані операції між вузлами у фіксованому наборі операцій. Всі вершини, крім вхідної, відображають операцію додавання. Це означає, що якщо вершина має декілька входів, то їх сума буде виходом цієї вершини. На рисунку 2.2 показано приклад архітектури комірки, де кількість вузлів дорівнює 4 (включаючи вхід і вихід).

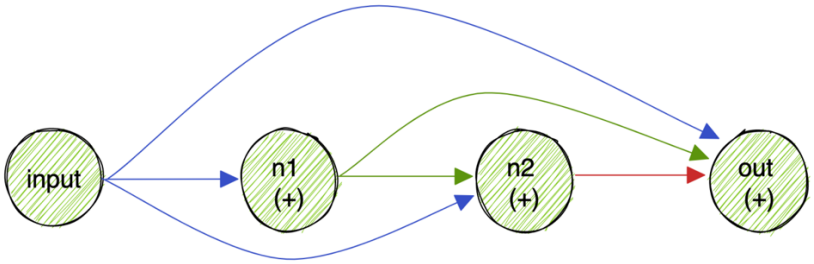


Рисунок 2.2. Приклад комірки

Основними операціями які повторюються у кожній згортковій мережі є згортка та пулінг (субдискретизація). Отже, ці дві операції є мінімальною множиною операцій для побудови такої мережі.

У множині можливих операцій розробленого методу задано такі операції: *skip connection, zeroize, 3×3 convolution, 1×1 convolution, 3×3 average pooling, 3×3 max pooling.* Розглянемо детальніше кожну з них.

1) Skip connection. Вхідні дані після застосування операції не трансформуються. Дану операцію можна розглядати як функцію, що приймає дані та одразу їх повертає. Ця операція дозволяє виходу вузла бути безпосередньо пов'язаним з входом іншого вузла. Ця операція допомагає уникнути затухання градієнтів. Дану операцію ще називають функцією ідентичності.

2) Zeroize. Операція обнулення замінює вихід вузла матрицею нулів. Ця операція може бути використана для видалення вузла з архітектури комірки та ефективно спростити архітектуру. Застосування такої операції між двома вузлами означає, що між ними немає з’єднання.

3) Convolution Block. Вхідні дані перемножуються із ядром згортки, а потім результат сумується. На виході отримуємо нову матрицю. Дана операція відображає стандартну функцію згортки у нейронних мережах.

Кожен блок згортки на етапі мікропошуку складається з таких операцій у наступному порядку:

3.1) Функція активації ReLU застосовує функцію випрямляча до вхідних даних, замінюючи від'ємні значення на нульові. Ця операція вносить нелінійність у мережу.

3.2) Операція згортки застосовує фільтр до вхідних даних для виділення ознак із вхідних даних. Розмір фільтра та їх кількість можна змінювати. У розробленому методі використано фільтри 3×3 та 1×1.

3.3) Операція пакетної нормалізації (batch normalization) нормалізує активацію шару згортки для отримання нульового середнього значення та одиничної дисперсії. Ця операція може покращити стабільність навчання та швидкість сходження мережі.

4) Pooling. Шар пулінгу застосовується для зменшення розмірності вхідних даних. На цьому етапі карти ознак зменшуються після шарів згортки. Популярними типами пулінгу є: середній, максимальний та глобальний. В операціях пулінгу у комірці застосовано тільки два його види: середній з ядром 3×3 та максимальний також з ядром 3×3.

Таким чином, множина операцій мікропошуку визначає набір операцій, які можна використовувати для з'єднання вузлів у комірці. Етап мікропошуку оптимізує кількість і тип операцій, які використовуються для побудови архітектури комірки.

Етап макропошуку передбачає об'єднання кількох комірок послідовно для формування остаточної архітектури нейронної мережі. Також на даному етапі метод визначає після яких комірок необхідно застосувати операцію пулінгу для зменшення розмірності. В якості комірки обирається оптимальна архітектура, знайдена на етапі мікропошуку. Кількість комірок є параметром

простору пошуку, який можна налаштувати для оптимізації архітектури для даного завдання. Таким чином, простір пошуку для другого етапу визначається кількістю комірок та номерами комірок після яких необхідно застосувати пулінг.

Набір можливих операцій у комірці задано як масив, елементи якого йдуть в строгому порядку, як показано на рисунку 2.3.

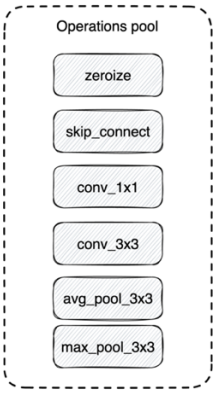


Рисунок 2.3. Набір можливих операцій

Архітектура комірки закодована так: *XXX-XX-X* (рисунок 2.4). Літера X представляє цифру від 0 до 5, тобто $X = \{0, 1, 2, 3, 4, 5\}$. Дані цифри відповідають індексам елементів у множині операцій. Перші три цифри у коді представляють операції, які з'єднують вузол *input* з вузлами *n1, n2, output* відповідно *(op1, op2, op3)*. Наступними двома цифрами закодовано операції між вузлами *n1*, *n2 (op4)* та *n1*, *output (op5)*. Остання цифра кодує операцію між вузлами *n2* та *output (op6)*.

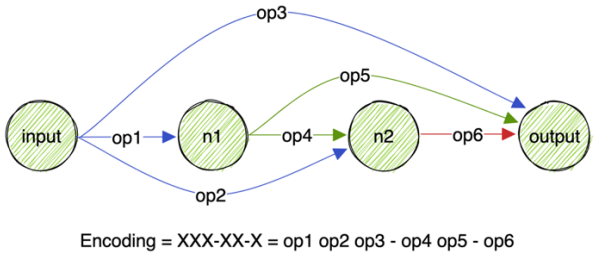


Рис. 2.4. Приклад кодування архітектури комірки

Кодування архітектури на макрорівні задається так: *N-[P]*, де N – кількість комірок, а *P* – масив з індексами комірок, після яких треба застосувати пулінг. Для прикладу кодування *4-[0,2]* відображає архітектуру мережі із чотирма послідовно з'єднаними комірками та шаром пулінгу після першої та третьої комірки.

Для пошуку оптимальної архітектури використано алгоритм AGA [105], який підтримує популяцію потенційних рішень, де кожне рішення представляє унікальну архітектуру ЗНМ. Використаний алгоритм є варіантом стандартного генетичного алгоритму, який запроваджує механізм старіння для покращення різноманітності популяції та запобігання передчасній конвергенції.

Основні кроки алгоритму такі:

1) Оцінка придатності кожної архітектури в популяції за допомогою набору навчальних даних та обчислення точності класифікації.
2) Вибір підмножини архітектур із популяції. Процес відбору базується на випадковій стратегії.
3) Вибір батьківської архітектури та застосування до неї генетичного оператора (мутація), щоб створити нову архітектуру.
4) Оцінка придатності нової архітектури (на основі точності класифікації) і додавання її до популяції.
5) Видалення найстарішої архітектури з популяції.

На першому кроці алгоритму популяція ініціалізується випадковими архітектурами для досягнення заданої кількості - *population_size*. Одночасно з ініціалізацією ці архітектури навчаються та обчислюється їхня точність - *accuracy.* Далі еволюція проходить циклами. На кожному із циклів випадковим чином обирається *sample_size* архітектур. Тоді з поміж цих архітектур обирається та, в якої найвища точність. Вона називається батьківською (*parent*). Застосовуючи мутацію (*mutation*), на основі батьківської архітектури створюється нова дочірня (*child*) архітектура. В даному алгоритмі мутація змінює архітектуру випадковим чином.

Мутація застосовується до кожного елемента в закодованій архітектурі з імовірністю *mutation_prob*. Для прикладу архітектуру комірки закодовано стрічкою 012-23-4. Для того щоб мутувати закодовану архітектуру потрібно пройтися по кожному елементу закодованої стрічки і мутувати його із заданою імовірністю *mutation_prob*. Відповідно після мутації ми отримуємо змінену архітектуру. На рисунку нижче елементи, що піддалися мутації, зображено заштрихованими квадратами. Як бачимо 3, 4 і 5 елемент в кодуванні змінено (було 223, а стало 314) (рисунок 2.5).

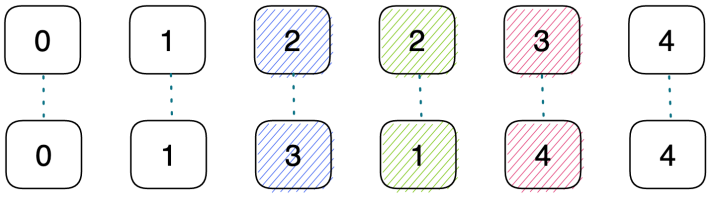


Рисунок 2.5. Приклад мутації архітектури.

Далі дочірня архітектура навчається і додається до популяції. При цьому найстаріша архітектура видаляється із популяції. Це дозволяє підтримувати популяцію фіксованого розміру протягом всіх ітерацій генетичного алгоритму. В кінці алгоритм повертає архітектуру із найвищою точністю.

Псевдокод алгоритму такий:

```
population ← [] (dequeue)
history ← []
while population length < population_size do
     model.architecture ← random()
     model.accuracy ← train(model.architecture)
          append model to population
          append model to history
end while
for i = 0; i < cycles; i++
     sample ← []
     while sample length < sample_size do
          candidate ← random model from population
          append candidate to sample
     end while
     parent ← model with the highest accuracy in sample
     child.architecture ← mutate(parent.architecture)
     child.accuracy ← train(child.architecture)
```

```
        append child to population
        append child to history
        pop oldest model from left of population
    end for
    return highest-accuracy model from history
```

2.4 Комп'ютерні експерименти

Експерименти виконувалися з використанням мови програмування Python та бібліотеки PyTorch. Реалізація програмного забезпечення базується на використанні хмарної інфраструктури AWS, що дозволяє ефективно використовувати ресурси хмарних обчислень. Інфраструктура програмного забезпечення показана на рисунку 2.6.

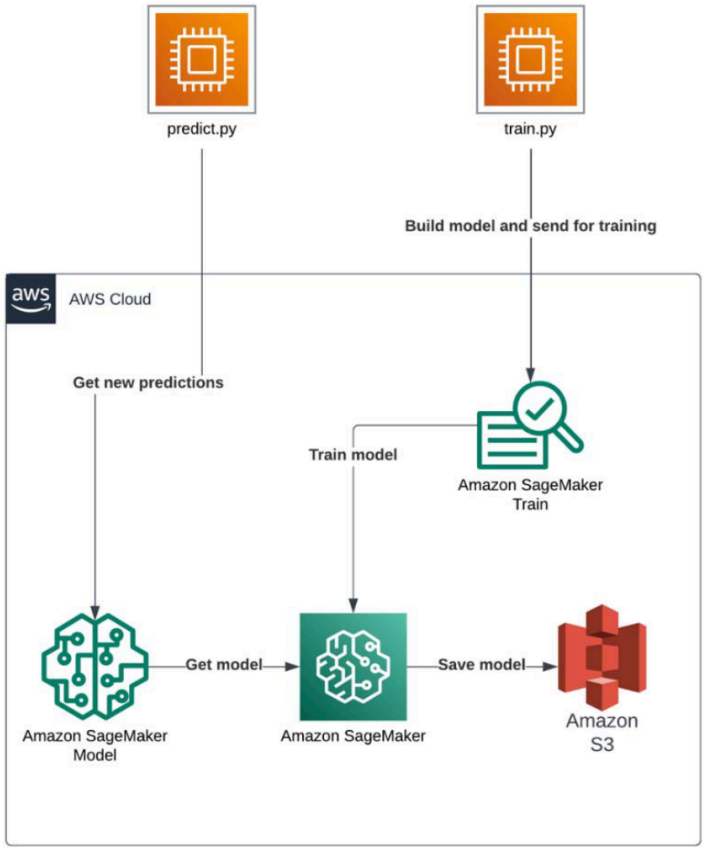


Рисунок 2.6. Хмарна інфраструктура модуля синтезу архітектур CNN

Розроблене програмне забезпечення складається з двох різних файлів Python: *train.py* і *predict.py*, кожен з яких виконує певну функцію у всьому робочому процесі. У файлі *train.py* описано алгоритм синтезу на навчання

архітектур CNN. Для процесу навчання використано сервіс AWS SageMaker. Цей сервіс пропонує масштабоване та контрольоване середовище для ефективного навчання з використанням GPU (у експериментах використано Nvidia V100 GPU). Після завершення процесу навчання модель завантажується і зберігається в сервісі AWS S3, яких пропонує сховище зберігання даних. Щоб зробити модель доступною для використання, в AWS SageMaker генерується її URL-адреса. Ця адреса дозволяє використовувати модель як у хмарній інфраструктурі AWS, так і в інших веб-додатках. Файл *predict.py* дозволяє використовувати навчену модель для класифікації нових даних. Ця програма розроблена як інструмент командного рядка. Щоб отримати результати класифікації зображень, достатньо викликати файл, вказавши шлях до директорії, що містить зображення, як перший параметр. Модель аналізує зображення і видає результати класифікації, які можна використовувати для подальшого аналізу або інтегрувати в інші дослідницькі процедури. Результати класифікації для зручності виводяться у файл *output.txt*, де дані розділені на дві колонки – назва файлу зображення та мітка класу.

В якості набору даних використано той самий набір, що і у попередніх розділах.

Для всіх проведених експериментів задано фіксовані параметри, які наведено в таблицях 2.1 і 2.2.

Таблиця 2.1. Параметри еволюційного алгоритму

Назва параметра	Значення
Optimization_mode	maximize
Population_size	100
Cycles	30

Продовження таблиці 2.1

Sample_size	25
Mutation_probability	0,05
Fitness_function	model accuracy

Таблиця 2.2. Загальні параметри навчання

Назва параметра	Значення
Loss_function	torch.nn.CrossEntropyLoss
Optimizer	Adam, lr = 1e-3
Train_epochs	10
Train_Test_dataset_split_%	80/20

Експеримент 1. У даному експерименті проведено пошук архітектури лише на мікрорівні. Кількість вузлів у комірці задано в діапазоні 3-6. Операції між вузлами у комірці обираються із множини операцій (див. рисунок 2.3). Узагальнену архітектуру мережі для першого експерименту показано на рисунку 2.7. Модель архітектури мережі з параметрами для пошуку архітектури на мікрорівні приведено у таблиці 2.3. Архітектура комірки з параметрами приведена у таблиці 2.4.

Таблиця 2.3. Архітектура моделі

Шар	Парметри	Вихідна форма
L1: Input		64×64×3
L2: Conv	Kernel size = 3×3, stride = 1, padding = 1	64×64×16
L3: CELL	Nodes = 5	64×64×16
L4: Batch norm		64×64×16
L5: ReLU		64×64×16

Продовження таблиці 2.3

L6: Adaptive average pooling	Out nodes = 1	1×1×16
L7: Fully connected	Out nodes = 4	1×4
L8: Output		1×4

Таблиця 2.4. Деталі шару L3 (архітектура комірки)

Шар	Параметри
LC1 Input	
LC2: Convolution Block	Kernel size = 3×3, stride = 1, padding = 1
LC3: LC2 + MaxPool(LC1)	Kernel size = 3×3, stride = 1
LC4: LC1 + AvgPool(LC3)	Kernel size = 3×3, stride = 1
LC5: LC1 + LC4 + AvgPool(LC3)	Kernel size = 3×3, stride = 1

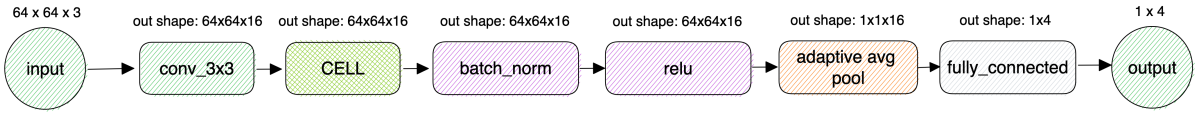


Рисунок 2.7. Загальна структура мережі для першого експерименту

Шар згортки *conv_3×3* (рисунок 2.6) слугує шаром попередньої обробки. Даний шар трансформує вхідне зображення 64×64×3 у 64×64×16, оскільки кількість вхідних та вихідних каналів у комірці задано як 16.

Вибір між малими та великими розмірами ядра залежить від типу об'єктів, які потрібно «захопити». Малі розміри ядра (наприклад, 3×3 або 5×5) добре підходять для початкових шарів, коли важливі базові властивості зображень, такі як краї та текстури. Вони чудово передають дрібні деталі та локальні візерунки. З іншого боку, великі розміри ядра (наприклад, 7×7 або 9×9) зазвичай використовуються в пізніх шарах для вивчення високорівневих абстракцій, оскільки вони ефективно передають ширші, глобальні властивості та складні взаємодії.

На практиці в дизайні нейромереж часто використовують поєднання малих і великих розмірів ядер в шарах згортки. Оптимальний розмір ядра

залежить від таких факторів, як роздільна здатність зображень у наборі даних, обчислювальні ресурси та архітектура мережі.

Результати експерименту наведено у таблиці 2.5. Наведено тільки топ-3 із 30 знайдених архітектур комірки. В таблиці 2.5 відображено пропоновану архітектуру комірки. Для прикладу, запис *0_1: skip_connect* означає, що між першим та другим вузлом застосовано операцію *skip connection* із набору всіх можливих операцій.

Таблиця 2.5. Результати експерименту 1

№ циклу	Точність	F1	Пропонована комірка	К-сть вузлів
20	87.75	86.97	0_1: conv_3x3; 0_2: max_pool_3x3; 1_2: skip_connect; 0_3: none; 1_3: skip_connect; 2_3: avg_pool_3x3; 0_4: skip_connect; 1_4: none; 2_4: avg_pool_3x3; 3_4: skip_connect	*5*
22	86.37	86.25	0_1: none; 0_2: conv_3x3; 1_2: max_pool_3x3; 0_3: none; 1_3: avg_pool_3x3; 2_3: skip_connect;	*4*
3	86.12	85.87	0_1: max_pool_3x3; 0_2: skip_connect; 1_2: conv_1x1; 0_3: max_pool_3x3; 1_3: conv_3x3; 2_3: avg_pool_3x3; 0_4: skip_connect; 1_4: conv_1x1; 2_4: avg_pool_3x3; 3_4: skip_connect; 0_5: none; 1_5: avg_pool_3x3; 2_5: max_pool_3x3; 3_5: max_pool_3x3; 4_5: conv_3x3;	*6*

Загальний час проведення експерименту склав 55 хвилин. Найкращу архітектуру комірки та ROC криву відображено на рисунку 2.8 та 2.9. Кодування цієї комірки таке: *3501-110-44-1*.

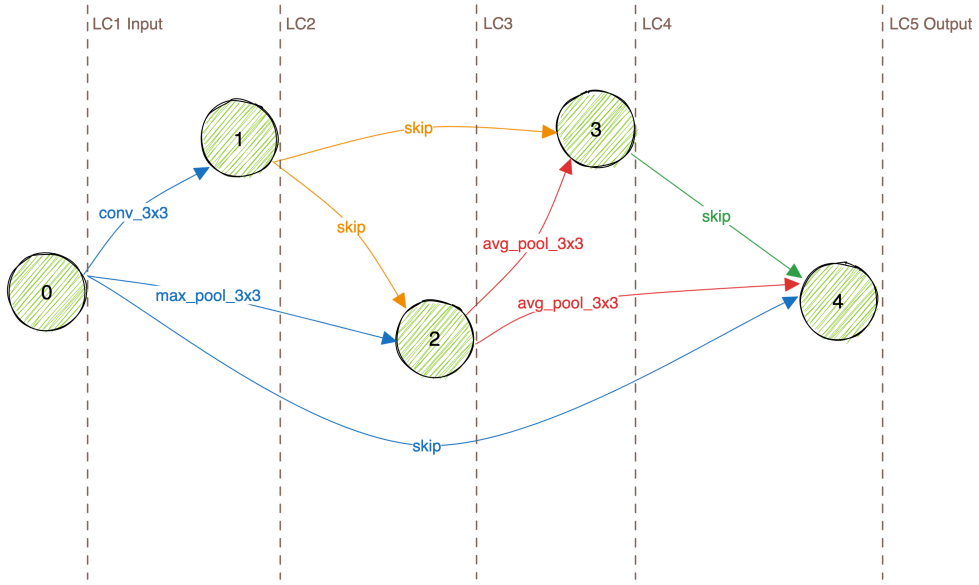


Рисунок 2.8. Архітектура знайденої комірки

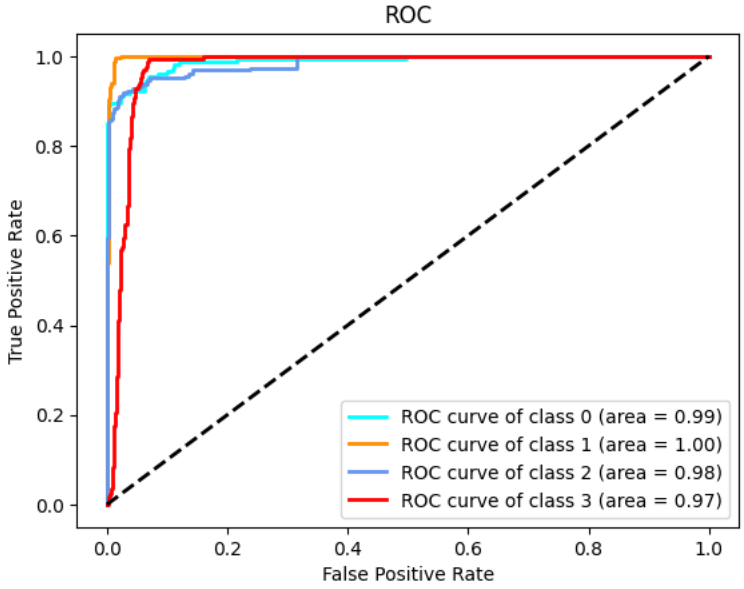


Рисунок 2.9. ROC крива знайденої мережі

Архітектура найкращої комірки має 5 вузлів:

$$A_{CL} = \left\{\left\langle O_{CL_1}\right\rangle;\left\langle O_{CL_2}\right\rangle;\left\langle O_{CL_3}\right\rangle;\left\langle O_{CL_4}\right\rangle;\left\langle O_{CL_5}\right\rangle\right\}. \quad (2.9)$$

У вузлах комірки використовуються наступні операції: операція згортки C; операція максимального пулінгу G_{max}, операція середнього пулінгу G_{avg}, операція S - пропуск з'єднання.

$$A_{CL} = \{\langle O_{CL_1}\rangle; \langle C(O_{CL_1})\rangle; \langle G_{max}(O_{CL_1}), S(O_{CL_2})\rangle; \langle S(O_{CL_2}), G_{avg}(O_{CL_3})\rangle; \langle S(O_{CL_4}), G_{avg}(O_{CL_3})\rangle\} \quad (2.10)$$

Уточнимо параметри операцій, формули 3.11-3.14:

$$O_{CL_1} := \left\{\left\langle I_{inp}\right\rangle\left\langle 64\times 64\times 3\right\rangle\right\}; \quad (2.11)$$

$$G_{\max} := \left\{\left\langle g_{\max}\right\rangle\left\langle 3\times 3\right\rangle\right\}; \quad (2.12)$$

$$C := \left\{\left\langle c\right\rangle\left\langle 3\times 3\right\rangle\right\}; \quad (2.13)$$

$$G_{avg} := \left\{\left\langle g_{avg}\right\rangle\left\langle 3\times 3\right\rangle\right\}. \quad (2.14)$$

Опишемо архітектури для першого обчислювального експерименту, формула 2.15:

$$A_{CNN_1} = \left\{\left\langle I_{inp_1}\right\rangle\left\langle 64\times 64\times 3\right\rangle;\left\langle A_{CL_2}\right\rangle\left\langle node = 5\right\rangle;\left\langle v_3\right\rangle\left\langle \beta = 1{,}04\cdot 10^{-7}, \gamma = 1, \varepsilon = 1\cdot 10^{-5}\right\rangle;$$
$$\left\langle k_4\right\rangle\left\langle ReLu\right\rangle;\left\langle g_{avg_5}\right\rangle\left\langle 3\times 3\right\rangle;\left\langle b_6\right\rangle\left\langle \text{number of layers} = 4\right\rangle;\left\langle f_7\right\rangle\left\langle 1\times 4\right\rangle\right\}. \quad (2.15)$$

Експеримент 2. У даному експерименті проведено пошук архітектури лише на макрорівні. У цьому випадку оптимізується кількість комірок, а також алгоритм визначає після яких комірок слід застосувати операцію максимального пулінгу для ущільнення карт ознак. Кількість комірок задано в діапазоні від 1 до 6. В якості комірки використано найкращу комірку із попереднього експерименту із кількістю вузлів 5. Узагальнену структуру

мережі для другого експерименту показано на рисунку 2.10. Модель архітектури мережі з параметрами для пошуку архітектури на макрорівні приведено у таблиці 2.6.

Таблиця 2.6. Архітектура моделі

Шар	Параметри	Вихідна форма
L1: Input		64×64×3
L2: Conv	Kernel size = 3x3, stride = 1, padding = 1	64×64×32
L3: CELL	Nodes = 5	64×64×32
L4: CELL	Nodes = 5	64×64×32
L5: MaxPool	Kernel size = 3x3, stride = 2	32×32×32
L6: CELL	Nodes = 5	32×32×32
L7: CELL	Nodes = 5	32×32×32
L8: CELL	Nodes = 5	32×32×32
L9: Batch norm		32×32×32
L10: ReLU		32×32×32
L11: Adapt. avg. pooling	Out nodes = 1	1×1×32
L12: Fully connected	Out nodes = 4	1×4
L13: Output		1×4

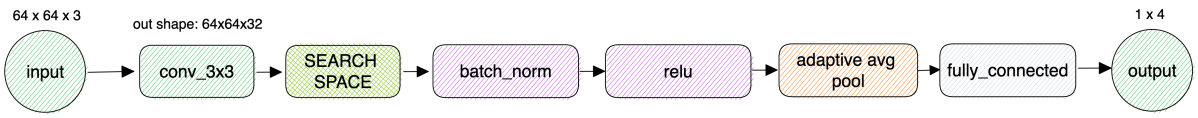


Рисунок 2.10. Загальна архітектура мережі для другого експерименту

У таблиці 2.7 наведено результати цього експерименту. Показано топ-3 із 30 знайдених архітектур.

Таблиця 2.7. Результати експерименту 2

№ циклу	Точність	Пропонована мережа
8	99,125	*5-[1]*
28	97,75	*3-[1,2]*
30	95,25	*2-[0,1]*

Загальний час проведення експерименту склав 62 хвилини. ROC криву та найкращу архітектуру мережі показано на рисунках 2.11 і 2.12 відповідно.

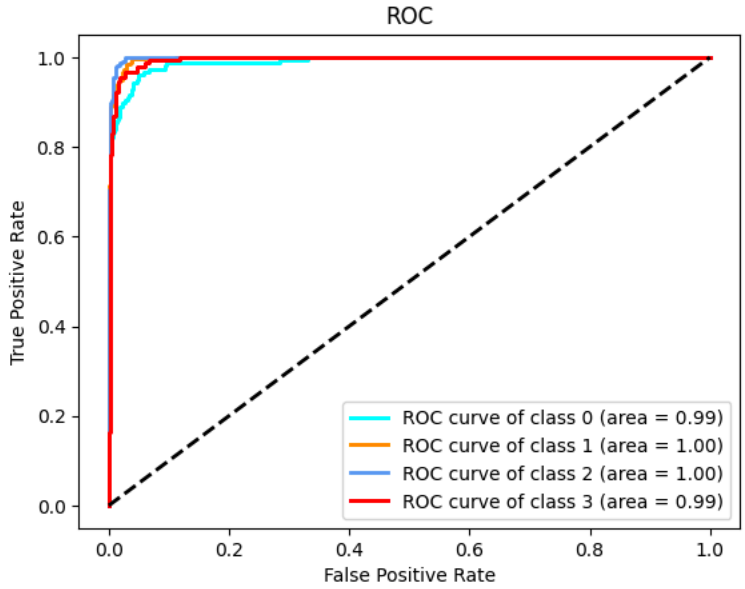


Рисунок 2.11. ROC крива знайденої мережі

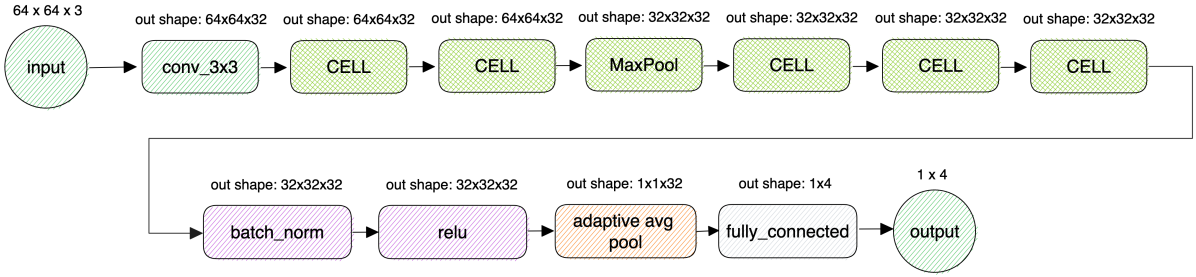


Рисунок 2.12. Архітектура оптимізованої мережі

Опишемо архітектури для другого обчислювального експерименту:

$$A_{CNN_2} = \left\{ \left\langle I_{inp_1} \right\rangle \left\langle 64 \times 64 \times 3 \right\rangle; \left\langle c_2 \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$

$$\left\langle A_{CL_3} \right\rangle \left\langle \text{node} = 5 \right\rangle; \left\langle A_{CL_4} \right\rangle \left\langle \text{node} = 5 \right\rangle; \left\langle g_{\max_5} \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 1 \right\rangle;$$

$$\left\langle A_{CL_6} \right\rangle \left\langle \text{node} = 5 \right\rangle; \left\langle A_{CL_7} \right\rangle \left\langle \text{node} = 5 \right\rangle; \left\langle A_{CL_8} \right\rangle \left\langle \text{node} = 5 \right\rangle; \left\langle v_9 \right\rangle \left\langle \beta = 1{,}04 \cdot 10^{-7}, \gamma = 1, \varepsilon = 1 \cdot 10^{-5} \right\rangle;$$

$$\left\langle k_{10} \right\rangle \left\langle ReLu \right\rangle; \left\langle g_{ad\_avg_{11}} \right\rangle \left\langle \text{out nodes} = 1 \right\rangle; \left\langle b_{12} \right\rangle \left\langle \text{out nodes} = 4 \right\rangle; \left\langle f_{13} \right\rangle \left\langle \text{clauses} = 4 \right\rangle \Big\}.$$

Далі проаналізовано класичні архітектури згорткових нейронних мереж AlexNet, LeNet-5, VGG-16, ResNet-18, ResNet-50 та MobileNetV3. Крім того, були проведені експерименти з класифікації цитологічних зображень. Всі зображення були попередньо оброблені відповідно до вхідних розмірів кожної архітектури. Для навчання мереж використано той самий набір цитологічних зображень, аналогічно до інших експериментів.

В архітектурі LeNet-5 вхідними зображеннями є зображення у відтінках сірого з розмірами 32×32×1. Потім додаються дві пари шарів згортки з кроком 2 і шари середнього пулінгу з кроком 1. Повнозв'язні вихідні шари використовують функцію активації Softmax. Загалом ця мережа містить 60 000 параметрів [106].

Експеримент. Вхідні зображення конвертовано у роздільну здатність 32×32 пікселі відповідно до параметрів архітектури. Кількість епох навчання дорівнює 30, а розмір пакету – 10. Як активаційну функцію використано ReLU. В результаті експерименту ця мережа досягла точності 65%. ROC-крива показана на рисунку 2.13.

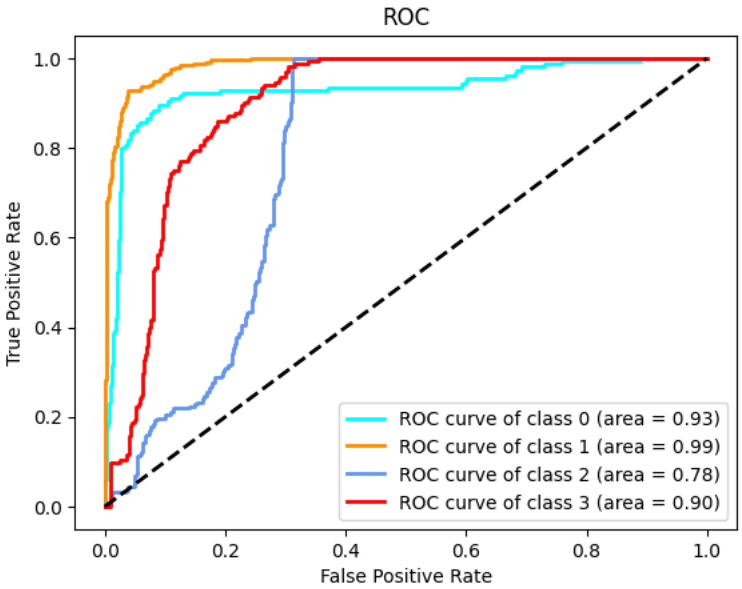


Рисунок 2.13. ROC крива LeNet-5

Мережа AlexNet – це 8-шарова мережа зі збільшеними згортковими фільтрами порівняно з LeNet-5. Ця мережа має 5 згорткових шарів і 3 повністю з'єднаних шари. Загальна кількість параметрів становить приблизно 60 мільйонів. Суттєвим недоліком мережі є велика кількість гіперпараметрів (і, відповідно, більший час навчання). Крім того, в мережі представлено нову концепцію локальної нормалізації (Local Response Normalization) [24]. Ця концепція полягає в нормалізації вхідних даних перед основними шарами нейронної мережі.

Експеримент. Вхідні зображення конвертовано у роздільну здатність 227×227 пікселі відповідно до параметрів архітектури. Кількість епох навчання – 50, розмір пакету – 12. Точність цієї мережі після навчання становить 77%. ROC-крива показана на рисунку 2.14.

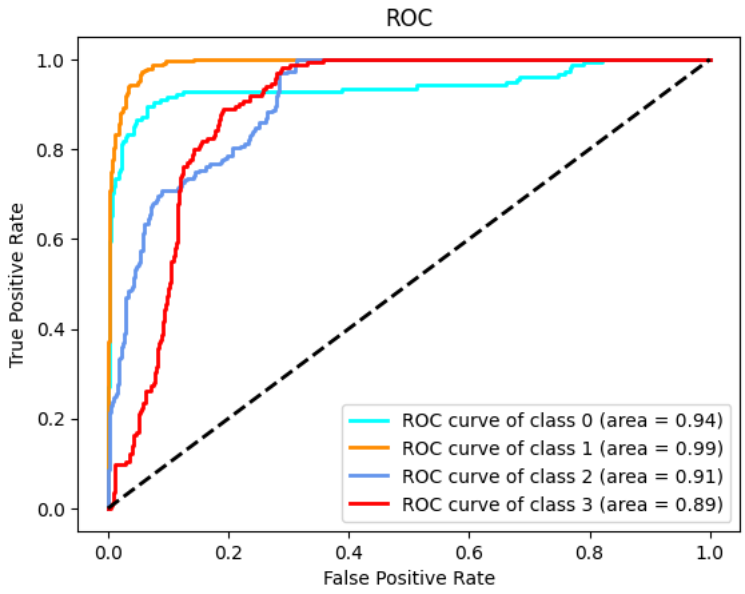


Рисунок 2.14. ROC крива AlexNet

У VGG-16 вирішено проблему великої кількості гіперпараметрів у AlexNet. Для цього замість великих фільтрів з розміром ядра 3×3 на 1-му та 2-му шарах згортки використовуються декілька фільтрів з розміром ядра 11 та 5. Мережа має 16 шарів з фільтрами згортки 3×3 і максимальним пулінгом 2×2 з кроком 1, а також однаковим відступом для збереження розміру даних між

шарами. Вхідним шаром є кольорове зображення 224×224×3 пікселів, за яким слідують 5 пар шарів згортки (фільтри: 64, 128, 256, 512, 512) і шар максимального пулінгу. Вихід останньої пари подається на три повнозв’язні шари і має функцію активації Softmax на останньому шарі. Загалом ця мережа має 138 мільйонів параметрів [70], [107]. Недоліками мережі є довгий час навчання, велика кількість параметрів і проблема затухання градієнтів. Через цю проблему стало складніше навчати глибокі згорткові мережі, а після VGG-16 нові архітектури стали ще глибшими. Тому у 2015 році було запропоновано рішення, засноване на концепції пропускання з'єднань.

Експеримент. Вхідні зображення конвертовано у роздільну здатність 224×224 пікселі відповідно до параметрів архітектури. Кількість епох навчання – 50, розмір пакету – 8. Точність мережі VGG становить 91%. ROC-крива зображена на рисунках 2.15.

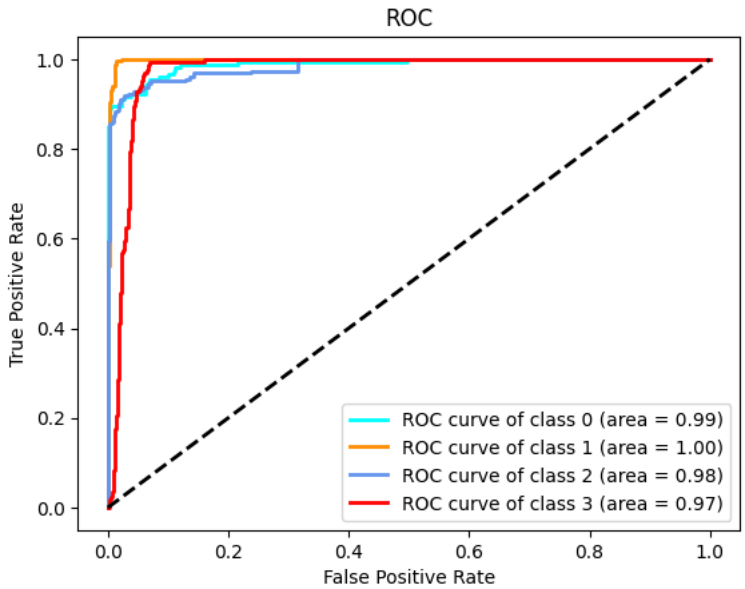


Рисунок 2.15. ROC крива VGG-16

ResNet-18 — це глибока згорткова нейронна мережа з 18 шарами, відома завдяки використанню залишкових зв'язків, які допомагають у вирішенні проблеми затухаючих градієнтів в глибоких нейронних мережах. Ця мережа складається з 17 згорткових шарів і 1 повнозв’язного шару в кінці. Загальна

кількість параметрів у ResNet-18 значно менша, ніж у AlexNet, і становить приблизно 11.7 мільйонів параметрів. Однією з ключових особливостей ResNet-18 є введення залишкових блоків. Кожен блок має залишкове з'єднання, що дозволяє передавати дані в обхід одного або декількох шарів. Така архітектура полегшує навчання глибших мереж, дозволяючи пряме поширення градієнтів через залишкові з'єднання, ефективно вирішуючи проблему затухаючих градієнтів [70].

Експеримент. Вхідні зображення конвертовано у роздільну здатність 224×224 пікселі відповідно до параметрів архітектури. Кількість епох навчання – 50, а розмір партії – 12. Точність цієї мережі після навчання становить 90%. ROC-крива показана на рисунку 2.16.

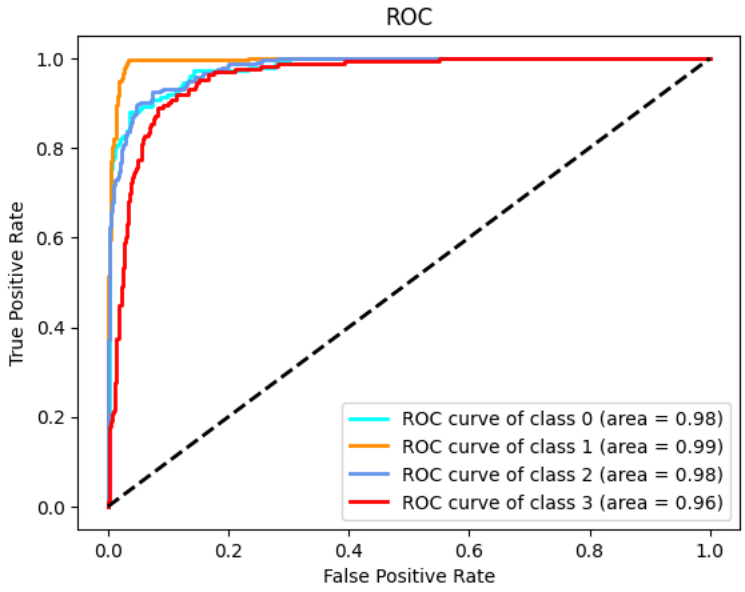


Рисунок 2.16. ROC крива ResNet-18

ResNet-50 — це глибший і складніший варіант архітектури ResNet порівняно з ResNet-18, що має 50 шарів. Дана мережа розроблена для забезпечення більшої точності в задачах розпізнавання зображень за рахунок збільшення глибини мережі. ResNet-50 має близько 25.6 мільйонів параметрів, що більш ніж у два рази перевищує кількість параметрів у ResNet-18.

Експеримент. Вхідні зображення конвертовано у роздільну здатність 224×224 пікселі відповідно до параметрів архітектури. Кількість епох навчання – 50, а розмір партії – 12. Точність цієї мережі після навчання становить 93%. ROC-крива показана на рисунку 2.17.

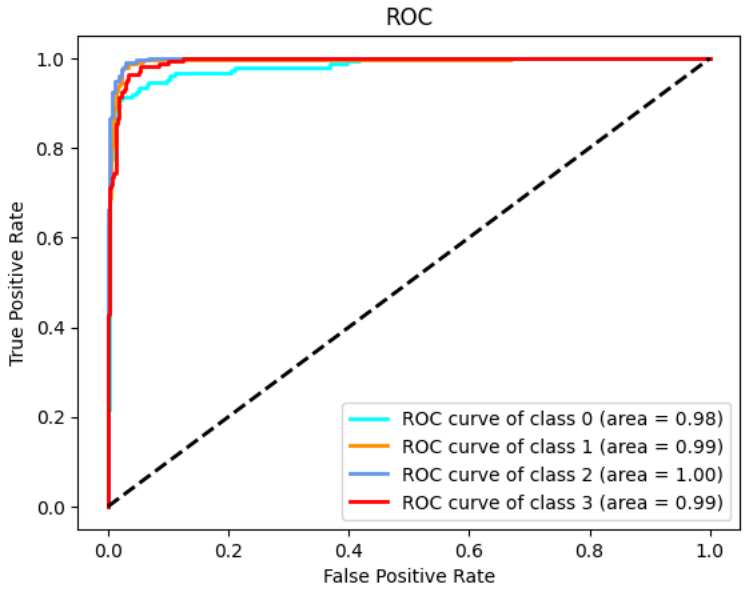


Рисунок 2.17. ROC крива ResNet-50

MobileNetV3 — це високоефективна мережа, розроблена для мобільних та периферійних пристроїв, де обчислювальні ресурси обмежені. У порівнянні з традиційними архітектурами, такими як ResNet-18 і ResNet-50, дана мережа зосереджена на оптимізації швидкості та ефективності при збереженні високої точності [108]. MobileNetV3 має набагато менше параметрів порівняно з ResNet-18 і ResNet-50. Таке зменшення параметрів призводить до зменшення розміру моделі та обчислювальних вимог, що робить її ідеальною для мобільних пристроїв. Хоча ResNet-50 і навіть ResNet-18 можуть перевершити MobileNetV3 з точки зору вихідної точності для певних завдань, MobileNetV3 пропонує кращий компроміс між точністю і ефективністю. Дана мережа оптимізована для малопотужних пристроїв, забезпечуючи хорошу продуктивність з меншими обчислювальними ресурсами та енергоспоживанням.

Експеримент. Вхідні зображення також конвертовано у роздільну здатність 224×224 пікселі. Кількість епох навчання дорівнює 50, а розмір партії – 12. Точність цієї мережі після навчання становить 85%. ROC-крива показана на рисунку 2.18.

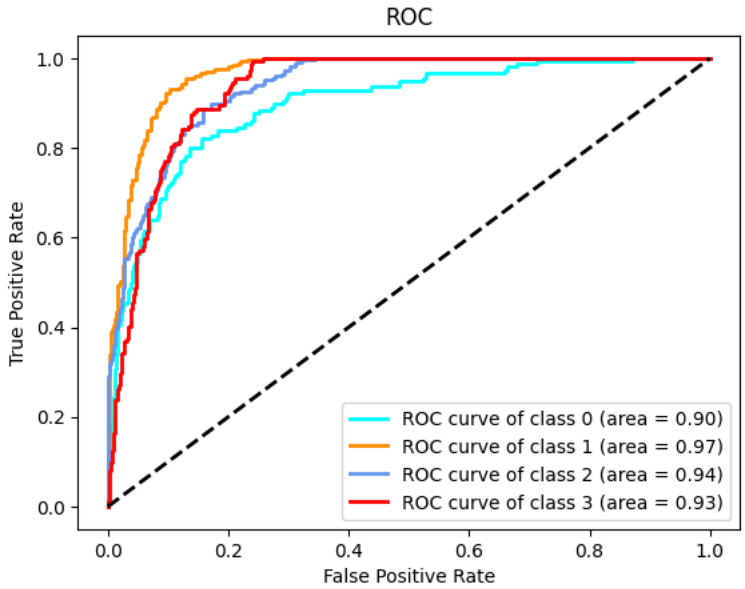


Рисунок 2.18. ROC крива MobileNet

Результати експериментів показали, що класичні архітектури згорткових нейронних мереж мають низьку точність класифікації. Найвище значення точності серед класичних архітектур показала мережа ResNet-50 — 93%. Така точність є досить низькою, особливо в області класифікації біомедичних зображень, де від діагнозу практично залежить подальше лікування пацієнта.

Ручне проектування архітектур нейронних мереж залежить від досвіду дослідника, є довготривалим процесом, та вимагає глибоких і специфічних знань у сфері машинного навчання. Експерименти показали високу точність класифікації цитологічних зображень. Розроблений метод автоматичного пошуку архітектури може бути використаний для синтезу архітектури для класифікації інших класів зображень та зображень з вищою роздільною здатністю. Збільшення роздільної здатності вхідних зображень може призвести до зниження точності класифікації. Для забезпечення необхідної точності

необхідно збільшувати кількість комірок. Таким чином, обмеженнями розробленого методу є кількість комірок та кількість операцій у комірці, оскільки вони обмежені конкретними значеннями в реалізації.

Синтезована розробленим методом архітектура показала вищу точність класифікації, ніж відомі архітектури, такі як VGG-16, AlexNet, LeNet-5, ResNet-18, ResNet-50 та MobileNetV3, як показано у таблиці 2.8.

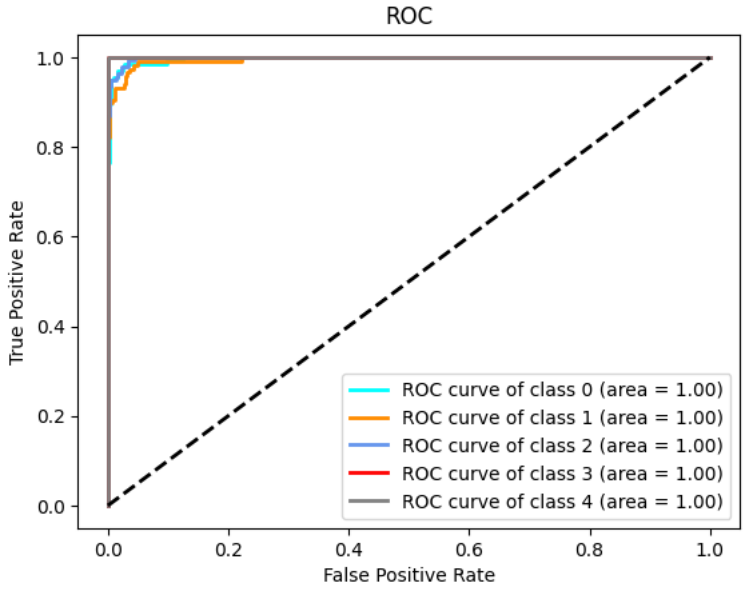
Таблиця 2.8. Результати класифікації відомими архітектурами

Аріхтектура	Точність %
LeNet-5	65
AlexNet	77
VGG-16	91
ResNet-18	90
ResNet-50	93
MobileNetV3	85

У таблиці 2.9 наведено результати порівняння точності класифікації синтезованої архітектури з іншими дослідженнями в області класифікації раку. Для порівняння використано 5 класів набору даних Sipakmed [109]. Мережа навчалася протягом 50 епох. Точність мережі після навчання становить 96.86%. ROC-крива показана на рисунку 2.19.

Таблиця 2.9. Порівняння результатів знайденої розробленим методом архітектури з відомими дослідженнями

Дослідження	Метод	Набір даних	Точність
[110]	AlexNet	Sipakmed	93,33
[109]	Adapted VGG-19	Sipakmed	95,35
[111]	Ensemble learning	Sipakmed	94,09
[112]	DenseNet201	Sipakmed	95,79
[113]	ResNet101	Sipakmed	93
Власне	Cell-based custom CNN	Sipakmed	96,86



Риснок 2.19. ROC крива пропонованої моделі на наборі даних Sipakmed

Основні шари, які повторюються в розглянутих класичних архітектурах — це згортка та пулінг. Вони можуть бути об'єднані в один блок (комірку). Розглянуті архітектури відрізняються кількістю шарів у блоці та їх порядком. Крім того, ще одним недоліком проаналізованих архітектур є те, що всі шари в комірці з'єднані послідовно. Така організація зв'язків між шарами в комірці не є гнучкою і може суттєво вплинути на кінцеву точність класифікації. Вона також збільшує ймовірність виникнення проблеми затухаючих градієнтів. Розроблений метод позбавлений цього недоліку завдяки архітектурним особливостям комірки. Вузли в комірці з'єднані не в класичний послідовний спосіб, як показано на прикладі шарів проаналізованих вище архітектур, де кожен наступний шар отримує вхідні дані тільки від попереднього. Натомість, у розробленому методі кожен наступний вузол у комірці може отримувати вхідні дані від усіх попередніх вузлів, залежно від обраної операції між ними.

Крім того, в цьому дисертаційному дослідженні використовуються невеликі зображення, зокрема, розміром 64×64 пікселів. Такі зображення зазвичай потребують меншої кількості шарів для ефективної обробки, оскільки вони містять менше складних ознак порівняно з великими зображеннями. Тому рішення обмежити кількість шарів у пропонованому методі також зумовлено

природою вхідних даних. Вибираючи саме такий діапазон шарів, ми прагнемо створити модель, яка одночасно ефективно навчається на невеликих за розміром зображеннях і є доцільною з точки зору використання ресурсів. Такий підхід гарантує, що пропонована модель залишається практичною та ефективною, особливо в сценаріях, де оптимізація витрат і ресурсів є пріоритетом.

2.5 Висновки до розділу 2

1. Розроблено метод синтезу архітектур згорткових нейронних мереж, який складається з двох фаз: фази мікропошуку та фази макропошуку. Фаза мікропошуку передбачає задання кількості вузлів і операцій між ними. Фаза макропошуку передбачає оптимізацію кількості комірок і операцій між ними.

2. В результаті експериментів синтезовано нову архітектуру ЗНМ мережі, що показала точність класифікації 99,125% на цитологічних зображеннях.

3. Здійснено порівняння синтезованої ЗНМ архітектури із відомими класифікаторами біомедичних зображень на основі ЗНМ мереж. Розроблена архітектура нейронної мережі перевершує відомі рішення по точності класифікації приблизно на 5%.

РОЗДІЛ 3. МЕТОДИ АВТОМАТИЧНОГО СИНТЕЗУ БІОМЕДИЧНИХ ЗОБРАЖЕНЬ ТА АРХІТЕКТУР ГЕНЕРАТИВНО-ЗМАГАЛЬНИХ НЕЙРОННИХ МЕРЕЖ

У даному розділі розроблено метод синтезу архітектур ГЗМ мереж, а також метод для синтезу біомедичних зображень на основі ГЗМ. Проведено порівняння архітектур, що синтезовані розробленим методами, з іншими відомими архітектурами для синтезу біомедичних зображень.

Результати розділу опубліковано у працях автора [5], [6], [15], [55].

3.1 Метод синтезу архітектур генеративно-змагальних нейронних мереж

Синтезовані зображення допомагають покращити навчання моделей машинного навчання (зокрема класифікаторів) шляхом розширення існуючих наборів навчальних даних, які часто є малої розмірності [114].

Складний процес синтезу архітектур ГЗМ вручну ускладнює їх використання. Ручне проектування архітектури ГЗМ мереж вимагає глибокого розуміння принципів машинного навчання та знання специфіки предметної області. Цей процес є трудомістким, тривалим і часто вимагає від розробників прийняття низки складних рішень. Наприклад, розробникам доводиться вибирати між різними типами шарів, функціями активації та методами оптимізації, які мають великий вплив на продуктивність нейронної мережі.

Одна з основних проблем ручного проектування архітектури є масштабованість. Чим складнішими стають біомедичні зображення, тим складнішими стають архітектури ГЗМ. Ручне проектування архітектури залежить від досвіду та інтуїції розробника.

У медицині ГЗМ використовуються для розширення навчальних даних, що має важливе значення для вирішення проблеми дефіциту анотованих медичних зображень. Автори роботи [115] представили новаторський підхід до синтезу зображень сітківки ока за допомогою методу генеративних моделей. На основі генеративних моделей були синтезовані реалістичні зображення сітківки

ока, які були використані для навчання діагностичних алгоритмів. Аналогічно, Frid-Adar та ін. використали ГЗМ для розширення наборів даних зображень уражень печінки, які були використані класифікатором [58]. Таким чином, автори продемонстрували практичну корисність ГЗМ для розширення наборів даних і підвищення точності нейронних мереж у задачах класифікації біомедичних зображень.

Оскільки сфера NAS була першочергово зосереджена тільки на згорткових нейронних мережах, то NAS для генеративних мереж це відносно новий напрямок досліджень. Ранні роботи в NAS для ГЗМ були зосереджені на пошуку ефективних архітектур, які могли б генерувати високоякісні зображення із зменшеними обчислювальними ресурсами.

Основоположною роботою в цій області є стаття [116], в якій розроблено метод автоматичного пошуку архітектур ГЗМ. Автори застосували метод навчання з підкріпленням для оптимізації архітектури ГЗМ. Фреймворк AutoGAN продемонстрував, що автоматичне проектування архітектури може конкурувати з моделями, створеними людиною. Для оцінки методу автори використали метрику FID, фінальне значення якої становить 12.42. Час пошуку архітектури становить 48 GPU годин. Основними недоліками роботи є: обмежений простір пошуку, фіксований дискримінатор, безумовна (unconditional) генерація зображень.

Розвитком попереднього дослідження є робота [117]. У статті автори застосували генетичний алгоритм для оптимізації архітектури нейронної мережі та розширили простір пошуку такими операціями як skip connection, згортка з різними розмірами ядра (1, 3 та 5) [118]. Перевагою даної роботи є пошук архітектури генератора і дискримінатора. Пошук архітектури розділено на два етапи – на першому здійснюється пошук архітектури генератора (дискримінатор фіксований), а на другому – дискримінатора. Як зазначають автори, такий підхід дозволив значно стабілізувати навчання ГЗМ.

Експерименти зайняли 1.2 GPU дні та показали FID 9.91 на датасеті CIFAR-10. Недоліком роботи є неможливість генерувати зображення за мітками.

З точки зору оптимізації архітектур ГЗМ, дослідники визначили кілька ключових факторів, які впливають на продуктивність мереж. Брок та ін. представили концепцію масштабування ГЗМ, показавши, що більші моделі і розміри пакетів (batch size) можуть покращити якість зображень [68]. Ця концепція має важливе значення для автоматичного пошуку архітектур ГЗМ. Алгоритми пошуку повинні враховувати компроміс між складністю моделі та обчислювальною можливістю.

Незважаючи на досягнутий прогрес, в літературі залишаються значні прогалини. Проаналізовані статті в сфері NAS для ГЗМ не фокусуються на синтезі біомедичних зображень і використовують відкритий датасет – CIFAR. Це можна пояснити складністю отримання самих датасетів біомедичних зображень, що в свою чергу ще більше підкреслює актуальність їх синтезу. Також розглянуті методи мають обмежений простір пошуку, не використовують найсучасніші операції (ELU, Self-Attention) в просторі пошуку, не синтезують зображення за мітками (використовують безумовну генерацію).

Таким чином у даному розділі розроблено метод автоматичного пошуку архітектур ГЗМ. Автоматичний метод пошуку архітектур ГЗМ дозволить швидше і ретельніше досліджувати великий простір пошуку (типи шарів, функцій активації і т.д.), знаходячи найкращі мережеві конфігурації, які можуть бути пропущені людьми. Такий спосіб зменшить час синтезу архітектур ГЗМ і підвищить ефективність процесу. На основі розробленого методу запропнований метод синтезу біомедичних зображень.

3.1.1 Постановка задачі оптимізації архітектури генеративно-змагальної мережі

Введемо такі умовні позначення:

I_t – множина навчальних зображень;

I_G – множина синтетичних зображень;

G – генератор;

D – дискримінатор;

L – множина шарів дискримінатора;

Q – множина шарів генератора;

n – кількість шарів дискримінатора;

m – кількість шарів генератора;

i – індекс шарів дискримінатора;

j – індекс шарів генератора;

$CELL_D$ – комірка дискримінатора;

$CELL_G$ – комірка генератора;

o – кількість вузлів в комірці генератора;

p – кількість вузлів в комірці дискримінатора;

A_G – архітектура генератора;

A_D – архітектура дискримінатора;

O_G – множина операцій генератора;

O_D – множина операцій дискримінатора;

P_G – множина параметрів операцій генератора;

P_D – множина параметрів операцій дискримінатора;

$V(G, D)$ – функція втрат для генератора та дискримінатора;

q – потужність множини тренувальної вибірки;

r – потужність множини згенерованої вибірки;

$P_{I_t}(x)$ – густина функції розподілу навчальної вибірки;

$P(z)$ – густина функції розподілу шуму генератора;

$E(x)$ – математичне сподівання випадкової величини x.

M_F – метрика FID;

$\langle \mathrm{C} \rangle$ – множина згорткових шарів;

$\langle K \rangle$ – множина функцій активацій;

$\langle U \rangle$ – множина функцій Upsample block;

$\langle W \rangle$ – множина функцій комірки генератора $CELL_G$;

$\langle Z \rangle$ – множина функцій комірки дискримінатора $CELL_D$;

$\langle T \rangle$ – множина операцій Dawnsample block;

S – функція Self Attention.

Нехай задана тренувальна вибірка зображень I_t потужністю q. Необхідно згенерувати множину зображень I_G потужністю r, причому $r \gg q$. Для генерування зображень I_G використаємо ГЗМ, яка складається з генератора і дискримінатора. Крім того задана архітектура генератора і дискримінатора A_G і A_D відповідно. Архітектуру дискримінатора опишемо так:

$$A_{D=}\{L_i, i = \overline{1,n}\}, \tag{3.1}$$

де L – множина шарів дискримінатора а архітектуру генератора наступним чином:

$$A_G = \{Q_j, j = \overline{1,m}\}. \tag{3.2}$$

Множину операцій дискримінатора представимо так:

$$O_D = \{\langle C \rangle; \langle K \rangle; \langle Z \rangle; \langle T \rangle\}, \tag{3.3}$$

а множину операцій генератора наступним чином:

$$O_G = \{\langle C \rangle; \langle K \rangle; \langle U \rangle; S\}. \tag{3.4}$$

Тоді необхідно провести двохрівневу оптимізацію архітектур дискримінатора і генератора, тобто:

$$A_D = arg \min_{O_D, P_D} M_F(P_{D,} O_D, I_t, I_D), \tag{3.5}$$

$$A_G = arg \min_{O_G, P_G} M_F(P_{G,} O_G, I_t, I_G). \quad (3.6)$$

При цьому функція втрат дискримінатора і генератора буде визначатися так:

$$V(G,D) = \min_G \max_D F(G,D) = E_{x \sim P_{I_t}(x)}[logD(x)] + E_{z \sim p_z(z)}[\log\left(1 - D\big(G(z)\big)\right)] \quad (3.7)$$

3.1.2 Метод автоматичного пошуку архітектури генеративно-змагальної мережі

Визначення простору пошуку є першим важливим кроком у розробці архітектур ГЗМ. На цьому кроці визначається діапазон можливих типів шарів та їх комбінацій, які будуть застосовуватися в процесі побудови моделі.

Простір пошуку в розробленому методі базується на основі комірок, як і у методі синтезу архітектур ЗНМ. Комірка складається із певної кількості вузлів та операцій між ними (див. рисунок 2.3).

Після аналізу сучасних архітектур ГЗМ визначено набір шарів, які найчастіше застосовуються в побудові цих архітектур. До них відносяться такі операції: операція згортки ядром 1×1, 3×3 та 5×5, операції max- та average-pooling, механізм self-attention. Також архітектури генератора та дискримінатора в більшості ГЗМ складаються із повторюваних конволюційних блоків, а вхідний вектор шуму трансформується в матрицю розміром 4×4. Кількість таких блоків варіюється залежно від роздільної здатності згенерованого зображення. Відповідно якщо генератор синтезує зображення роздільністю 64×64, то він складається з 4 конволюційних блоків.

Повний набір можливих операцій у комірці є таким самим як і для методу синтезу архітектур ЗНМ у попередньому розділі, але в конволюційному блоці замінено порядок операцій на згортку, активацію ELU, батч-нормалізацію. Також додано нову операцію Self-attention, яку винесено поза межі комірок Інтеграція механізму Self-Attention в ЗНМ дозволяє зосередитися на найбільш

інформативних ознаках, що може бути корисним для виявлення об'єктів, сегментації і розпізнавання зображень [119].

Простір пошуку генератора та дискримінатора показано на рисунку 3.1.

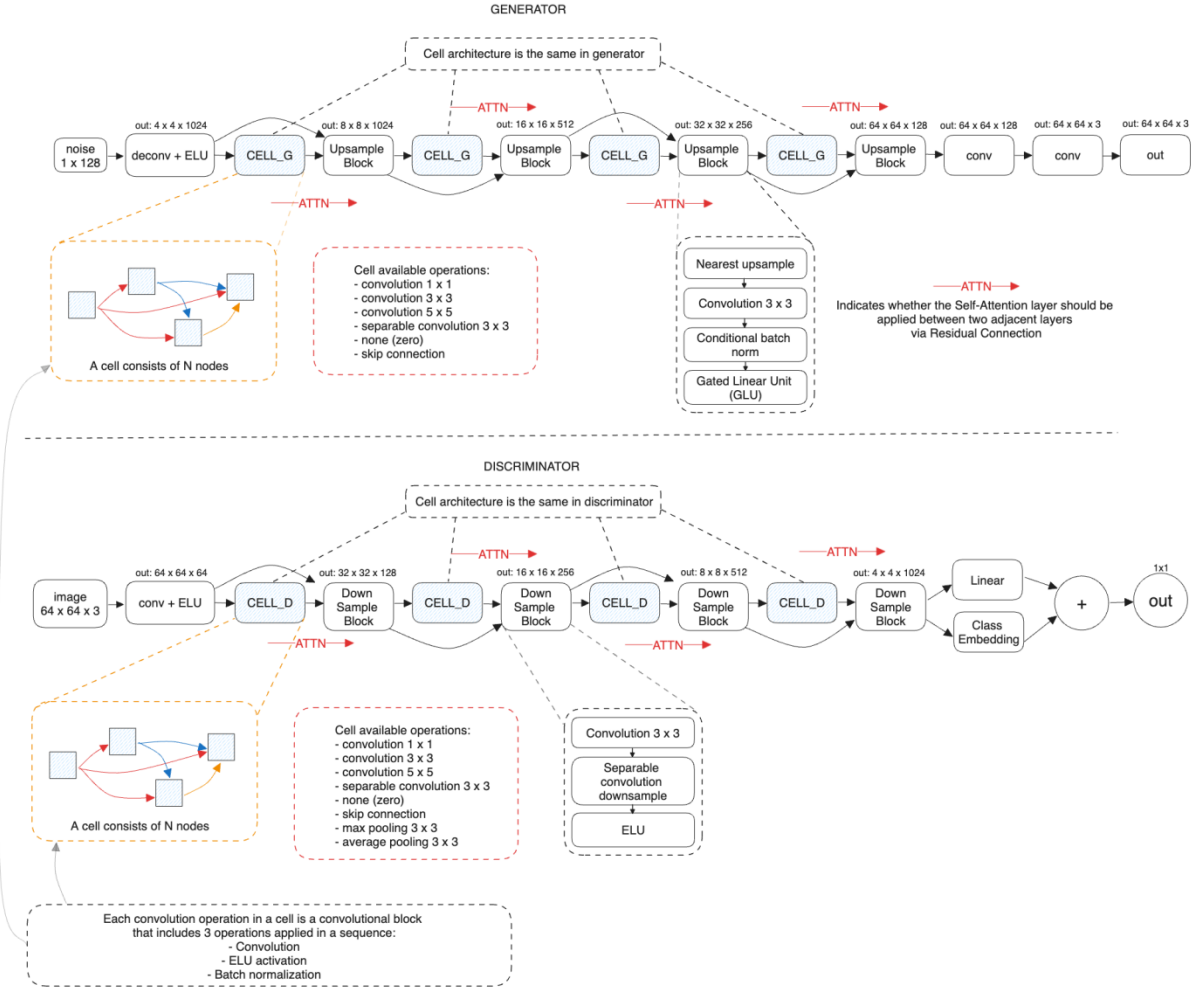


Рисунок 3.1. Простори пошуку генератора та дискримінатора

У розробленому методі застосовано стратегію Conditional GAN, яка дозволяє синтезувати зображення на основі мітки класів. Відповідно в моделі генератора використано операцію Conditional Batch Normalization в кожному UpSample блоці. В моделі дискримінатора використано Embeddig шар в поєднанні з повнозв'язним шаром. Сума виходів двох цих шарів і є виходом моделі дискримінатора.

Генератор складається із одного шару деконволюції для конвертації вектора вхідного шуму у розмірність 4×4×1024, чотирьох комірок, чотирьох

блоків для збільшення роздільності в два рази та двох згорткових шарів на виході мережі. Кінцева роздільна здатність синтезованого зображення становить 64×64×3, що є достатньою для навчання класифікаторів, як показано в експериментах розділу 2.

Основним елементом в просторі пошуку генератора є комірка, яка складається з *o* вузлів. Доступними операціями у комірці визначено такі: згортка ядром 1×1, 3×3, 5×5; separable convolution ядром 3×3; zero; skip connection. Архітектура комірки залишається однаковою для всієї моделі генератора. Відповідно простір пошуку генератора зводиться до визначення кількості вузлів у комірці, вибору операцій між вузлами, а також вибору шарів після яких потрібно застосувати операцію Self-Attention через залишкове з'єднання (на рисунку 3.1 показано червоними стрілками *ATTN*).

Простір пошуку генератора закодовано так: *{N, C, [ATTN]}*. Деталізуємо ці параметри.

Параметр *N* – кількість вузлів у комірці. Мінімальне значення 3, максимальне – 5. Таке рішення було зумовлене обмеженими обчислювальними ресурсами. Крім того, як вже було зазначено, в цьому дослідженні використовуються переважно невеликі зображення, зокрема, розміром 64×64 пікселі. Цей аспект суттєво впливає на архітектури мереж, оскільки такі зображення зазвичай потребують меншої кількості шарів. Тому рішення обмежити кількість шарів у моделі також зумовлене природою самих даних. А також взято до уваги те, що в експерименті по синтезу архітектур згорткових мереж найкраща комірка має також 5 вузлів.

Параметр *C* – архітектура комірки, в якій закодовано операції між вузлами. Представлена стрічкою *xxx-xx-x* довжина якої варіюється залежно від кількості вузлів у комірці. Наприклад, для комірки з кількістю вузлів 3 закодована стрічка може мати вигляд *12-6*. Перша фракція розділена дефісом представляє операції між першим вузлом і всіма наступними, друга — між другим вузлом та всіма наступними. Відповідно *1* означає операцію згортки

ядром 1×1 між першим і другим вузлом (див. рис. 3.1, доступні операції в комірці), *2* — операцію згортки ядром 3×3 між першим та другим вузлом. Цифра *6* означає операцію skip connection між другим та третім вузлом.

Параметр *ATTN* представляє номери комірок після яких потрібно застосувати операції Self-Attention.

Відповідно до описаного кодування та наведеного прикладу кодування архітектури генератора має вигляд *{3, 12-6, [1, 2]}*. Як видно із прикладу, після першої та другої комірки застосовано операцію Self-Attention.

Дискримінатор складається із одного шару згортки, чотирьох комірок, чотирьох блоків для зменшення роздільної здатності в два рази, та одного лінійного і Embedding шару.

Основним елементом в просторі пошуку дискримінатора є комірка, яка складається з *p* вузлів. На відміну від генератора, доступний набір операцій у комірці дискримінатора розширено двома операціями: максимальний пулінг ядром 3×3 та середній пулінг ядром 3×3.

Для операції зменшення розмірності застосовано операції Separable Downsample Convolution. Автори статті заявляють, що це найоптимальніший метод зменшення розмірності в згорткових мережах на даний час [19].

Архітектура комірки також залишається однаковою для всієї моделі дискримінатора. Простір пошуку дискримінатора зводиться до тих самих кроків, що і для генератора.

Простір пошуку дискримінатора закодовано таким самим чином, як і генератора: *{N, C, [ATTN]}.*

Розроблений метод складається з двох етапів: пошук архітектури генератора з фіксованим дискримінатором та пошук архітектури дискримінатора із найкращим генератором.

Схему реалізації розробленого методу наведено на рисунку 3.2.

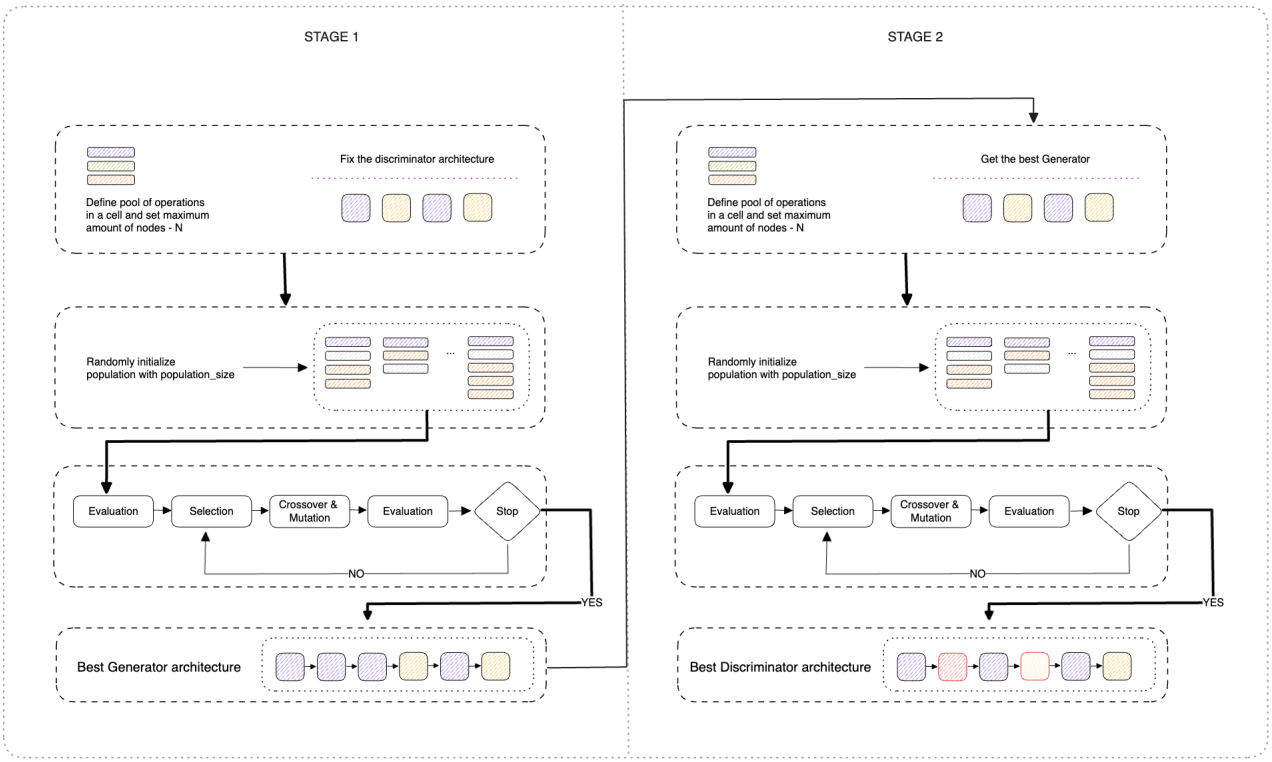


Рисунок 3.2. Схема реалізації розробленого методу

На першому етапі ми визначаємо фіксовану архітектуру дискримінатора та здійснюємо пошук тільки генератора. Архітектуру дискримінатора визначено вручну так – чотири вузли у комірці, архітектуру комірки закодовано як 235-66-7 (згортка ядром 3×3 та 5×5, zero, skip connection, skip connection, max pooling ядром 3×3), Self-Attention механізм застосовано тільки після останньої комірки.

На кожному циклі алгоритму оптимізації ми ініціалізуємо популяцію випадковими архітектурами генераторів. Далі ми створюємо пару генератора та дискримінатора для кожного генератора в популяції. Отримані пари ГЗМ мереж навчаються та оцінюються в кінці кожного циклу за допомогою FID метрики. В кінці циклу ми обираємо генератор з найнижчим значенням FID та копіюємо його вагові коефіцієнти. Далі в наступному циклі ми ініціалізуємо всі генератори скопійованими ваговими коефіцієнтами. Відповідно після першого кроку ми отримуємо архітектуру найкращого генератора, тобто моделі із найнижчим значенням FID.

На другому етапі ми використовуємо найкращу архітектуру генератора та проводимо пошук аріхтектури дискримінатора. На кожному циклі алгоритму оптимізації ми створюємо популяцію дискримінаторів та ініціалізуємо її випадковими архітектурами. Далі по аналогії з першим кроком ми створюємо пари генераторів та дискримінаторів. Отримані пари ГЗМ мереж навчаються незалежно одна від одної та оцінюються, використовуючи метрику FID. В кінці кожного циклу ми копіюємо вагові коефіцієнти найкращого дискримінатора та ініціалізуємо ними всі дискримінатори в наступному циклі. Після другого кроку ми отримуємо аріхтектуру найкращого дискримінатора, а відповідно і найкращу аріхтектуру ГЗМ.

На обох етапах до вхідних зображень моделі дискримінатора застосовано техніку диференційованих спотворень (Differentiable Augmentations) із застосуванням політики випадкового кольору та трансляції. Дана техніка допомагає додатково розширити набір навчальних даних безпосередньо під час процесу навчання та є особливо ефективною на малих наборах даних [120].

Для оптимізації архітектури ГЗМ мережі використано той самий генетичний алгоритм AGA із попереднього розділу.

3.1.3 Комп’ютерні експерименти

Для проведення експериментів використано той самий набір даних, що і в розділі 3.1.3. Реалізація програмного забезпечення базується на використанні хмарної інфраструктури GCP. Такий підхід дозволяє ефективно використовувати ресурси хмарних обчислень, оскільки не кожен має персональний комп’ютер із потужним GPU. Інфраструктуру програми зображено на рисунку 3.3.

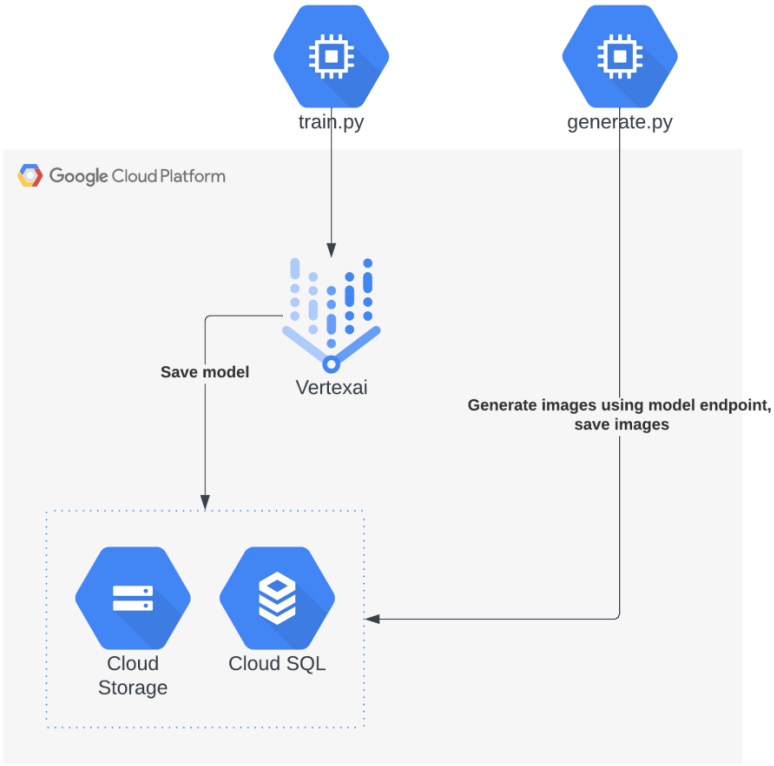


Рисунок 3.3. Хмарна інфраструктура модуля синтезу архітектур GAN

Програмний модуль складається з двох основних скриптів на мові Python: *train.py* та *generate.py*. Скрипт *train.py* призначений для побудови та навчання архітектур генеративних змагальних мереж. Для навчання моделі використано можливості сервісу Vertex AI, який надає інфраструктуру та інструменти, необхідні для ефективного навчання моделі (використано Nvidia V100 GPU).

Після того, як процес навчання завершено модель завантажується на Cloud Storage. Наступним кроком навчена модель GAN розгортається на платформі Vertex AI шляхом створення адреси URL до якої можна отримати доступ для генерації зображень. Процес розгортання передбачає задання необхідних обчислювальних ресурсів та пакування коду нашої моделі в Docker контейнер для ефективного використання.

Після розгортання моделі її основні дані зберігаються в хмарному екземплярі Cloud SQL. Ця база даних слугує центральним сховищем, де зберігаються такі дані, як URL-адреса моделі та метрики оцінки, такі як FID та

IS. Це дозволяє відстежувати версії моделі та відповідні показники ефективності.

Для генерації нових зображень застосовується файл *generate.py*. Користувачі можуть вказати кількість зображень, які вони хочуть згенерувати, як параметр під час запуску файлу. Необов'язковим параметром є ідентифікатор моделі, яку потрібно використати для синтезу зображень. За замовчування використовується остання додана в базу даних модель. Наприклад, для генерації 1000 зображень використовуючи модель із ідентифікатором 5 достатньо запустити програму викликавши *python generate.py 1000 5*. Даний скрипт взаємодіє з URL-адресою розгорнутої моделі, надсилаючи запити на генерацію зображень.

Зображення, згенеровані моделлю, отримуються та зберігаються у хмарному сховищі Cloud Storage, що забезпечує надійне та масштабоване рішення для зберігання контенту. Одночасно з цим інформація про створені зображення, включаючи часові мітки та ідентифікатор використаної моделі, записується в хмарний SQL екземпляр PostgreSQL. Модель бази даних показано на рисунку 3.4.

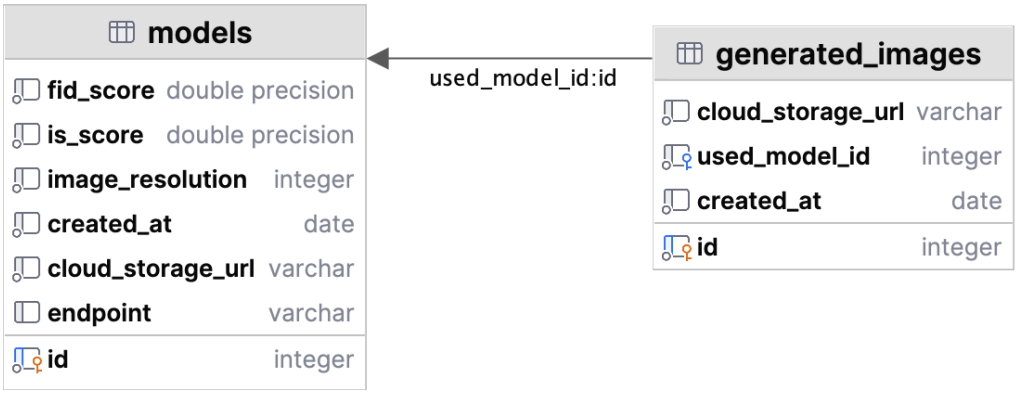


Рисунок 3.4. Модель бази даних для зберігання результатів експерименту

Для обох етапів пошуку архітектури ГЗМ використано Hinge Loss в якості функції втрат [121] та оптимізатор Adam (betas = 0.5, 0.999) [76]. Також

застосовано техніку під назвою Two Time-scale Update Rule (TTUR), яка передбачає використання різних норм навчання для генератора і дискримінатора. [59]. Відповідно норма навчання генератора 0,0001, а дискримінатора 0,0004.

Для всіх згорткових, деконволюційних та лінійних шарів в обох моделях застосовано техніку спектральної нормалізації, яка дозволяє стабілізувати навчання ГЗМ [122].

Загальна кількість циклів (*cycles*) алгоритму оптимізації на кожному етапі становить 25, а кількість епох (*epochs*) навчання для кожної моделі із популяції – 30. Розмір популяції (*population_size*) – 100. Кількість випадково обраних кандидатів для подальшого відбору батьківської архітектури для мутації (*sample_size*) – 25. Імовірність мутації (*mutation_prob*) – 0.05. Розмір батчу (*batch_size*) для генератора та дискримінатора однаковий та становить 128 зображень.

Загалом перший і другий етап зайняв 15,6 та 10,3 GPU годин відповідно. Після завершення обох етапів отримана ГЗМ мережа навчалася з нуля протягом 100 000 ітерацій. Це зайняло ще ~13,6 GPU годин. Тобто загальний час, витрачений від пошуку архітектур до отримання повністю навченої ГЗМ мережі становить 39,5 GPU годин.

Архітектуру знайденого генератора та дискримінатора показано на рисунках 3.5 і 3.6 та в таблицях 3.1 і 3.2 відповідно.

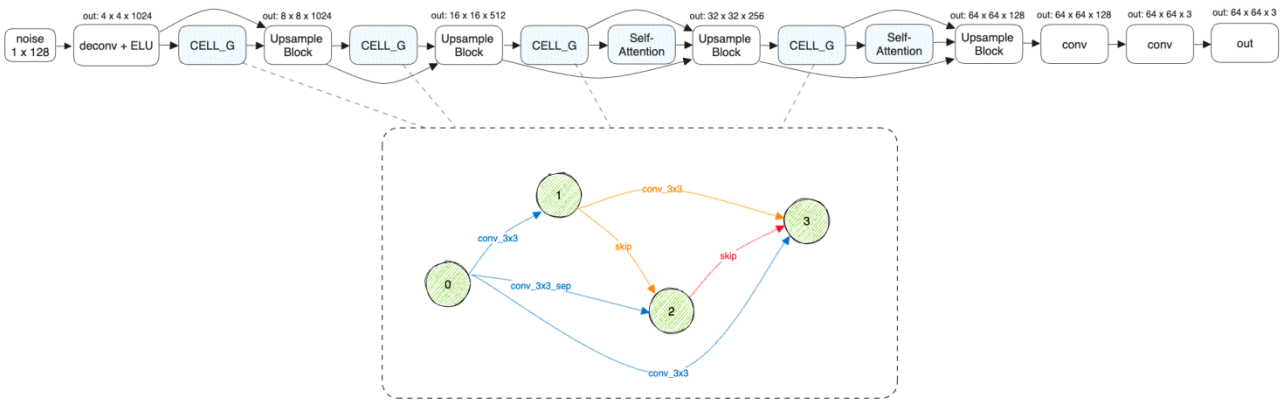


Рисунок 3.5. Архітектура генератора

Таблиця 3.1. Архітектура генератора

Шар	Параметри	Вихідна форма
L1: Input	Gaussian noise	1×128
L2: Transposed Conv + ELU activation	Kernel = 4, stride = 1, padding = 0	4×4×1024
L3: $CELL_G$	Nodes = 4	4×4×1024
L4: L2 + L3		4×4×1024
L5: Upsample	Scale = 2	8×8×1024
L6: $CELL_G$	Nodes = 4	8×8×1024
L7: L5 + L6		8×8×1024
L8: Upsample	Scale = 2	16×16×512
L9: $CELL_G$	Nodes = 4	16×16×512
L10: Self Attention	Input channels = 512	16×16×512
L11: L8 + L10 + L9		16×16×512
L11: Upsample	Scale = 2	32×32×256
L12: $CELL_G$	Nodes = 4	32×32×256
L13: Self Attention	Input channels = 256	32×32×256
L14: L11 + L13 + L12		32×32×256
L15: Upsample	Scale = 2	64×64×128
L16: Convolution	Kernel = 3, stride = 1, padding = 1	64×64×128
L17: Convolution	Kernel = 3, stride = 1, padding = 1	64×64×3
L18: Output		64×64×3

Структура комірки $CELL_G$	
L0: Input	
L1: Conv → ELU → Batch Norm	Kernel = 3, stride = 1, padding = 1
L2: L1 + Conv 3×3 → Conv 1×1 → ELU → Batch Norm	Conv 3x3 = (Kernel = 3, stride = 1, padding = 1), Conv 1x1 = (Kernel = 1, stride = 1, padding = 0)
L3: L2 + Conv (L1) + Conv (L0)	Kernel = 3, stride = 1, padding = 1
L0: Input	
L1: Conv → ELU → Batch Norm	Kernel = 3, stride = 1, padding = 1

Структура блока Upsample		
L0: Input		H × W × C
L1: Upsample	Scale = 2, mode = nearest	(H × 2) × (W × 2) × C
L2: Convolution	Kernel = 3, stride = 1, padding = 1	(H × 2) × (W × 2) × C
L3: Conditional Batch Norm	Number of classes = 4	(H × 2) × (W × 2) × C

Продовження таблиці 3.1

L4: Gated Linear Unit (GLU)	Dimension = 1	(H × 2) × (W × 2) × (C / 2)

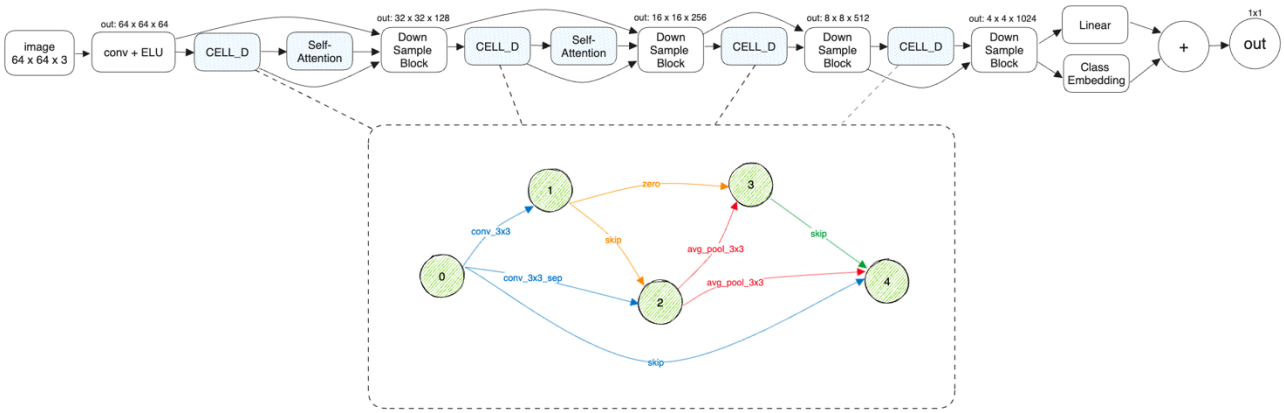


Рисунок 3.6. Архітектура дискримінатора

Таблиця 3.2. Архітектура дискримінатора

Шар	Параметри	Вихідна форма
L1: Input	Image	64×64×3
L2: Conv + ELU activation	Kernel = 3, stride = 1, padding = 1	64×64×64
L3: $CELL_D$	Nodes = 5	64×64×64
L4: Self Attention	Input channels = 64	64×64×64
L5: L2 + L4 + L3		64×64×64
L6: Downsample	Scale = 2	32×32×128
L7: $CELL_D$	Nodes = 5	32×32×128
L8: Self Attention	Input channels = 64	32×32×128
L9: L6 + L8 + L7		32×32×128
L10: Downsample	Scale = 2	16×16×256
L11: $CELL_D$	Nodes = 5	16×16×256
L12: L10 + L11		16×16×256
L13: Downsample	Scale = 2	8×8×512
L14: $CELL_D$	Nodes = 5	8×8×512
L15: L13 + L14		8×8×512
L16: Downsample	Scale = 2	4×4×1024
L17: Linear(Sum(L16))		1×1
L18: Sum(Multiply(Sum(L16), Embedding))	Number of classes = 4	1×1
L19: L17 + L18		1×1

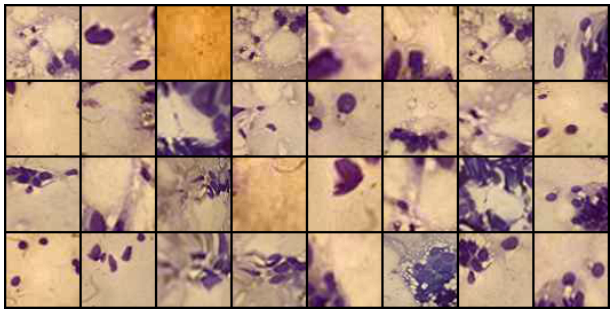
Продовження таблиці 3.2

L20: Output		1×1
Структура комірки $CELL_D$		
L0: Input		
L1: Conv → ELU → Batch Norm		Kernel = 3, stride = 1, padding = 1
L2: L1 + Conv 3×3 → Conv 1×1 → ELU → Batch Norm		(Kernel = 3, stride = 1, padding = 1), (Kernel = 1, stride = 1, padding = 0)
L3: AvgPool 3× 3 (L2)		Kernel = 3, stride = 1
L4: L0 + L3 + AvgPool 3× 3 (L2)		Kernel = 3, stride = 1
Структура блока Downsample		
L0: Input		H × W × C
L2: Convolution	Kernel = 3, stride = 1, padding = 1	H × W × (C × 2)
L3: Pixel Rearrange → Convolution	Kernel = 1, stride = 1, padding = 0	(H / 2) × (W / 2) × (C × 2)
L4: ELU		(H / 2) × (W / 2) × (C × 2)

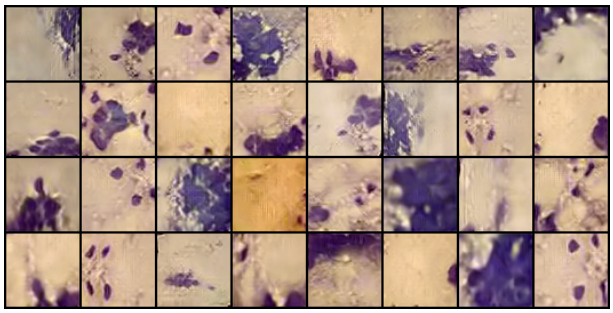
Як видно з рисунків 3.5 і 3.6, кількість вузлів у комірках генератора та дискримінатора становить 4 та 5 відповідно. В комірці генератора дві операції *skip connection*, а в комірці дискримінатора — 3. Також в комірці дискримінатора присутня операція *zero*, якої немає в генераторі. Операцію Self-Attention застосовано 2 рази як і в генераторі так і в дискримінаторі. Проте в генераторі дана операція розміщена ближче до кінця мережі, а в дискримінаторі навпаки — ближче до початку.

Значення метрики FID для синтезованої архітектури ГЗМ дорівнює 3,39, а значення метрики IS – 3,95.

Приклади порівняння синтезованих зображень із оригінальними для кожного класу наведено на рисунках 3.7–3.10. Зображення обрано випадковим чином. Порівняння з іншими архітектурами за метрикою FID наведено у таблиці 3.3.

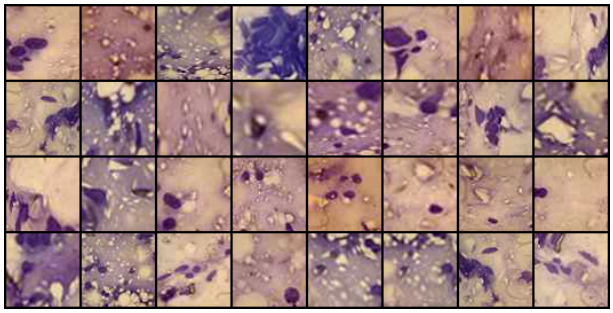


а

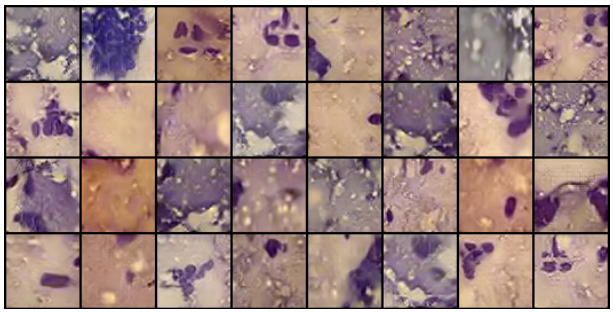


б

Рисунок 3.7. Оригінальні (а) та синтезовані (б) зображення, клас 1

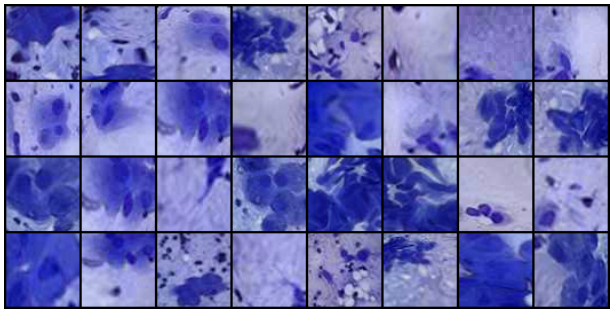


а

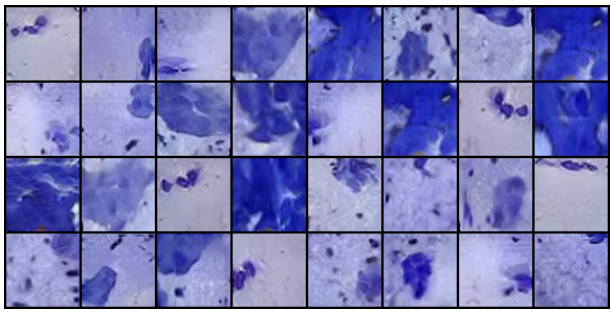


б

Рисунок 3.8. Оригінальні (а) та синтезовані (б) зображення, клас 2

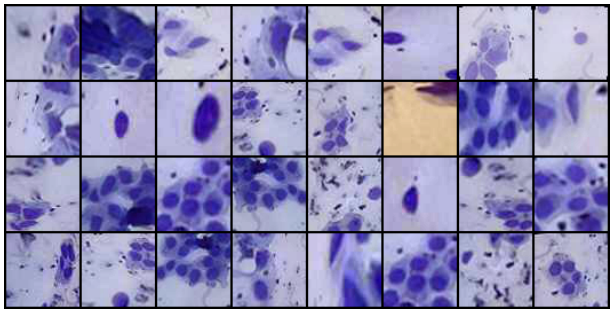


а

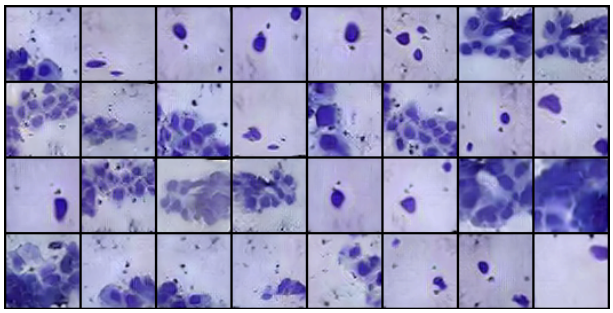


б

Рисунок 3.9. Оригінальні (а) та синтезовані (б) зображення, клас 3



а



б

Рисунок 3.10. Оригінальні (а) та синтезовані (б) зображення, клас 4

Таблиця 3.3 Результати порівняння з іншими архітектурами

Метод	FID
DCGAN	12,67
WGAN	12,72

Продовження таблиці 3.3

WGAN-GP	19,09
BGAN	10,03
BEGAN	15,32
Розроблений метод GA-GAN	3,39

На відміну від існуючих архітектур, в розробленому методі використано механізм Self-Attention в генераторі та дискримінаторі, що дозволило підвищити якість синтезованих зображень. Також розроблений метод підтримує механізм синтезу зображень за мітками (conditional generation), що не є актуальним для вищенаведених архітектур та підходів.

Рисунки 3.7–3.10 показують порівняння пар оригінальних та синтезованих зображень для кожного класу із оригінального та синтезованого датасету.

Розроблений метод не був протестований на зображеннях більшої роздільної здатності, бо це б призвело до збільшення часу пошуку. Тому обмеженням цього дослідження є відносно низька роздільна здатність синтезованих зображень – 64×64 пікселі. Для того, щоб синтезувати зображення більшої роздільної здатності потрібно збільшувати кількість комірок у генераторі та дискримінаторі.

Крім того, експерименти проведено тільки на одному підкласі біомедичних зображень – цитологічних зображенях. Відповідно подальшим напрямком дослідження може бути тестування та адаптація розробленого методу під інші класи та роздільні здатності біомедичних зображень.

3.2 Метод синтезу біомедичних зображень

Розроблений метод генерування і класифікації гістологічних зображень складається з таких кроків:

- формування початкового набору гістологічних зображень трьох класів: G1, G2, G3 на основі афінних спотворень;
- розширення вибірки гістологічних зображень на основі ГЗМ;
- проектування архітектури нейронної мережі;

- визначення точності нейронної мережі на розширеній вибірці;

Оригінальна вибірка містить зображення по класах: G1 – 9 зображень, G2 – 100 зображень, G3 – 76 зображень. Дану вибірку було розширено до 100 зображень в кожному класі шляхом застосування афінних спотворень, як і у попередніх розділах.

Приклад гістологічних зображень різних типів раку наведено на рисунку 3.11.

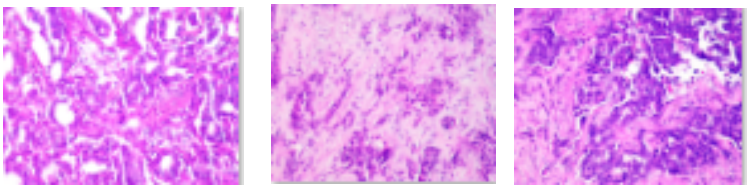


Рисунок 3.11. Приклади гістологічних зображень типу G1, G2, G3.

В основі архітектур генератора та дискримінатора лежить ResNet Block, запозичений із згорткової мережі ResNet. Генератор приймає на вхід вектор шуму із розподілу Гауса розмірністю 1×100, а на виході видає зображення 64×64×3. Архітектуру генератора можна умовно розділити на три шари, як показано на рисунку 3.12.

Перший шар є шаром попередньої обробки. В цьому шарі зчитується одновимірний вхідний вектор шуму та подається на лінійний шар для подальшої конвертації у трьохвимірний масив.

Другий шар є основним обчислювальним шаром. Він складається із 4 ResNet блоків. Після кожного такого блоку застосовано інтерполяцію nearest-neighbor для збільшення зображення в два рази. Також після третього ResNet блоку застосовано механізм Self-Attention. Усі згорткові шари використовують крок рівний 1. Також в якості функції активації застосовано ReLU.

Останній шар є вихідним шаром. Тут застосовується батч-нормалізація, активація, ще один шар згортки та фінальна функція активації Tanh.

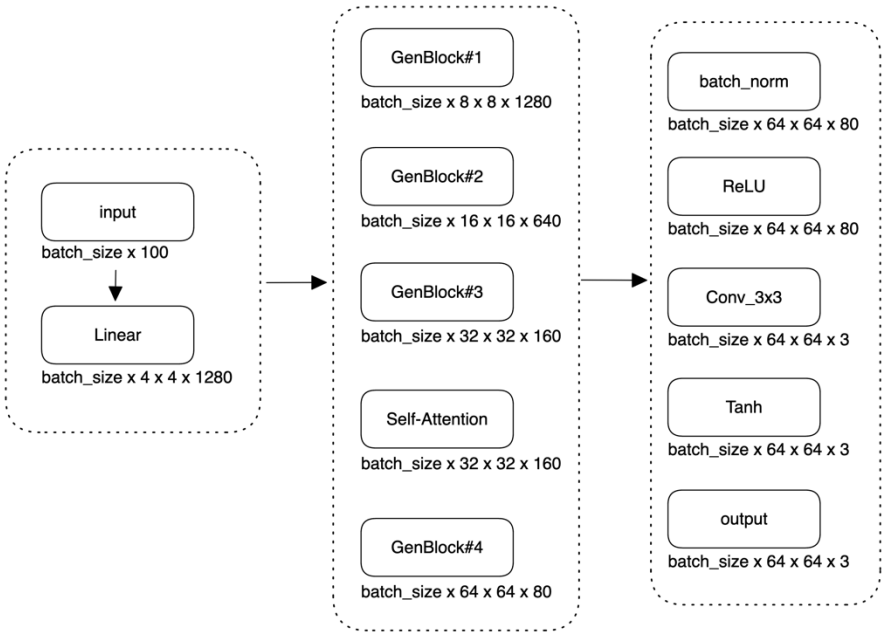


Рисунок 3.12. Архітектура генератора

Дискримінатором є згорткова нейронна мережа, яка приймає на вхід зображення розміром 64×64×3 пікселі. Дискримінатор складається із п’яти ResNet блоків. Після першого такого блоку застосовано механізм Self-Attention. Для зменшення розмірності зображення застосовується операція Average Pooling з розміром ядра 2×2 та кроком 2 в кожному із ResNet блоків. Проте в останньому блоці зменшення розмірності не застосовується. Шари згортки використовують крок 1. В якості функції активації також застосовано ReLU. Виходом дискримінатора є два лінійних шари. Перший з кількістю нейронів 1280, а другий – 1. Архітектуру дискримінатора наведено на рисунку 3.13.

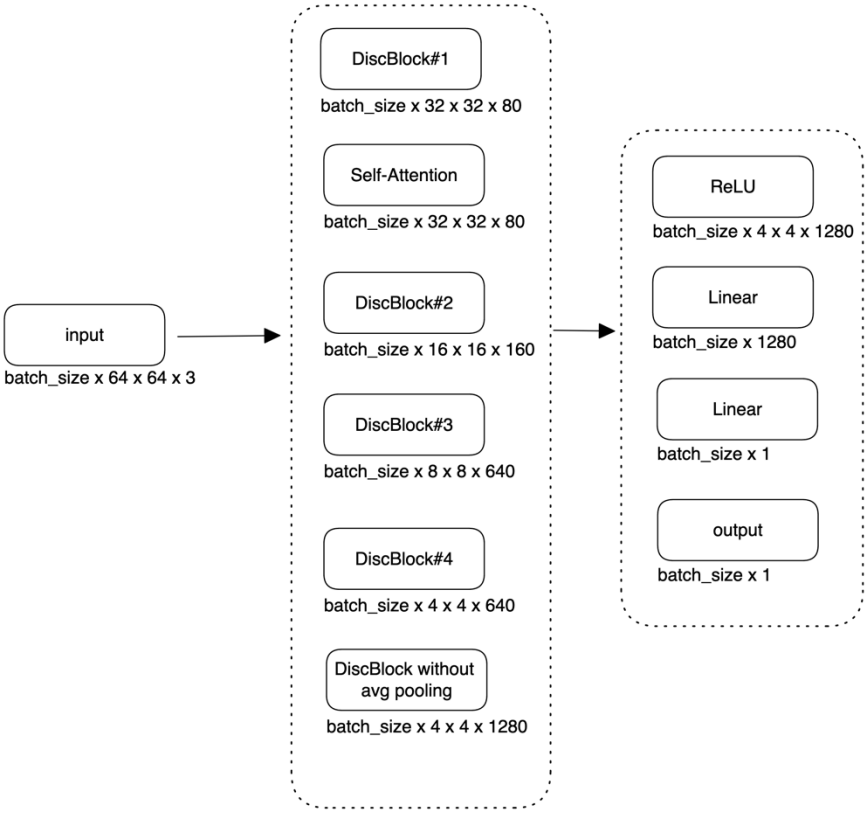


Рисунок 3.13. Архітектура дискримінатора

Архітектура розробленої ЗНМ представлена на рисунку 3.14. Архітектура ЗНМ складається із 9 згорткових шарів, 4 шарів пулінгу та одного повнозв'язного (лінійного шару).

Всі згорткові шари використовують крок 1. Для зменшення розмірності зображення використовуються шари максимального пулінгу із ядром 3×3 та кроком 2. На вхід мережі подається зображення розміром 64×64×3 пікселі. Далі зображення подається на перший згортковий шар, який використовує 64 карти ознак із ядром 3×3. Наступними двома шарами є згорткові блоки, які складаються з послідовних шарів згортки ядром 3×3, активації ReLU та батч-нормалізації. Перший блок використовує 64 карти ознак, а другий – 128. Після цих блоків застосовано шар максимального пулінгу. Відповідно тепер розмір зображення становить 32×32×128. Далі є два згорткові блоки із шаром максимального пулінгу в кінці. Проте тут шари згортки використовують ядро розміром 1×1 та однакову кількість карт ознак – 128. Розмір зображення після цих шарів дорівнює 16×16×128. Наступні два згорткові блоки ідентичні до

перших двох і також використовують однакову кількість карт ознак – 128. В кінці застосовано шар максимального пулінгу. Розмір зображення – 8×8×128. Далі є два згорткові блоки, ідентичні до попередніх, проте шар пулінгу після них вже не застосовується. Розмір зображення лишається тим самим – 8×8×128. Вихідний шар складається із послідовних шарів батч-нормалізації, шару адаптивного середнього пулінгу із кількістю вихідних вузлів 1 та лінійного шару із 3 вузлами.

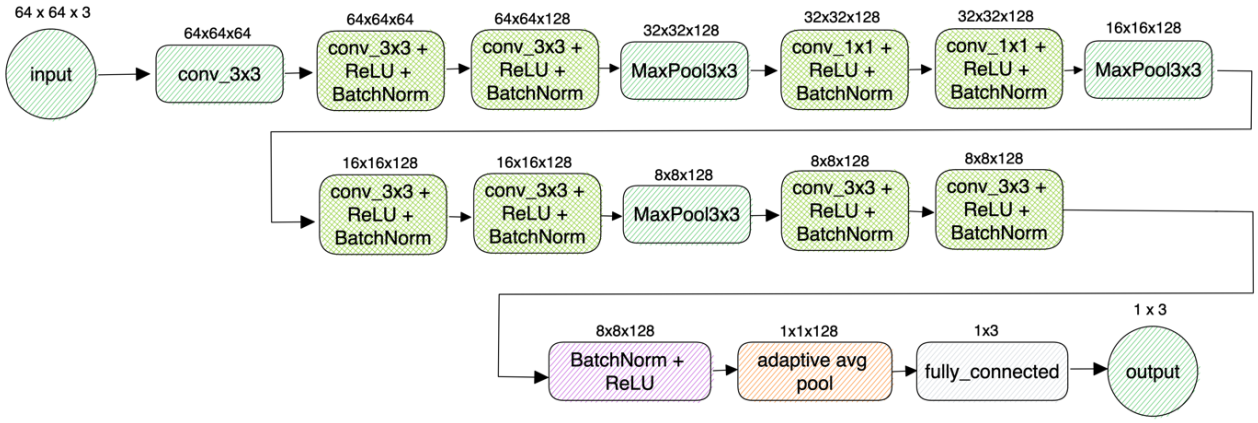


Рисунок 3.14. Архітектура розробленої ЗНМ

Формалізований опис розробленої ЗНМ такий:

$$A_{CNN} = \left\{ \left\langle I_{inp_1} \right\rangle \left\langle 64 \times 64 \times 3 \right\rangle; \left\langle c_2 \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$
$$\left\langle \left\langle c_3 \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$
$$\left\langle k_3 \right\rangle \left\langle \text{Re}\, Lu \right\rangle; \left\langle v_3 \right\rangle \left\langle Batch\ Norm \right\rangle \Big\rangle;$$
$$\left\langle \left\langle c_4 \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$
$$\left\langle k_4 \right\rangle \left\langle \text{Re}\, Lu \right\rangle; \left\langle v_4 \right\rangle \left\langle Batch\ Norm \right\rangle \Big\rangle;$$
$$\left\langle g_{\max_5} \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 2 \right\rangle;$$
$$\left\langle \left\langle c_6 \right\rangle \left\langle \text{kernel size} = 1 \times 1, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$
$$\left\langle k_6 \right\rangle \left\langle \text{Re}\, Lu \right\rangle; \left\langle v_6 \right\rangle \left\langle Batch\ Norm \right\rangle \Big\rangle;$$
$$\left\langle \left\langle c_7 \right\rangle \left\langle \text{kernel size} = 1 \times 1, \text{stride} = 1, \text{padding} = 1 \right\rangle; \right.$$
$$\left\langle k_7 \right\rangle \left\langle \text{Re}\, Lu \right\rangle; \left\langle v_7 \right\rangle \left\langle Batch\ Norm \right\rangle \Big\rangle;$$
$$\left\langle g_{\max_8} \right\rangle \left\langle \text{kernel size} = 3 \times 3, \text{stride} = 2, \text{padding} = 1 \right\rangle;$$

$\langle\langle c_9\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 1,\ \text{padding} = 1\rangle;$
$\langle k_9\rangle\langle \text{Re}\,Lu\rangle;\langle v_9\rangle\langle Batch\ Norm\rangle\rangle;$
$\langle\langle c_{10}\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 1,\ \text{padding} = 1\rangle;$
$\langle k_{10}\rangle\langle \text{Re}\,Lu\rangle;\langle v_{10}\rangle\langle Batch\ Norm\rangle\rangle;$
$\langle g_{\max_{11}}\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 2,\ \text{padding} = 1\rangle;$
$\langle\langle c_{12}\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 1,\ \text{padding} = 1\rangle;$
$\langle k_{12}\rangle\langle \text{Re}\,Lu\rangle;\langle v_{12}\rangle\langle Batch\ Norm\rangle\rangle;$
$\langle\langle c_{13}\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 1,\ \text{padding} = 1\rangle;$
$\langle k_{13}\rangle\langle \text{Re}\,Lu\rangle;\langle v_{13}\rangle\langle Batch\ Norm\rangle\rangle;$
$\langle\langle v_{14}\rangle\langle Batch\ Norm\rangle;\langle k_{14}\rangle\langle \text{Re}\,Lu\rangle\rangle;$
$\langle g_{\max_{15}}\rangle\langle \text{kernel size} = 3\times 3,\ \text{stride} = 1,\ \text{padding} = 1\rangle;$
$\langle f_{16}\rangle\langle clauses = 3\rangle\}.$

Метод синтезу штучних баз цитологічних зображень базується на використанні технологій глибокого навчання, а саме ГЗМ. Розроблений метод складається із декількох етапів. Першим етапом є завантаження навчальних зображень із директорії. Оскільки дуже часто навчальні набори даних містять досить невелику кількість зображень, то наступним кроком є розширення навчальної вибірки шляхом застосування афінних спотворень (розділ 2). Другим етапом є навчання та оцінка ГЗМ використовуючи розширену навчальну вибірку із попереднього етапу. Мережа навчається протягом заданої кількості ітерацій *Iter*. Щоб оцінити продуктивність моделі, використовуються метрики такі як FID та IS, які допомагають оцінити якість та різноманітність синтетичних зображень. Після того як задану кількість ітерацій *Iter* досягнуто, ми генеруємо N зображень та зберігаємо їх та навчену модель у базі даних. Таким чином, розроблений метод поєднує в собі завантаження даних, доповнення даних, навчання ГЗМ, оцінювання та генерування зображень. Схематичну реалізацію методу представлено на рисунку 3.15.

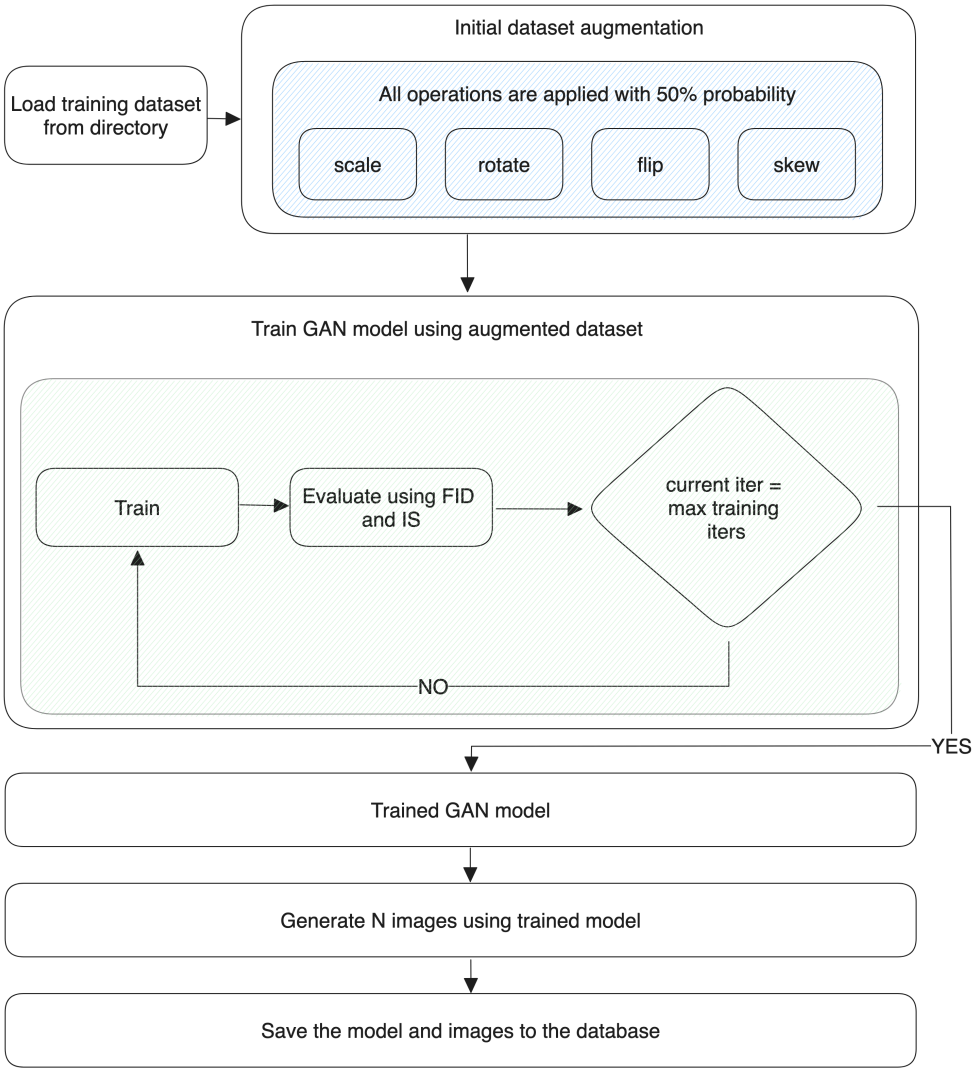


Рисунок 3.15. Схема реалізації методу синтезу біомедичних зображень

Параметри навчання ГЗМ мережі такі: оптимізатор Adam, норма навчання генератора – $1e-4$, дискримінатора – $4e-4$, кількість епох – 100000, розмір пакету – 96, функція втрат – Hinge Loss.

Для запуску експериментів використано віртуальну машину із такою конфігурацією: 16 GB RAM, 10 vCPU x 2.2 GHz, Nvidia Tesla V100 GPU 16 GB (13.2 TFLOPS). GAN мережа навчалася протягом 11 годин. В результаті експериментів значення метрик для мережі наступні: IS – 3.024, FID – 68.

Приклади синтезованих зображень приведені на рисунку 3.16

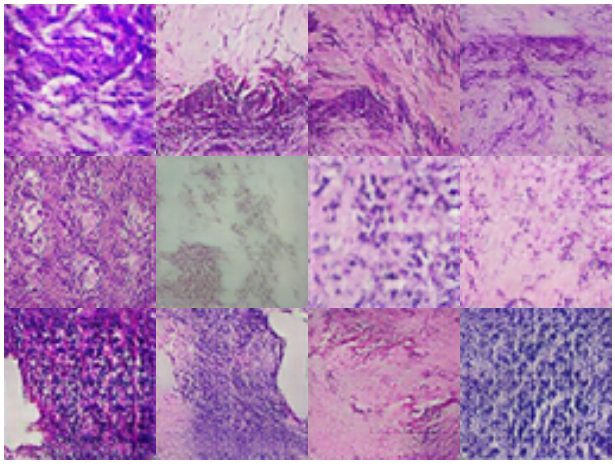


Рисунок 3.16. Приклади синтезованих зображень

Навчальну вибірку розширено шляхом застосування генеративно-змагальної мережі до 3000 зображень на клас. Загалом вибірка налічує 9000 зображень. Навчальну вибірку поділено у співвідношенні 80/20.

Для експерименту по класифікації використано наступні параметри навчання – оптимізатор Adam (1e-4), кількість епох 10, розмір пакету - 100. В результаті експерименту розроблена CNN мережа досягла точності класифікації 96%, а ROC крива показана на рисунку 3.17.

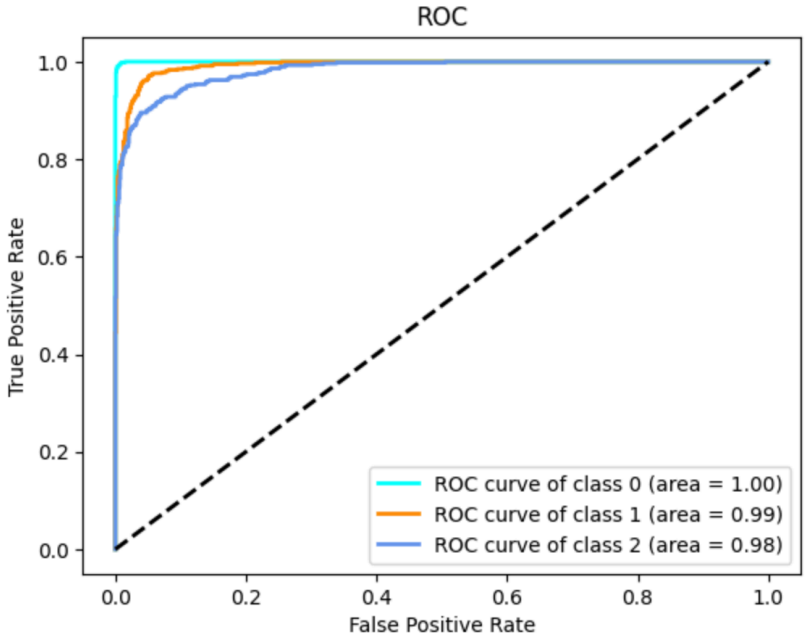


Рисунок 3.17. ROC крива розробленої CNN

3.3 Висновки до розділу 3

1. Розроблено метод синтезу архітектур ГЗМ, який складається з двох етапів: пошуку архітектури генератора із фіксованим дискримінатором, та пошуку архітектури дискримінатора із найкращим генератором.

2. В результаті експериментів синтезовано нову архітектуру ГЗМ, яка показала значення 3.39 згідно метрики FID.

3. Здійснено порівняння синтезованої архітектури ГЗМ із класичними архітектурами ГЗМ. Розроблена архітектура нейронної мережі перевершує відомі рішення по метриці FID на приблизно 10 одиниць по середніх показниках.

4. Розроблено метод синтезу біомедичних зображень, який базується на методі синтезу архітектур ЗНМ і методі синтезу архітектур ГЗМ, що дало можливість синтезувати зображення розширеної вибірки із потрібною якістю.

РОЗДІЛ 4. ПРОГРАМНИЙ ЗАСІБ СИНТЕЗУ ТА КЛАСИФІКАЦІЇ БІОМЕДИЧНИХ ЗОБРАЖЕНЬ

У розділі спроектовано архітектуру та розроблено програмний засіб для синтезу та класифікації біомедичних зображень. Спроектовано структуру організації бази даних для зберігання даних системи. Спроектовано структуру бази даних синтетичних зображень.

Результати розділу опубліковано у працях автора [7], [18].

4.1 Функціональні вимоги до програмного забезпечення

Програмний засіб складається із таких основних модулів: набори даних, класифікатори, генератори. Кожен із цих модулів має підтримує створення, отримання, редагування і видалення ресурсів. У веброзробці дані операції описано однією абревіатурою CRUD.

Модуль наборів даних призначений для додавання директорії зображень для навчання класифікаторів та генераторів. Основними вимогами до модуля є:

- створення нові набори даних, вказуючи їх назви та шляхи до директорій з зображеннями;
- імпорт зображень у різних форматах (наприклад, JPEG, PNG).
- зміна назви набору даних;
- зміна шляху директорії набору даних (заборонено при наявності пов'язаних класифікаторів чи генераторів);
- видалення цілого набору даних з системи;
- підтвердження (чи заборона) видалення для запобігання випадкової втрати даних при наявності пов'язаних класифікаторів чи генераторів.

Модуль класифікаторів призначений для класифікації зображень на основі готових або власних моделей згорткових нейронних мереж. Основними вимогами до модуля є:

- створення нових класифікаторів з вказанням їх архітектури (на основі готових моделей чи власних) та набору даних для навчання;
- зміна архітектури і набору даних класифікаторів;

- навчання класифікаторів на нових набір даних;
- задання параметрів (норма навчання, алгоритм оптимізації) перед початком навчання класифікатора;
- видалення існуючих класифікаторів з системи;
- підтвердження видалення для запобігання випадковому видаленню.

Модуль генераторів призначений для синтезу зображень на основі готових або власних архітектур ГЗМ. Основними вимогами до модуля є:

- створення нових генераторів з вказанням їх архітектури (на основі готових моделей чи власних) та набору даних для навчання;
- зміна архітектури і набору даних генераторів;
- навчання генераторів на нових наборі даних;
- задання параметрів (норма навчання, алгоритм оптимізації) перед початком навчання генератора;
- видалення існуючих генераторів з системи;
- підтвердження видалення для запобігання випадковому видаленню.

4.2 Вимоги до апаратного забезпечення

Для правильної роботи програмного забезпечення комп'ютер користувача повинен відповідати рекомендованим системним вимогам. Ключовим фактором у швидкої роботи програмного засобу є наявність графічного процесора компанії NVIDIA або комп'ютера компанії Apple із підтримкою технології Metal (наприклад, Mac Book M1 Pro). Програмний засіб або окремі його частини (модулі) можна розгортати як на локальному комп'ютері так і використовуючи провайдери хмарних технологій.

Рекомендованими системними вимогами є: комп'ютер із операційною системою MacOS/Linux/Windows; обсяг оперативної пам'яті не менше 12 гігабайт; графічний процесор (для навчання нейромереж) із обсягом пам'яті не менше 16 гігабайт. Програмний засіб може працювати на різних операційних системах, таких як Windows 10/11, macOS і Linux, наприклад, Ubuntu 20.04 LTS

або новішої версії. При використанні хмарної інфраструктури рекомендовано використання операційної системи Linux.

CPU. Для локальних комп'ютерів рекомендовано використовувати багатоядерний процесор (Intel i7 або AMD Ryzen 7 і вище), оскільки сучасні процесори з високою тактовою частотою і більшою кількістю ядер можуть значно покращити можливості обробки даних програми. При використанні хмарних технологій рекомендовано використовувати віртуальні процесори, які еквівалентні наведеним специфікаціям. Хмарні провайдери, такі як AWS, GCP або Azure, пропонують процесори, пристосовані для обчислювальних завдань, які відповідають цим вимогам.

GPU. Графічний процесор має вирішальне значення для ефективного навчання, а потім використання глибоких моделей нейронних мереж. Загалом програмний засіб має працювати на графічному процесорі з принаймні 16 ГБ відеопам'яті. Рекомендованими графічними процесорами в цій категорії можна назвати NVIDIA RTX 2080 Ti, RTX 3080, RTX 3090 або еквівалент із серії Quadro. Ці графічні процесори забезпечують необхідну обчислювальну потужність і пам'ять для обробки великих наборів даних і навчання складних моделей нейронних мереж. Для хмарної інфраструктури рекомендовано використовувати такі графічні процесори, як NVIDIA Tesla V100, A100 або аналогічні. Саме ці процесори призначені для високопродуктивних обчислень і глибокого навчання.

RAM. Важливим фактором є також достатній обсяг оперативної пам'яті. Для забезпечення безперебійної роботи програмного засобу рекомендовано щонайменше 12 ГБ оперативної пам'яті. Проте на етапі навчання нейронних мереж, коли завантаження даних і попередня обробка можуть займати досить багато пам'яті, рекомендовано використовувати 16 ГБ оперативної пам'яті. Більші обсяги оперативної пам'яті, як 32 ГБ або 64 ГБ, можуть бути корисними при роботі з більшими наборами даних (а також із зображеннями більшої роздільної здатності) і складнішими моделями.

Пристрій зберігання даних. Рекомендовано використовувати твердотільний накопичувач (SSD) з мінімум 256 ГБ пам’яті. Такі накопичувачі забезпечують більш високу швидкість доступу до даних порівняно з традиційними жорсткими дисками, що може значно скоротити час навчання та підвищити загальну продуктивність. При використанні хмарних технологій потрібно переконатися, що віртуальні сервери також використовують SSD накопичувачі.

4.3 Архітектура програмного засобу та структури бази даних

Для реалізації цього етапу дослідження використано онлайн сервіс Diagrams.Net. Архітектура програмного забезпечення базується на клієнт-серверній технології (рисунок 4.1), що в майбутньому дасть змогу масштабувати розроблену програму відповідно до потреб навантаження. На рисунку 4.2 зображено архітектуру клієнт-серверної взаємодії у системі.

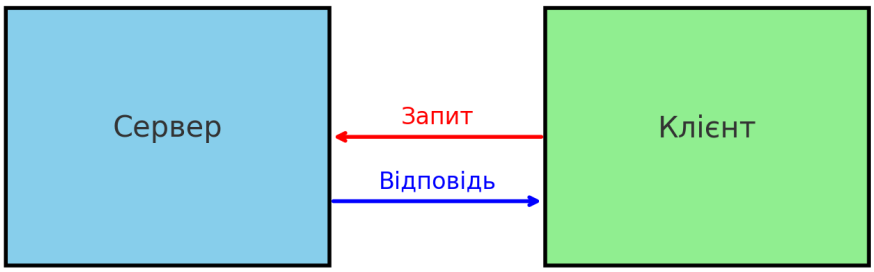


Рисунок 4.1. Клієнт-серверна взаємодія

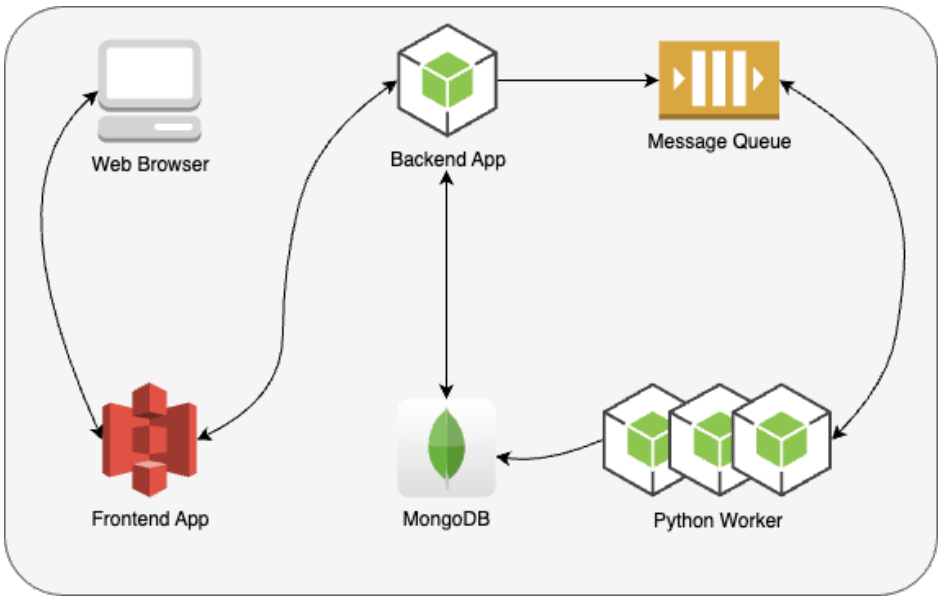


Рисунок 4.2. Клієнт-серверна архітектура системи

Архітектура системи містить такі компоненти:

- Web Browser (веб-браузер). Представляє користувача, який взаємодіє з системою через веб-інтерфейс. Веб-браузер надсилає запити до фронтенду та отримує від нього відповіді;
- Frontend App (фронтенд-додаток). Це інтерфейс, з яким взаємодіє користувач (через веб-браузер). Він приймає вхідні дані від користувача та відправляє запити до бекенду. Зазвичай це набір вебсторінок, що містять HTML, CSS та JavaScript;
- Backend App (бекенд-додаток). Це серверна частина веб-додатку, яка обробляє запити, надіслані з фронтенду, виконує логіку додатку, звертається до бази даних за необхідною інформацією, а потім повертає відповіді до фронтенду;
- MongoDB. Це система управління базами даних, яка зберігає дані програмної системи. MongoDB – це NoSQL база даних, оптимізована для великої кількості операцій читання та запису та горизонтального масштабування;
- Message Queue (черга повідомлень). Механізм для асинхронного обміну повідомленнями між різними частинами додатку. Це може допомагати у вирішенні завдань, які не потребують миттєвого

оброблення, та врівноважувати навантаження між різними воркерами або сервісами.

- Python Worker (пайтон-працівник). Це сервери або процеси, які обробляють завдання асинхронно. Вони можуть отримувати завдання з черги повідомлень та виконувати фонову обробку, таку як операції з базою даних, складні обчислення або інтеграція з зовнішніми сервісами.

Розроблена архітектура відображає сучасний веб-застосунок з чітко розділеними компонентами для фронтенду, бекенду, бази даних, оброблення черги повідомлень та фонового оброблення. Це частково демонструє використання шаблону проектування "мікросервісів", де кожен компонент може бути розміщений та масштабований незалежно. Такий підхід дасть можливість в майбутньому масштабувати систему горизонтально.

Для реалізації програмної системи необхідно спроектувати структуру бази даних типу NoSQL. Логічну модель бази даних у вигляді UML діаграми класів наведено на рисунку 4.3.

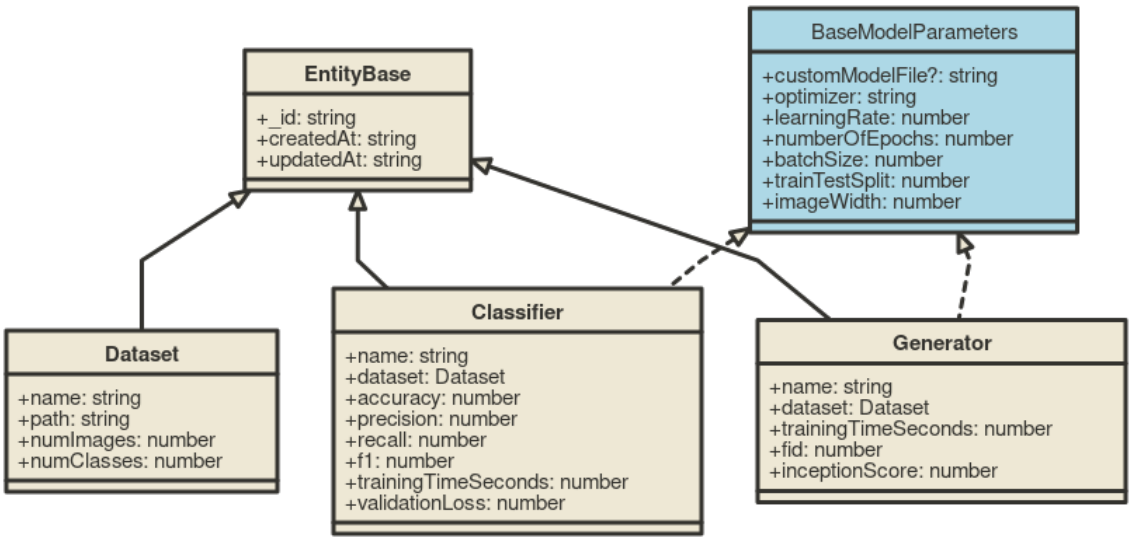


Рисунок 4.3. Логічна модель бази даних

Основними сутностями бази даних є: Dataset (набір даних) – відображає директорію із навчальними зображеннями, що поділені на класи; Classifier (класифікатор) – відображає модель нейронної мережі класифікатора, його навчальні параметри та фінальні метрики; Generator (генератор) – відображає модель нейронної мережі генератора, його навчальні параметри та фінальні метрики.

Як СУБД обрано нереляційну базу даних MongoDB, яка є досить потужним інструментом, а найголовніше дуже добре піддається горизонтальному масштабуванню. Перевагою є те, що документи зберігаються у форматі JSON, що значно спрощує взаємодію з ними із серверного додатку.

Для написання коду використано Visual Studio Code IDE. Відповідно до спроектованої архітектури програми її реалізовано окремими модулями, кожен з яких відповідає за певні завдання. Розглянемо більш детальніше кожен з них.

Frontend App. Цей додаток відповідає за взаємодію користувача із системою. Користувач має змогу створювати, редагувати та видаляти дані, як описано у функціональних вимогах. Приклади сторінок фронтенд додатку показано на рисунках 4.4–4.6.

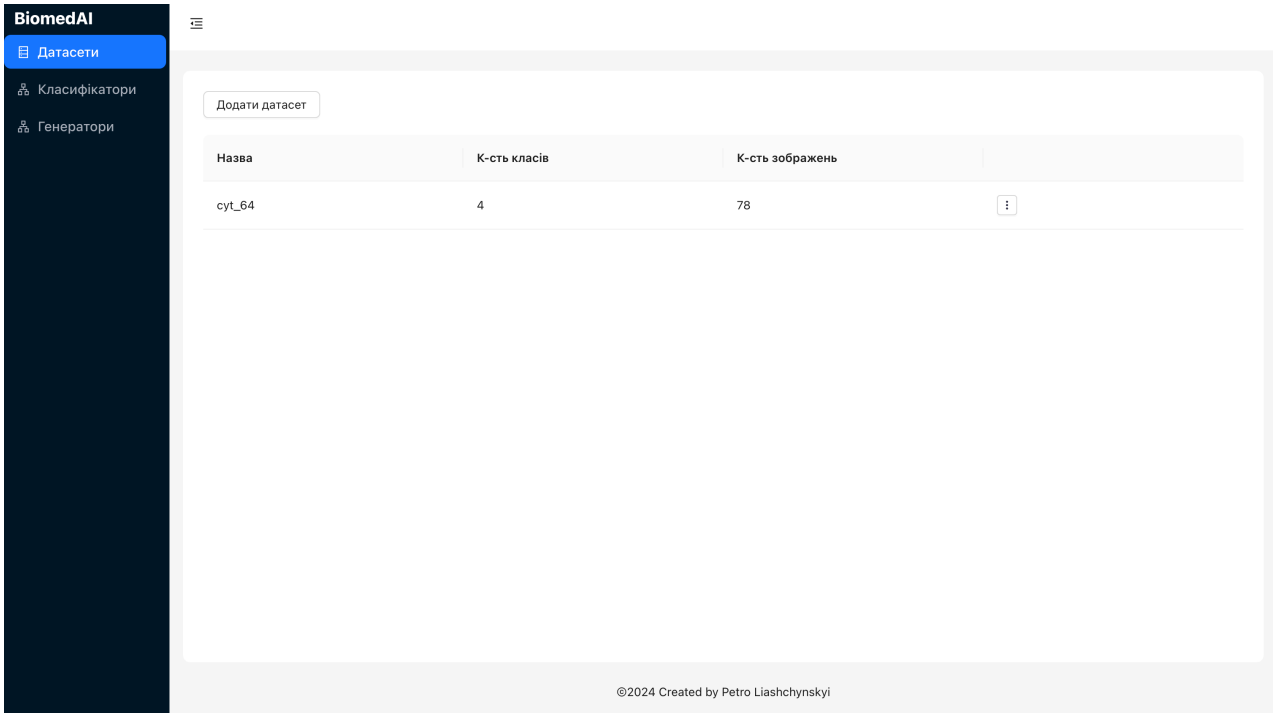


Рисунок 4.4. Сторінка модуля датасетів

Для реалізації цієї частини програми використано мову програмування TypeScript, бібліотеку React та бібліотеку готових UI-компонентів Ant Design. Загалом, цей додаток поєднує React з Ant Design для компонентів інтерфейсу, використовує React Router для навігації, Redux для управління станами, власні SVG іконоки та Axios для HTTP-запитів, дотримуючись загальної структури для сучасного розроблення веб-додатків.

Backend App. Для реалізації серверної частини додатку використано фреймворк NestJS та мову програмування TypeScript. NestJS – це прогресивна платформа, яка використовує Node.js для створення ефективних і масштабованих серверних додатків. У першу чергу платформа відома завдяки використанню TypeScript, що покращує розробку та забезпечує безпеку типів.

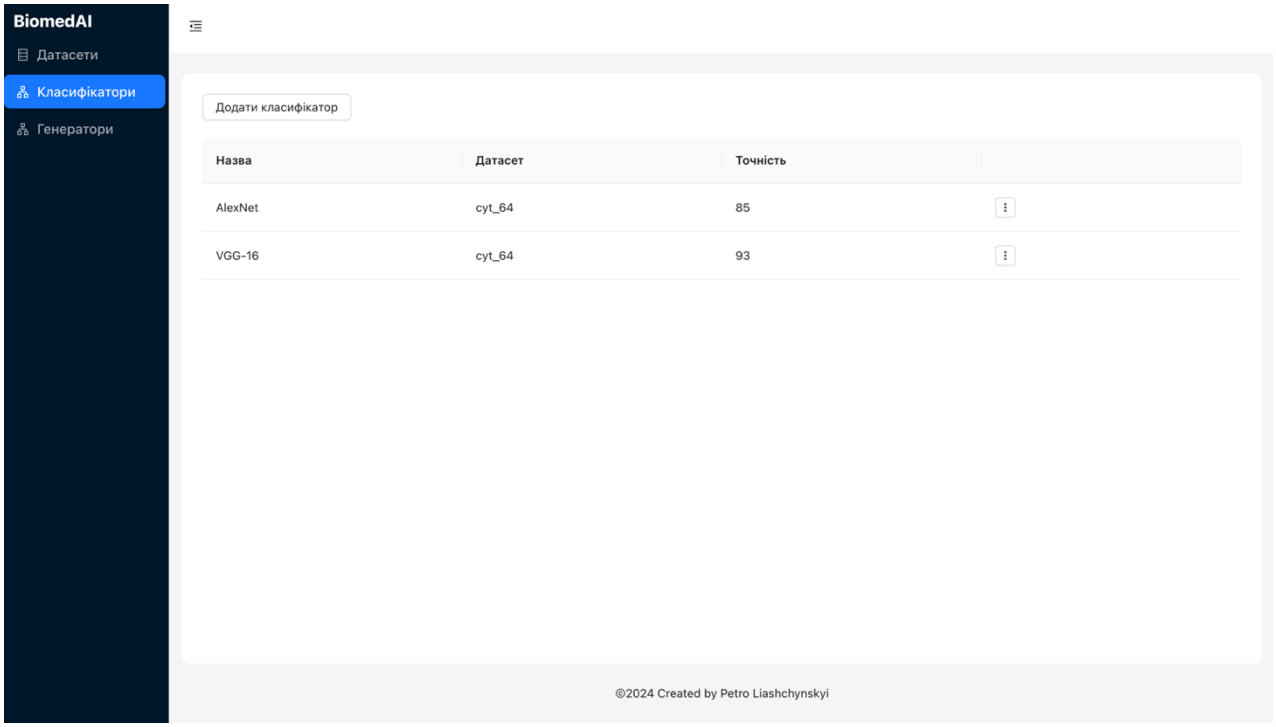


Рисунок 4.5. Сторінка модуля класифікаторів

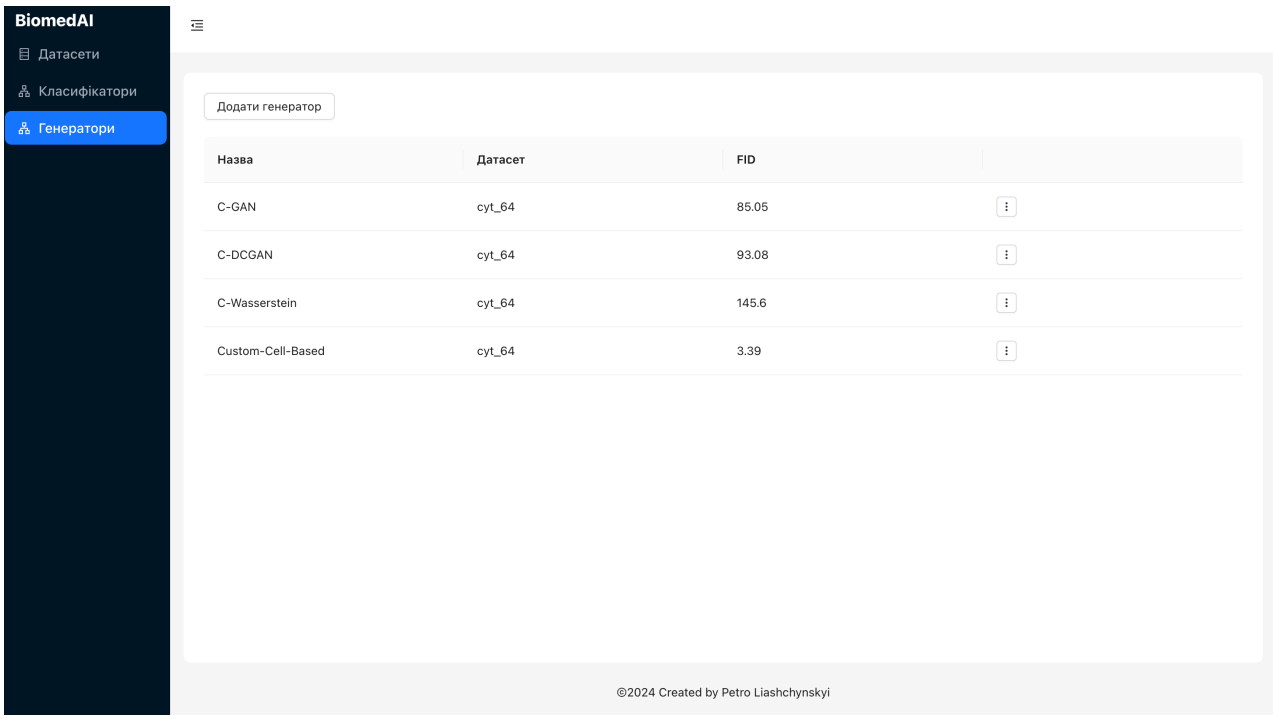


Рисунок 4.6. Сторінка модуля генераторів

NestJS також добре працює з такими середовищами розроблення, як Visual Studio Code, а додатки, написані з використанням платформи, прості в обслуговуванні та масштабуванні завдяки модульній архітектурі. Загалом платформа дає змогу використовувати рішення та шаблони проектування з інших парадигм програмування, таких як функційне програмування та об'єктно-орієнтоване програмування.

NestJS використовує модульну архітектуру, що дозволяє легко розділяти додаток на окремі логічні компоненти. Це сприяє кращій організації коду, полегшує підтримку та тестування. Інші фреймворки часто не надають такої гнучкості у структуризації додатків. Іншою перевагою NestJS є те, що він використовує вбудовану систему інверсії залежностей, що дозволяє легше керувати залежностями між компонентами додатку. Це значно полегшує процес модульного тестування та підвищує модульність коду. Проте однією з найбільших переваг NestJS є повна підтримка TypeScript. TypeScript надає статичну типізацію, яка допомагає виявляти помилки ще на етапі компіляції та покращує читабельність коду. Інші фреймворки часто вимагають додаткових налаштувань для роботи з TypeScript або не підтримують його повною мірою.

Також при розробці можна активно використовувати декоратори для оголошення класів, методів та властивостей. Це робить код більш виразним та декларативним, спрощує конфігурацію та інтеграцію з іншими компонентами. NestJS підтримує досить багато технологій, а саме HTTP, WebSocketi, GraphQL, Microservices (за допомогою різних транспортних шарів, таких як MQTT, NATS, Redis, Apache Kafka та інші), що дозволяє створювати різноманітні типи додатків без необхідності використовувати додаткові бібліотеки або фреймворки. Завдяки своїй архітектурі та ефективному використанню ресурсів, NestJS забезпечує високу продуктивність та масштабованість додатків. Це робить його чудовим вибором для створення великих проектів.

NestJS легко інтегрується з популярними бібліотеками та фреймворками, такими як Express або Fastify, що дозволяє використовувати існуючі рішення та покращує сумісність з іншими інструментами.

Наведені переваги роблять NestJS потужним та зручним інструментом для розробники сучасних, продуктивних та гнучких серверних додатків.

Отже, серверний додаток побудований на основі модульної архітектури. Відповідно до функціональних вимог в системі є три модулі (набори даних, класифікатори, генератори), тому кожен модуль винесений в окрему директорію. Структуру директорій зображено на рисунку 4.7.

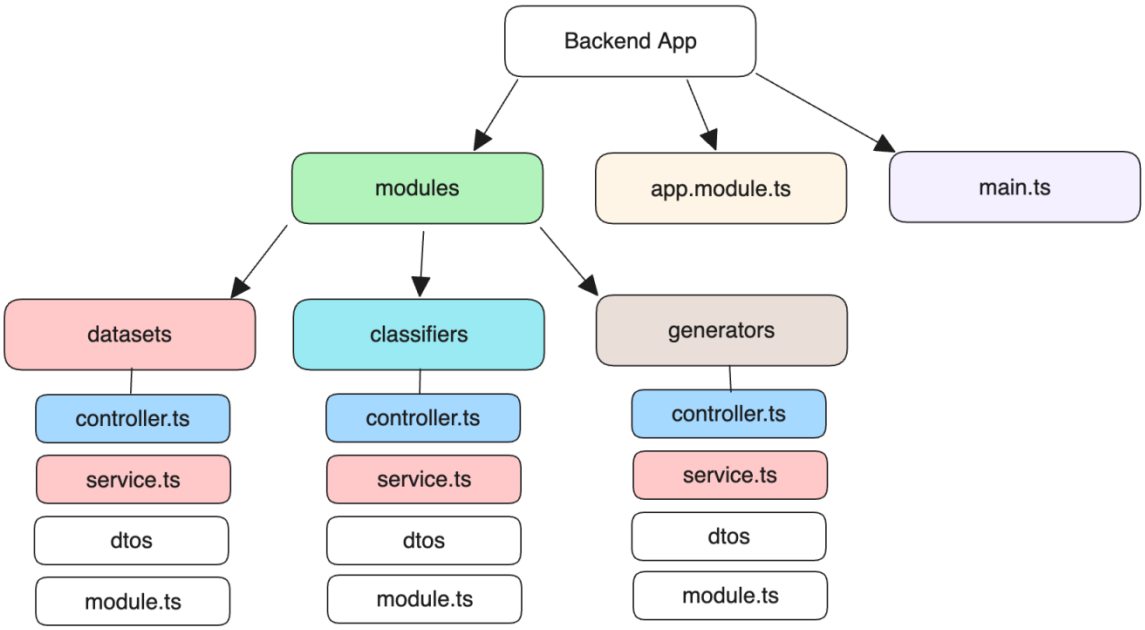
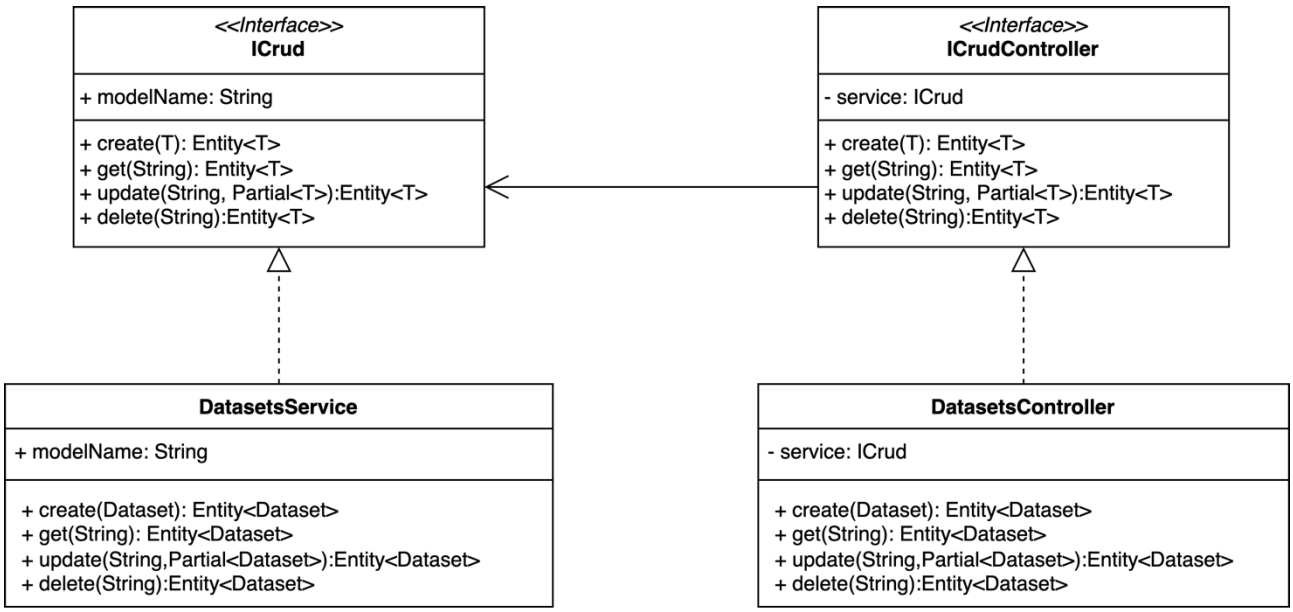


Рисунок 4.7. Структура директорій додатку

Вхідною точкою додатку є файл *main.ts*, де описано логіку запуску веб-сервера та основні його конфігурації. В цьому файлі створюється головний модуль (AppModule) та запускається веб-сервер. У файлі *app.module.ts* описано головний модуль, який імпортує всі інші модулі системи.

Далі йде директорія із всіма додатковими модулями. Наприклад, розглянемо модуль наборів даних. У папці *dtos* знаходяться класи Data Transfer Object, які описують в якому форматі веб-сервер може приймати дані, коли до нього звертаються за певними URL адресами. Файл *controller.ts* описує клас HTTP роутера, який маршрутизує запити з конкретних URL адрес веб-сервера на певні обробники, які якраз описані в файлі *service.ts*. Файл *module.ts* є вхідною точкою будь-якого модуля. В ньому імпортуються та експортуються всі залежності. Саме цей файл підключається до головного модуля програми AppModule. На рисунках 4.8–4.10 наведено UML діаграми класів для кожного з модулів.



Рисунку 4.8. Узагальнена UML діаграма класів модуля Datasets

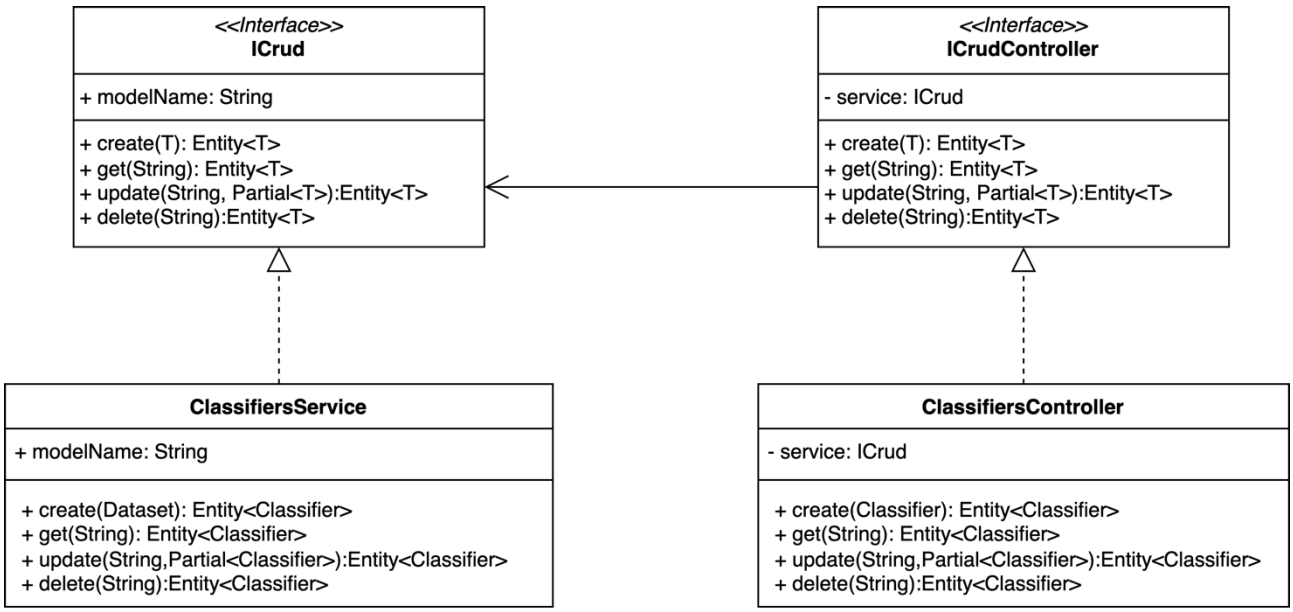


Рисунок 4.9. Узагальнена UML діаграма класів модуля Classifiers

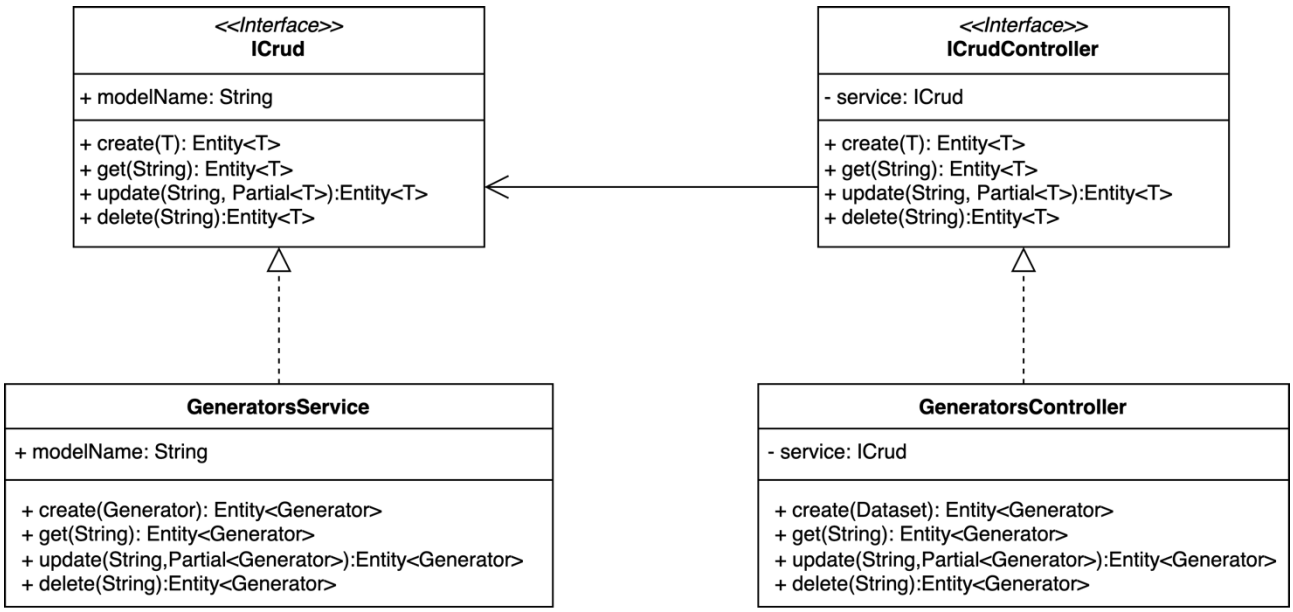


Рисунок 4.10. Узагальнена UML діаграма класів модуля Generators

Важливою перевагою використання NestJS є також те, що користувач практично з мінімальними затратами може налаштувати генерування OpenAPI документації для описаних контролерів. OpenAPI — це стандартний фреймворк для створення REST API. Цей стандарт дає можливість розробникам визначати та документувати структуру та поведінку API у зрозумілому форматі. Документація OpenAPI зазвичай створюється використовуючи такі формати, як YAML або JSON і описує різні елементи API, включаючи кінцеві точки (endpoints), параметри запитів, формати відповідей та методи автентифікації.

Специфікація визначає доступні кінцеві точки API, вказуючи методи (такі як GET, POST, PUT, DELETE), які можуть бути використані на кожній кінцевій точці, разом з їх параметрами та очікуваними відповідями. Це дозволяє визначити структуру запитів і відповідей, включаючи заголовки, параметри запитів, тіла запитів і тіла відповідей. Це забезпечує узгодженість і зрозумілість того, як відбувається обмін даними з API. OpenAPI містить інформацію про методи автентифікації, такі як ключі API та OAuth2, а також про те, як вони повинні бути реалізовані, щоб забезпечити безпечний доступ до даних. Однією з основних переваг OpenAPI є його здатність генерувати інтерактивну

документацію. Такі інструменти, як Swagger UI і Redoc, можуть читати документи OpenAPI і надають зручні інтерфейси для вивчення і тестування API.

Загалом використовуючи OpenAPI, розробники можуть створювати добре задокументовані, узгоджені та легкі у використанні API, які полегшують інтеграцію та взаємодію між різними системами та командами.

Для прикладу, документацію API для модуля наборів даних показано на рисунку 4.11.

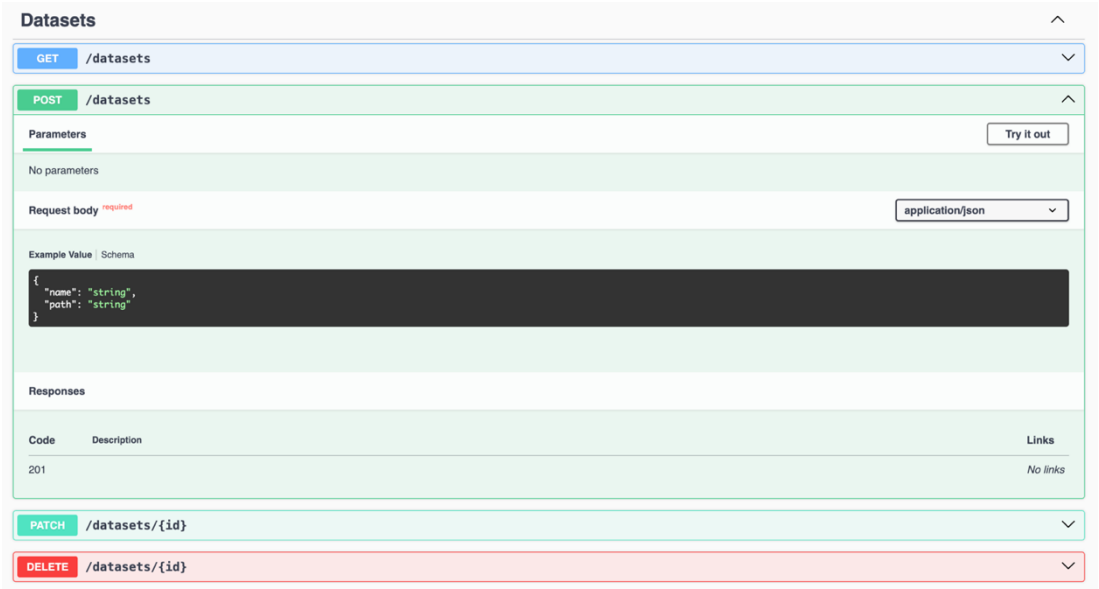


Рисунок 4.11. Приклад OpenAPI документації для модуля наборів даних

Message Queue. Черга повідомлень, також відома як просто queue, є функцією програми, яка дає змогу обмінюватися повідомленнями між різними її частинами. Її можна порівняти з поштовою скринькою, куди надсилаються повідомлення. Після цього отримувач може взяти їх, коли йому зручно. Основними концептами черги повідомлень є:

- асинхронний зв'язок: означає, що людина, яка відправляє повідомлення, не зобов'язана чекати, доки воно буде отримано одержувачем. Це збільшує масштабованість і ефективність системи;

- надійність: черга повідомлень гарантує безпечну доставку. Повідомлення можна повторно спробувати обробити пізніше, навіть якщо виникає збій під час оброблення;
- взаємодія незалежних програм: дає змогу різним частинам програми спілкуватися між собою, незважаючи на те, що вони написані на різних мовах або працюють на різних комп'ютерах.

Загалом, черга повідомлень є потужним інструментом, який може допомогти створити програми, які більш масштабовані та гнучкі.

В розробленому програмному засобі чергу повідомлень застосовано у модулях класифікаторів та генераторів в серверному додатку. Процес навчання моделі нейронної мережі часто є довготривалим. Для того щоб користувач не чекав поки він завершиться, ми додаємо повідомлення в чергу, а користувачу показуємо сповіщення, що процес навчання класифікатора або генератора розпочато. Даний підхід дає змогу не блокувати взаємодію користувача із додатком, тому користувач може виконувати інші завдання паралельно (наприклад, додавати чи редагувати набори даних). Для імплементації черги повідомлень в нашій програмі ми використали платформу RabbitMQ. В основі даної технології лежить концепція продюсерів і споживачів. Продюсер публікує повідомлення в чергу, а споживач його отримує та обробляє. Проста взаємодія двох сервісів використовуючи чергу повідомлень показано на рисунку 4.12.

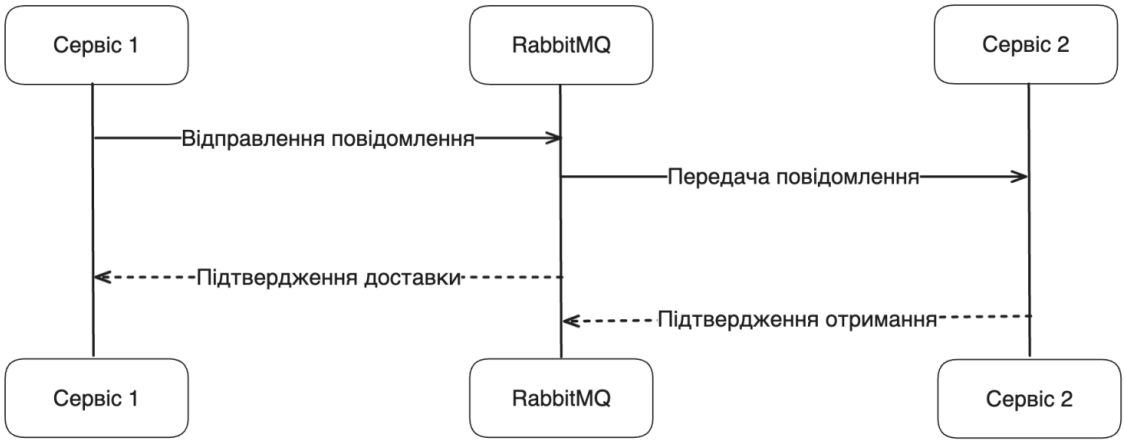


Рисунок 4.12. Взаємодія двох сервісів використовуючи RabbitMQ

Python Worker. Ці сервіси фактично є отримувачами (споживачами) повідомлень із RabbitMQ, а відповідно один отримувач відповідає за навчання моделі нейронної мережі. Серверний додаток є продюсером, який публікує повідомлення про потребу запуску навчання мережі в чергу. Після того як навчання завершене споживач створює нотифікацію в базі даних, після чого користувач бачить відповідне повідомлення в системі. Для імплементації споживачів використано мову програмування Python та фреймворк машинного навчання PyTorch (для створення та навчання моделей нейронних мереж).

Оскільки навчання мережі потребує досить багато системних ресурсів, то споживачі повідомлень також дуже добре піддаються масштабуванню. Наприклад, вони можуть бути винесені на окремі потужні комп'ютери, щоб пришвидшити навчання моделей.

Отже, програмний засіб реалізовано із використанням клієнт-серверної технології використовуючи такі мови програмування як TypeScript та Python. Програмний засіб складається із клієнтської (реалізовано з використанням бібліотеки React) та серверної (реалізовано використовуючи фреймворк NestJS) частин, а також із обробника повідомлень на базі RabbitMQ та Python. Функціональними особливостями системи є можливість класифікації та синтезу біомедичних зображень. Систему реалізовано із використанням розподіленої архітектури, що задовольняє таку нефункціональну вимогу як гнучкість та масштабованість.

Оскільки архітектура системи побудована з використанням клієнт-серверної технології, то кожен з описаних сервісів може бути розгорнутий як і локально, так і використовуючи технології хмарних обчислень. Навчання нейронних мереж часто потребує потужних графічних процесорів, які не кожен користувач має в наявності, тому можливість перенесення цих обчислень на хмару є перевагою розробленої системи. Проте варто відзначити, що при використанні системи для синтезу зображень використовуючи складні моделі нейронних мереж чи велику роздільну здатність зображень необхідно буде винести модуль *Python Worker* на окрему хмарну інфраструктуру, що може

призвести до високих грошових витрат. В основному витрати залежать від складності моделі та часу її навчання.

У сучасній практиці розробки програмного забезпечення процеси ручного розгортання все частіше розглядаються як неефективні та схильні до помилок. Знання, необхідні для ручного розгортання програми, охоплюють кілька областей, включаючи конфігурацію сервера, управління мережею, протоколи безпеки та конвеєри безперервної інтеграції/безперервного розгортання (CI/CD). Кожна з цих областей вимагає значних знань і уваги до деталей, що є суттєвою перешкодою для ефективної розробки та розгортання програмного забезпечення.

Ручне розгортання часто передбачає низку складних кроків: компіляцію програмного забезпечення, перенесення збірки на сервер, налаштування серверного оточення, управління залежностями і, нарешті, запуск програми. Цей процес необхідно повторювати щоразу, коли програмне забезпечення оновлюється або модифікується, що може забирати багато часу і бути схильним до людських помилок. Такі повторювані завдання можуть призвести до неузгодженості між середовищами розробки, що потенційно може спричинити непередбачувані проблеми, які важко діагностувати та виправити.

Щоб пом'якшити ці проблеми, автоматизація процесу розгортання стала основою сучасних методологій розробки, таких як DevOps. Автоматизація використовує інструменти та скрипти для виконання повторюваних завдань, мінімізуючи людську взаємодію з процесом розгортання та зменшуючи ймовірність помилок. Основними перевагами автоматизованого розгортання є узгодженість, швидкість, надійність і масштабованість. Автоматизоване розгортання гарантує, що кожного разу виконуються однакові процедури, а процес розгортання є ідентичним, що зменшує розбіжності між середовищами та підвищує надійність процесу розгортання. Автоматизація також значно прискорює розгортання, дозволяючи частіше оновлювати і швидше ітерації продуктів. Це особливо вигідно на конкурентному ринку, де здатність швидко

адаптувати та вдосконалювати програмне забезпечення може бути значною перевагою.

Такі інструменти, як Docker і Terraform, є прикладом підходу до автоматизації. Контейнери Docker забезпечують узгоджене середовище для додатків, абстрагуючись від базової інфраструктури і гарантуючи, що програмне забезпечення працює однаково в будь-якому середовищі, від ноутбука розробника до виробничого сервера. Terraform, з іншого боку, автоматизує надання інфраструктури, дозволяючи командам визначати свою інфраструктуру у вигляді коду. Це не лише пришвидшує налаштування середовища, але й покращує відтворюваність та ремонтопридатність конфігурацій інфраструктури.

Запуск серверної частини розробленого програмного засобу на локальному комп’ютері вимагає від користувача встановлення системи управління базами даних MongoDB, інтерпретатора Python, середовища NodeJS та інших бібліотек. Цього кроку можна уникнути використовуючи єдину програму Docker. Для цього потрібно створити Docker файли для кожного окремого сервісу, а потім зібрати всі контейнери в одному Docker Compose файлі.

Як вже було зазначено вище, відповідно до архітектури розробленого програмного засобу його можна розгорнути на хмарній інфраструктурі, використовуючи такі сервіси CI/CD, як GitHub Actions чи Gitlab Pipelines. Подальша інтеграція створених Docker файлів у конвеєри CI/CD покращує процес автоматизації. Конвеєри CI/CD автоматизують етапи від подання коду до розгортання, що дозволяє безперервно інтегрувати нові зміни коду і полегшує безперервне розгортання у виробничих середовищах. Ці конвеєри використовують автоматизовані тести, щоб гарантувати, що нові зміни відповідають стандартам якості перед об'єднанням і розгортанням. Такий рівень автоматизації не лише прискорює цикли розробки та розгортання, але й значно зменшує ймовірність внесення помилок. Приклад файлів Docker для серверної частини та Python Worker наведено на рисунках 4.13 і 4.14.

```
FROM node:18.18.0-alpine as development
LABEL stage=builder
WORKDIR /app
COPY package*.json ./
RUN apk update && apk upgrade && apk add git && npm ci
COPY . .

FROM node:18.18.0-alpine as builder
LABEL stage=builder
WORKDIR /app
RUN apk update && apk upgrade && apk add git
COPY --from=development /app ./
RUN npm run build && rm -rf node_modules && npm install --only=production

FROM node:18.18.0-alpine
WORKDIR /app

COPY --chown=node:node --from=builder /app/dist ./dist
COPY --chown=node:node --from=builder /app/node_modules ./node_modules
COPY --chown=node:node --from=builder /app/package.json ./
COPY --chown=node:node --from=builder /app/.env ./

USER node
EXPOSE 3000

ENV TZ UTC
CMD node dist/main.js
```

Рисунок 4.13. Docker файл серверної частини додатку

```
FROM python:3.10-slim

WORKDIR /app
COPY . /app

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 5000
CMD ["python", "worker.py"]
```

Рисунок. 4.14. Docker файл Python Worker

Для розгортання серверної частини додатку можна використати сервіс AWS Fargate, який надає можливість динамічно розгортати Docker контейнери ефективно керуючи обчислювальними ресурсами. Для того, щоб розгорнути потрібну інфраструктуру на AWS можна використати технологію Terraform. Приклади файлів для розгортання програми на хмарній інфраструктурі наведено у додатку Д.

4.4 База даних синтетичних зображень

База даних штучних цифрових цитологічних зображень раку молочної залози містить зображення синтезовані за допомогою ГЗМ. Вона включає зображення, що відображають різні типи раку, забезпечуючи цінний ресурс для підготовки та оцінки алгоритмів комп'ютерного зору, призначених для автоматизації процесу діагностики раку молочної залози. Завдяки використанню штучних зображень можна зменшити залежність від реальних біологічних зразків, що є особливо важливим у ситуаціях з обмеженим доступом до таких зразків або при необхідності забезпечення конфіденційності медичних даних.

Призначення бази даних: навчання автоматичних систем діагностування раку молочної залози.

Цифрові цитологічні зображення згенеровано за допомогою ГЗМ. Нейронна мережа була навчена на основі реальних цитологічних зображень чотирьох типів раку молочної залози: кістозна мастопатія, мастопатія, непроліферативна фіброзна мастопатія, непроліферативна кістозна мастопатія.

База даних в складається із набору 4000 файлів штучних цитологічних зображень, MySQL dump файлу із повним вмістом бази даних налаштувань використаних при синтезі штучних зображень, CSV файлу із 100 першими записами налаштувань ГЗМ для кожного окремого зображення. Файли зображень розміром 64×64 у форматі PNG згруповані по 4 каталогах 4-х типів раку молочної залози:

- cyt_kist_mast – кістозна мастопатія,

- cyt_mast – мастопатія,
- cyt_nep_fib_mast – непроліферативна фіброзна мастопатія,
- cyt_nep_mast – непроліферативна кістозна мастопатія.

В кожному каталозі також вміщено файл контрольної суми за алгоритмом MD5 для файлу кожного зображення. Обсяг набору зображень 19 Мб.

База даних штучних цифрових цитологічних зображень раку молочної залози «BCADCID» призначена для зберігання та організації даних, пов'язаних зі створенням синтетичних зображень з допомогою ГЗМ. Вона дозволяє зберігати деталі архітектур генераторів та дискримінаторів, які використовуються для створення і тестування моделей ГЗМ. База даних містить параметри навчання, які важливі для налаштування мережі та отримання оптимальних результатів. Також вона включає інформацію про класи, що можуть бути корисні для покращення точності класифікації згенерованих зображень. База даних забезпечує зв'язок між зображеннями та їх класами, спрощуючи доступ та аналіз згенерованих даних. Даталогічну модель бази даних показано на рисунку 4.15. Ключові поля на схемі позначено у вигляді ключа.

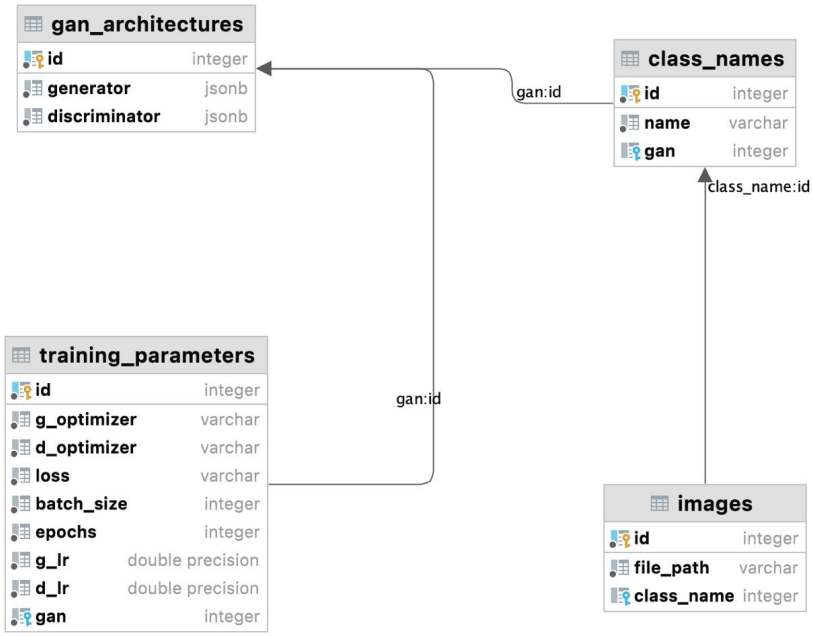


Рис. 4.15. Даталогічна модель бази даних

Для збереження даних застосовані такі типи:

- ціле число integer;
- стрічка змінної довжини varchar;
- дійсне число double precision;
- дані JSON у двійковому представленні jsonb.

Таблиця gan_architectures (таблиця 4.1) містить інформацію про різні архітектури ГЗМ. Вона зберігає структури генератора та дискримінатора, що є ключовими компонентами GAN, у форматі JSONB.

Таблиця 4.1. «ГЗМ архітектури» gan_architectures

Поле	Тип даних	Опис
Id	integer	Унікальний ідентифікатор архітектури GAN
generator	jsonb	JSONB об'єкт, що містить дані про генератор у мережі GAN.
discriminator	jsonb	JSONB об'єкт, що містить дані про дискримінатор у мережі GAN

Таблиця class_names (таблиця 4.2) призначена для зберігання назв класів зображень датасету, на якому вчилася ГЗМ і відповідно зображення цих класів вона може генерувати. Зовнішній ключ на таблицю gan дозволяє отримати всі класи на яких вчилася мережа.

Таблиця 4.2. «Пацієнти» class_names

Поле	Тип даних	Опис
Id	integer	Унікальний ідентифікатор класу.
Name	varchar	Назва класу.
Gan	integer	Зовнішній ключ, що вказує на ідентифікатор GAN у таблиці gan_architectures

Таблиця training_parameters (таблиця 4.3) використовується для зберігання параметрів навчання, які контролюють процес тренування моделі ГЗМ. Вона включає оптимізатори, функцію втрат та інші налаштування, які впливають на ефективність і стабільність навчання.

Таблиця 4.3. «Параметри навчання» traning_parameters

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор параметрів навчання.
g_optimizer	varchar	Оптимізатор, що використовується для генератора.
d_optimizer	varchar	Оптимізатор, що використовується для дискримінатора.
Loss	varchar	Функція втрат, що використовується при навчанні.
batch_size	integer	Розмір пакету даних для одного кроку навчання.
epochs	integer	Кількість епох навчання.
g_lr	double precision	це норма навчання (learning rate) для генератора.
d_lr	double precision	це норма навчання (learning rate) для дискримінатора.
Gan	integer	Зовнішній ключ, що вказує на ідентифікатор GAN у таблиці gan_architectures.

Таблиця images (таблиця 4.4) служить для зв'язування згенерованих зображень з їх відповідними класами та шляхами файлів. Вона дозволяє легко відстежувати та отримувати доступ до зображень, що були створені моделлю ГЗМ.

Таблиця 4.4. «Зображення» images

Поле	Тип даних	Опис
Id	integer	Унікальний ідентифікатор зображення.
file_path	varchar	Шлях до файлу зображення.
class_name	integer	Зовнішній ключ, що вказує на ідентифікатор класу у таблиці class_names.

4.5 Комп’ютерні експерименти

Розглянемо приклад використання готового програмного засобу. Експерименти проведено на ноутбуці MacBook Pro із процесором Apple M1 Pro та обсягом оперативної пам’яті 16 Гб.

Для проведення експерименту по синтезу цитологічних зображень обрано вже навчену синтезовану архітектуру ГЗМ із розділу 3 із значенням FID 3.39. В

результаті синтезовано по 4000 зображень роздільною здатністю 64×64 пікселі для кожного класу із навчального набору даних. Відповідно загальна кількість синтетичних зображень становить 16 000 (рисунок 4.16).

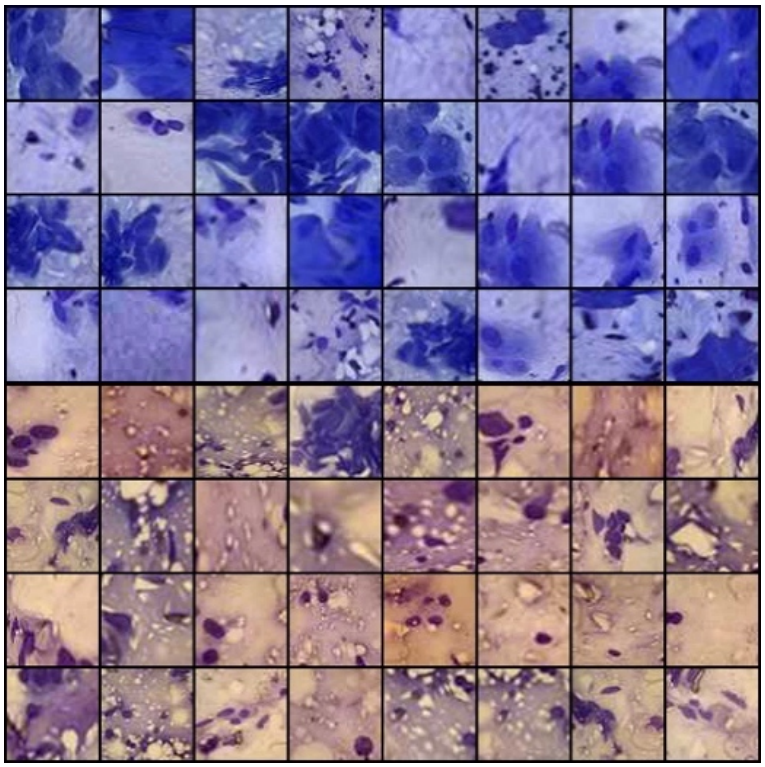


Рисунок 4.16. Приклади синтетичних зображень

Для кожного класу оригінальних і синтетичних зображень проведено порівняння за гістограмами. Порівняння гістограм оригінальних і синтетичних зображень, створених за допомогою ГЗМ, має ряд переваг. Для того, щоб визначити чи мають синтетичні зображення подібний розподіл кольорів до оригінальних зображень, можна провести аналіз гістограми, який показує розподіл кольорів або інтенсивностей пікселів на зображенні. Крім того, за допомогою цього порівняння можна виявити режим колапсу — поширену проблему при навчанні ГЗМ. Відмінності в розподілі значень пікселів між оригінальними та синтетичними зображеннями можна підкреслити якраз за допомогою гістограм. Вони дають кількісну оцінку за допомогою статистичних показників, таких як розбіжність дивергенції Кульбака-Лейблера (KL). Гістограми є простим візуальним інструментом для вивчення загальних

характеристик і якості синтетичних зображень, за допомогою якого можна визначити рад проблем, як нерівномірний розподіл кольорів або недостатня деталізація. Результати порівняння гістограм оригінальних і синтетичних зображень наведено на рисунках 4.17–4.20.

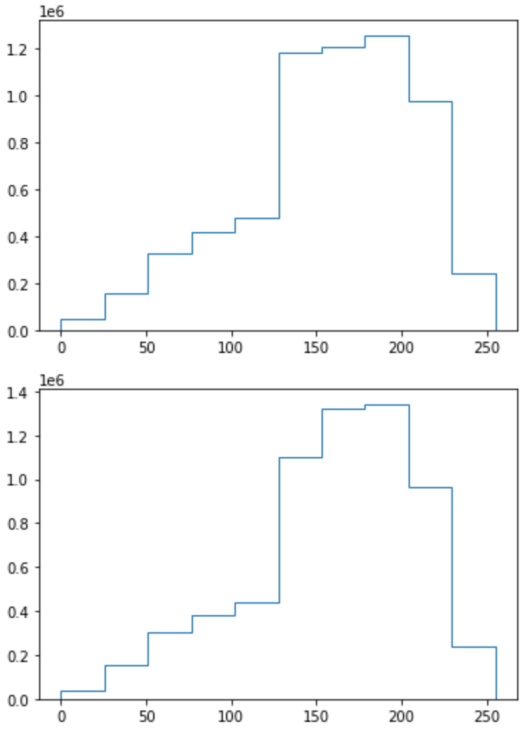


Рисунок 4.17. Гістограми для класу cyt_nep_mast

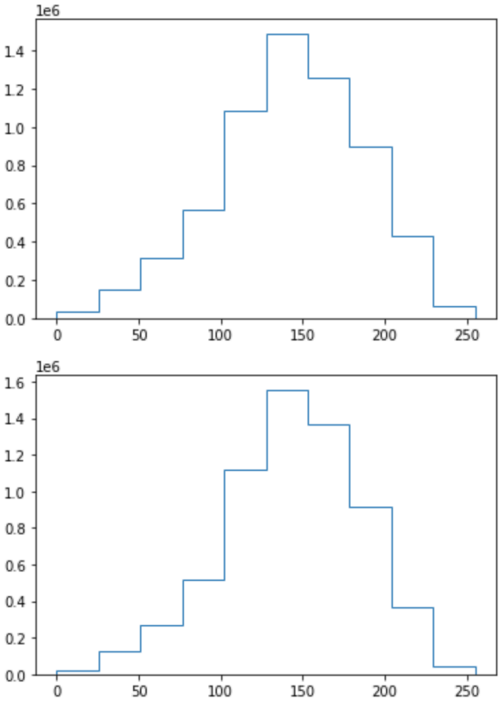


Рисунок 4.18. Гістограми для класу cyt_nep_fib_mast

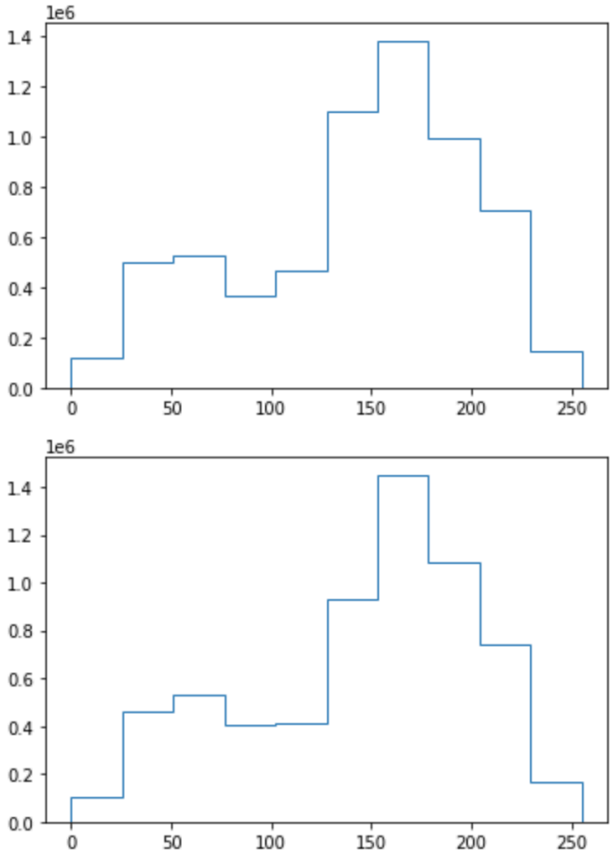


Рисунок 4.19. Гістограми для класу cyt_mast

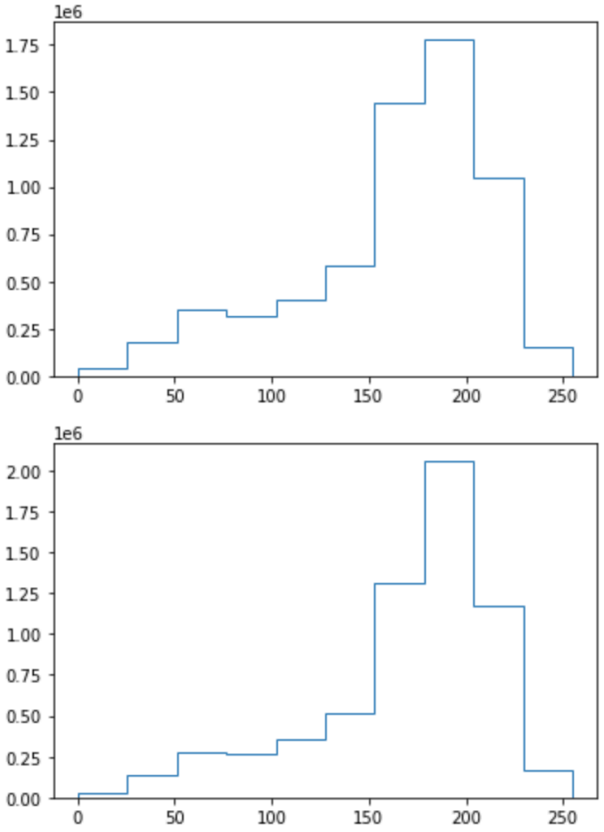


Рисунок 4.20. Гістограми для класу cyt_kist_mast

У таблиці 4.5 наведено значення дивергенції KL між оригінальними та синтетичними зображеннями для кожного класу зображень. Для порівняння обрано 150 оригінальних і 150 синтетичних зображень із кожного класу.

Таблиця 4.5. KL-дивергенція для кожного класу зображень

Клас	KL
cyt_nep_mast	0,0032
cyt_nep_fib_mast	0,0041
cyt_mast	0,0047
cyt_kist_mast	0,0095

Тепер додамо новий розширений навчальний набір даних цитологічних зображень. Цей набір буде містити ті самі дані, що і у всіх попердніх експериментах (800 зображень), але також і 16 000 синтетичних зображень. Назвемо його *cyt_64_extended_synthetic*.

Для проведення другого експерименту по класифікації в якості класифікатора обрано синтезовану архітектуру із розділу 2 (рисунок 4.21). Параметри навчання встановлено у такі самі значення, як і в розділі 2, але в якості оптимізатора використано алгоритм Adagrad. Мережа навчалася протягом 100 епох. В результаті експерименту модель досягла точності класифікації 99,6%. ROC криву показано на рисунку 4.22 (код у додатку Е).

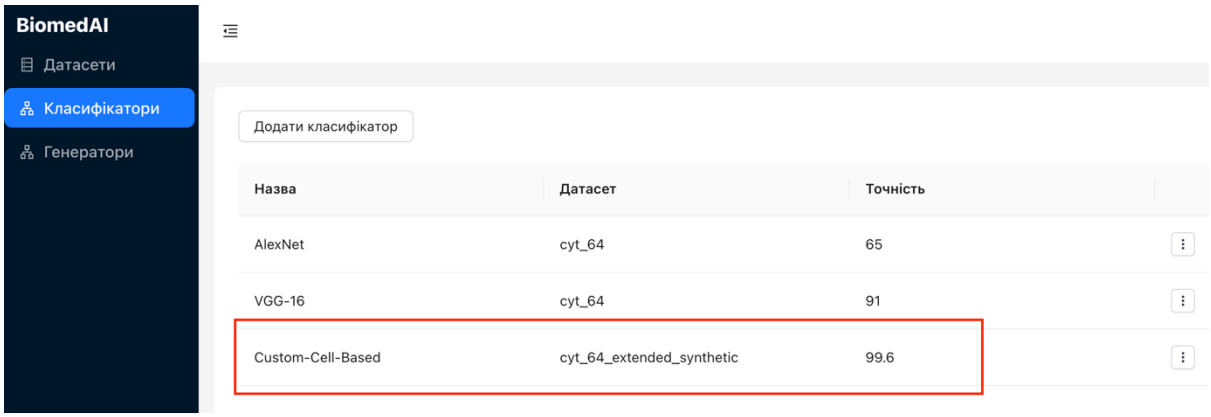


Рисунок 4.21. Вікно навчених класифікаторів

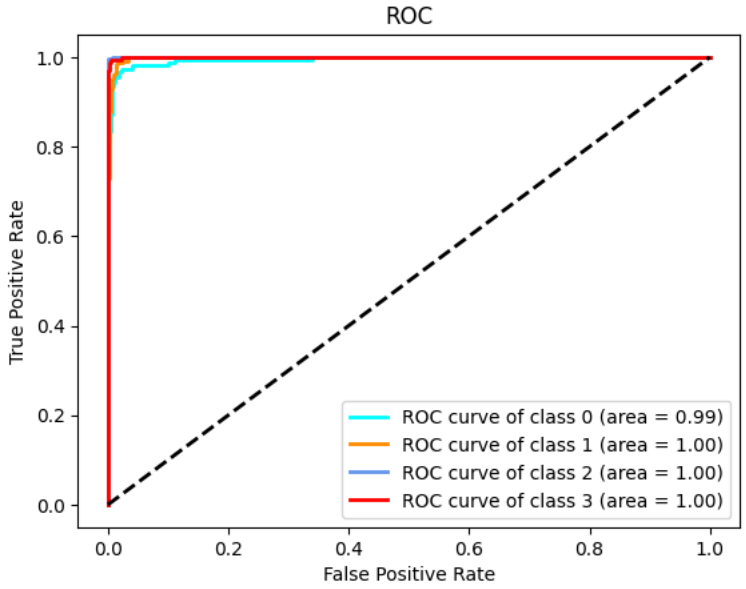


Рисунок 4.22. ROC-крива класифікатора на розширеногму наборі даних

4.6 Висновки до розділу 4

1. Спроектовано архітектуру програмного засобу, використовуючи сучасні, надійні технології та підходи в сфері розробки програмного забезпечення, що дасть можливість легко масштабувати систему в майбутньому відповідно до навантаження та обсягів даних, забезпечити високий рівень безпеки медичної інформації та оптимізувати взаємодію між модулями системи, що в результаті сприятиме ефективній інтеграції розробленого засобу в медичні інформаційні системи.

2. Розроблено структуру організації бази даних використовуючи UML-діаграму класів для надійного зберігання даних системи, яка забезпечить ефективне управління великими обсягами біомедичних зображень і відповідних метаданих, оптимізацію процесів пошуку та вибірки необхідної інформації для аналізу, а також сприятиме забезпеченню цілісності та конфіденційності медичних даних.

3. Розроблено програмний засіб для синтезу та класифікації біомедичних зображень. Даний засіб можна використовувати для розширення навчальних наборів даних з подальшим їх використанням для навчання класифікаторів. Програмний засіб побудований із використанням клієнт-серверної архітектури

та модульного підходу, який дає змогу в майбутньому масштабувати систему горизонтально (приклад коду у додатку Ж). Також такий підхід дозволяє розгортати окремі модулі системи як на локальному комп’ютері, так і на сервісах хмарних обчислень.

ВИСНОВКИ

У дисертаційні роботі розв’язано наукову задачу — розроблення, вдосконалення методів і засобів синтезу та розширення навчальних наборів для підвищення точності класифікації глибокими нейронними мережами біомедичних зображень.

При цьому отримано такі результати:

1. Проведено аналіз нейромережевих методів і засобів синтезу зображень, обґрунтовано вибір генеративно-змагальних нейронних мереж для задач генерування біомедичних зображень та сформульовано завдання дослідження.

2. Розроблено метод автоматичного синтезу архітектур згорткових нейронних мереж для класифікації біомедичних зображень, який за рахунок використання фаз мікропошуку та макропошуку забезпечує створення нейромереж з точністю класифікації біомедичних зображень 99,125%.

3. Розроблено метод автоматичного синтезу архітектур генеративно-змагальних нейронних мереж для задач генерування біомедичних зображень, який за рахунок використання механізмів самоуваги в генераторі і дискримінаторі та синтезу зображень за мітками забезпечує за метрикою FID підвищення якості синтезованих зображень з параметром FID – 3,39

4. Вдосконалено метод генерування та класифікації біомедичних зображень, який за рахунок використання методів автоматичного синтезу архітектур згорткових та генеративно-змагальних нейронних мереж забезпечив розширення та доповнення навчальної вибірки біомедичних зображень для навчання згорткових нейронних мереж.

5. Вдосконалено модель опису архітектур нейронних мереж, яка за рахунок використання теоретико-множинного підходу, забезпечила формалізацію представлення згорткових і генеративно-змагальних нейронних мереж.

6. Розроблено засоби автоматичного синтезу архітектур згорткових і генеративно-змагальних нейронних мереж з використанням клієнт-серверної

архітектури, модульного підходу та мови програмування Python і Typescript використання яких спрощує процес створення архітектур для синтезу та класифікації цитологічних зображень.

7. Розроблено засіб генерування та класифікації біомедичних зображень з використанням клієнт-серверної архітектури, модульного підходу та мови програмування Python і Typescript, використання яких забезпечує розширення і доповнення навчальних вибірок цитологічних зображень.

8. Розроблено базу даних для зберігання синтетичних зображень і архітектур генеративно-змагальних-нейронних мереж, використання яких забезпечує підвищення точності навчання згорткових нейронних мереж.

9. Проведено комп'ютерні експерименти синтезу та класифікації цитологічних зображень використовуючи розроблені методи. В результаті експериментів створено 16 000 синтетичних цитологічних зображень, які були об'єднані з початковою навчальною вибіркою (800 зображень). Точність розробленої згорткової нейронної мережі на розширеній вибірці склала 99,6%, що на 5-11% більше порівняно з класичними архітектурами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] О. М. Березький, О. Й. Піцун, П. Б. Лящинський, П. Б. Лящинський, і Г. М. Мельник, «Інтелектуальна система автоматизованої мікроскопії аналізу гістологічних та цитологічних зображень», *Штучний Інтелект*, вип. 2, вип. 76, 2017, Дата звернення: 16, Квітень 2024. [Online]. Доступний у: http://dspace.nbuv.gov.ua/handle/123456789/133671

[2] О. М. Березький, П. М. Лящинський, А. Р. Сухович, і Т. М. Долинюк, «Синтез біомедичних зображень на підставі генеративно-змагальних мереж», *Український Журнал Інформаційних Технологій*, вип. 1, вип. 1, с. 35–40, 2019.

[3] O. M. Berezsky і P. B. Liashchynskyi, «Comparison of generative adversarial networks architectures for biomedical images synthesis», *Appl. Asp. Inf. Technol.*, вип. 4, вип. 3, с. 250–260, Жов 2021, doi: 10.15276/aait.03.2021.4.

[4] P. Liashchynskyi і P. Liashchynskyi, «Analysis of metrcis for GAN evaluation», *Comput. Syst. Inf. Technol.*, вип. 4, с. 44–51, 2023, doi: 10.31891/csit-2023-4-6.

[5] O. M. Berezsky, P. B. Liashchynskyi, O. Y. Pitsun, і G. M. Melnyk, «Deep network-based method and software for small sample biomedical image generation and classification», *Radio Electron. Comput. Sci. Control*, вип. 4, Art. вип. 4, 2023, doi: 10.15588/1607-3274-2023-4-8.

[6] O. M. Berezsky і P. B. Liashchynskyi, «Method of generative-adversarial networks searching architectures for biomedical images synthesis», *Radio Electron. Comput. Sci. Control*, вип. 1, Art. вип. 1, Квіт 2024, doi: 10.15588/1607-3274-2024-1-10.

[7] П. Лящинський, «Програмний засіб для синтезу та класифікації біомедичних зображень», *Науковий Вісник НЛТУ України*, вип. 34, вип. 4.

[8] P. Liashchynskyi і P. Liashchynskyi, «Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS». arXiv, 12, Грудень 2019. doi: 10.48550/arXiv.1912.06059.

[9] O. M. Berezsky, P. B. Liashchynskyi, O. Y. Pitsun, і I. Izonin, «Synthesis of Convolutional Neural Network architectures for biomedical image classification», *Biomed. Signal Process. Control*, вип. 95, вип. 106325, 2024, doi: https://doi.org/10.1016/j.bspc.2024.106325.

[10] O. Berezsky, O. Pitsun, P. Liashchynskyi, B. Derysh, і N. Batryn, «Computational Intelligence in Medicine», в *Lecture Notes in Data Engineering, Computational Intelligence, and Decision Making*, вип. 149, S. Babichev і V. Lytvynenko, Ред., в Lecture Notes on Data Engineering and Communications Technologies, vol. 149. , Cham: Springer International Publishing, 2023, с. 488–510. doi: 10.1007/978-3-031-16203-9_28.

[11] O. Berezsky, O. Pitsun, L. Dubchak, P. Liashchynskyi, і P. Liashchynskyi, «GPU-based biomedical image processing», в *2018 XIV-th International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, IEEE, 2018, с. 96–99. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: https://ieeexplore.ieee.org/abstract/document/8365710/

[12] П. Б. Лящинський, «Порівняння GAN-архітектур для синтезу біомедичних зображень», представлена на III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі», ТНЕУ, 2020, с. 31.

[13] O. Berezsky, P. Liashchynskyi, O. Pitsun, P. Liashchynskyi, і M. Berezkyy, «Comparison of Deep Neural Network Learning Algorithms for Biomedical Image Processing.», в *CEUR Workshop Proceedings*, 2022, с. 135–145. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: https://ceur-ws.org/Vol-3302/paper7.pdf

[14] O. Berezsky, O. Pitsun, G. Melnyk, Y. Batko, B. Derysh, і P. Liashchynskyi, «Application Of MLOps Practices For Biomedical Image Classification.», в *CEUR Workshop Proceedings*, 2022, с. 69–77. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: https://ceur-ws.org/Vol-3302/short3.pdf

[15] O. M. Berezsky, P. B. Liashchynskyi, O. Y. Pitsun, і G. M. Melnyk, «Method and Software Tool for Generating Artificial Databases of Biomedical Images

Based on Deep Neural Networks», в *CEUR Workshop Proceedings*, 2023, с. 15–26. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: https://ceur-ws.org/Vol-3609/paper2.pdf

[16] O. M. Berezsky, O. Y. Pitsun, G. M. Melnyk, і P. B. Liashchynskyi, «MLOps Approach for Automatic Segmentation of Biomedical Images», в *CEUR Workshop Proceedings*, 6th International Conference on Informatics & Data-Driven Medicine, November 17 - 19, 2023, Bratislava, Slovakia, 2023, с. 241–248. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: https://ceur-ws.org/Vol-3609/short5.pdf

[17] П. Б. Лящинський і П. Б. Лящинський, «Використання систем автоматизованої мікроскопії для покращення аналізу біомедичних зображень», в *Економічний і соціальний розвиток України в XXI столітті: національна візія та виклики глобалізації*, с. 729. Дата звернення: 16, Квітень 2024. [Online]. Доступний у: http://dspace.wunu.edu.ua/bitstream/316497/49651/1/%D0%95%D0%BA%D0%BE%D0%BD.%20%D1%82%D0%B0%20%D1%81%D0%BE%D1%86.%20%D1%80%D0%BE%D0%B7%D0%B2.%2019.05.23.pdf#page=730

[18] «BCADCID breast cancer artificial digital cytology image database.: copyright certificate 123607 / O. M. Berezsky, P. B. Liashchynskyi, G. M. Melnyk, O. Y. Pitsun Application. November 20, 2023, published February 8, 2024.»

[19] О. М. Березький *et al.*, *Методи, алгоритми і програмні засоби опрацювання біомедичних зображень*. 2017. Дата звернення: 20, Квітень 2024. [Online]. Доступний у: http://dspace.wunu.edu.ua//handle/316497/42322

[20] O. M. Berezsky, «Hybrid Intelligent System for Diagnosing Breast Pre-Cancerous and Cancerous Conditions Based on Image Analysis», в *Intelligent System*, IntechOpen, 2018. doi: 10.5772/intechopen.72576.

[21] O. Pitsun, N. Batryn, T. Datsko, K. Berezska, L. Dubchak, і O. Berezsky, «Modern automated microscopy systems in oncology», в *CEUR Workshop Proceedings*, 2018, с. 311–325.

[22] M. Lapan, «Deep Reinforcement Learning Hands-On - Second Edition», Packt. Дата звернення: 20, Квітень 2024. [Online]. Доступний у: https://www.packtpub.com/product/deep-reinforcement-learning-hands-on-second-edition/9781838826994

[23] Y. LeCun і Y. Bengio, «Convolutional networks for images, speech, and time series», *Handb. Brain Theory Neural Netw.*, вип. 3361, вип. 10, с. 1995, 1995.

[24] A. Krizhevsky, I. Sutskever, і G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», в *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Дата звернення: 29, Травень 2022. [Online]. Доступний у: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[25] Y. LeCun, Y. Bengio, і G. Hinton, «Deep learning», *nature*, вип. 521, вип. 7553, с. 436–444, 2015.

[26] I. Goodfellow *et al.*, «Generative Adversarial Nets», в *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html

[27] I. Goodfellow, «NIPS 2016 Tutorial: Generative Adversarial Networks». arXiv, 03, Квітень 2017. doi: 10.48550/arXiv.1701.00160.

[28] T. Salimans *et al.*, «Improved Techniques for Training GANs», в *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, і R. Garnett, Ред., Curran Associates, Inc., 2016. [Online]. Доступний у: https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf

[29] A. G. Ivakhnenko, D. Wunsch, і G. A. Ivakhnenko, «Inductive sorting-out GMDH algorithms with polynomial complexity for active neurons of neural network», в *IJCNN'99. International Joint Conference on Neural Networks.*

Proceedings (Cat. No.99CH36339), Лип 1999, с. 1169–1173 вип.2. doi: 10.1109/IJCNN.1999.831124.

[30] О. Г. Руденко і Є. В. Бодянський, «Штучні нейронні мережі», *Харків Компанія CMIT*, 2006, Дата звернення: 20, Квітень 2024. [Online]. Доступний у: https://scholar.google.com/scholar?cluster=11264414915704044577&hl=en&oi=scholarr

[31] Є. В. Бодянський і С. О. Костюк, «Нейрон на основі адаптивного нечіткого перетворення для сучасних моделей штучних нейронних мереж», *Int. Sci. Tech. J. Probl. Control Inform.*, вип. 68, вип. 6, с. 94–105, 2023.

[32] С. О. Субботін і С. А. Субботин, «Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень». Запорізький національний технічний університет, 2008. Дата звернення: 20, Квітень 2024. [Online]. Доступний у: http://eir.zp.edu.ua/handle/123456789/2157

[33] А. О. Олійник, А. А. Олейник, С. О. Субботін, С. А. Субботин, О. О. Олійник, і А. А. Олейник, «Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей. Монографія». Запорізький національний технічний університет, 2009. Дата звернення: 20, Квітень 2024. [Online]. Доступний у: http://eir.zp.edu.ua/bitstream/123456789/2643/1/Oliinyk_Non-iterative.pdf

[34] V. M. Lovkin, S. A. Subbotin, A. O. Oliinyk, і N. V. Myronenko, «Method and software component model for skin disease diagnosis», *Radio Electron. Comput. Sci. Control*, вип. 1, с. 40–40, 2023.

[35] A. Vlasenko, N. Vlasenko, O. Vynokurova, і D. Peleshko, «An Empirical Mode Decomposition Based Method to Synthesize Ensemble Multidimensional Gaussian Neuro-Fuzzy Models in Financial Forecasting», в *Data Stream Mining & Processing*, S. Babichev, D. Peleshko, і O. Vynokurova, Ред., Cham: Springer International Publishing, 2020, с. 140–149. doi: 10.1007/978-3-030-61656-4_9.

[36] O. Vynokurova, D. Peleshko, O. Bondarenko, V. Ilyasov, V. Serzhantov, і M. Peleshko, «Hybrid Machine Learning System for Solving Fraud Detection Tasks», в *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*, Сер 2020, с. 1–5. doi: 10.1109/DSMP47368.2020.9204244.

[37] L. Seguin, M. Durandy, і C. C. Feral, «Lung Adenocarcinoma Tumor Origin: A Guide for Personalized Medicine», *Cancers*, вип. 14, вип. 7, Art. вип. 7, Січ 2022, doi: 10.3390/cancers14071759.

[38] S. Chen, M. Zhao, G. Wu, C. Yao, і J. Zhang, «Recent Advances in Morphological Cell Image Analysis», *Comput. Math. Methods Med.*, вип. 2012, с. 101536, 2012, doi: 10.1155/2012/101536.

[39] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, і B. Ommer, «High-Resolution Image Synthesis with Latent Diffusion Models». arXiv, 13, Квітень 2022. doi: 10.48550/arXiv.2112.10752.

[40] OpenAI *et al.*, «GPT-4 Technical Report». arXiv, 04, Березень 2024. doi: 10.48550/arXiv.2303.08774.

[41] M. Abadi *et al.*, «TensorFlow: A system for large-scale machine learning». arXiv, 31, Травень 2016. doi: 10.48550/arXiv.1605.08695.

[42] A. Paszke *et al.*, «PyTorch: An Imperative Style, High-Performance Deep Learning Library». arXiv, 03, Грудень 2019. doi: 10.48550/arXiv.1912.01703.

[43] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, і M. Cifrek, «A brief introduction to OpenCV», в *2012 Proceedings of the 35th International Convention MIPRO*, Трав 2012, с. 1725–1730. Дата звернення: 02, Червень 2024. [Online]. Доступний у: https://ieeexplore.ieee.org/document/6240859

[44] «Pillow», Pillow (PIL Fork). Дата звернення: 02, Червень 2024. [Online]. Доступний у: https://pillow.readthedocs.io/en/stable/index.html

[45] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, і A. A. Bharath, «Generative Adversarial Networks: An Overview», *IEEE Signal Process. Mag.*, вип. 35, вип. 1, с. 53–65, Січ 2018, doi: 10.1109/MSP.2017.2765202.

[46] C. Donahue, J. McAuley, і M. Puckette, «Adversarial Audio Synthesis». arXiv, 08, Лютий 2019. doi: 10.48550/arXiv.1802.04208.

[47] Y. He, K. P. Seng, і L. M. Ang, «Generative Adversarial Networks (GANs) for Audio-Visual Speech Recognition in Artificial Intelligence IoT», *Information*, вип. 14, вип. 10, Art. вип. 10, Жов 2023, doi: 10.3390/info14100575.

[48] S. Bahmani *et al.*, «3D-Aware Video Generation». arXiv, 09, Серпень 2023. doi: 10.48550/arXiv.2206.14797.

[49] Y. Liu *et al.*, «Sora: A Review on Background, Technology, Limitations, and Opportunities of Large Vision Models». arXiv, 28, Лютий 2024. doi: 10.48550/arXiv.2402.17177.

[50] G. H. de Rosa і J. P. Papa, «A survey on text generation using generative adversarial networks», *Pattern Recognit.*, вип. 119, с. 108098, Лис 2021, doi: 10.1016/j.patcog.2021.108098.

[51] H. Huang, P. S. Yu, і C. Wang, «An Introduction to Image Synthesis with Generative Adversarial Nets». arXiv, 17, Листопад 2018. doi: 10.48550/arXiv.1803.04469.

[52] T. Che, Y. Li, A. P. Jacob, Y. Bengio, і W. Li, «Mode Regularized Generative Adversarial Networks». arXiv, 02, Березень 2017. doi: 10.48550/arXiv.1612.02136.

[53] Y.-D. Zhang, V. V. Govindaraj, C. Tang, W. Zhu, і J. Sun, «High Performance Multiple Sclerosis Classification by Data Augmentation and AlexNet Transfer Learning Model», *J. Med. Imaging Health Inform.*, вип. 9, вип. 9, с. 2012–2021, Груд 2019, doi: 10.1166/jmihi.2019.2692.

[54] L. Lan *et al.*, «Generative Adversarial Networks and Its Applications in Biomedical Informatics», *Front. Public Health*, вип. 8, Трав 2020, doi: 10.3389/fpubh.2020.00164.

[55] O. M. Berezsky, P. B. Liashchynskyi, P. B. Liashchynskyi, A. R. Sukhovych, і T. M. Dolynyuk, «Synthesis of biomedical images based on generative adversarial networks», *Ukr. J. Inf. Technol.*, с. 35, Лис 2019.

[56] G. Zhang, «Risk Prediction Model for Knee Arthroplasty | IEEE Journals & Magazine | IEEE Xplore», вип. 7, с. 34645–34654, 2019, doi: 10.1109/ACCESS.2019.2900619.

[57] S. Pandey, P. Singh, і J. Tian, «An image augmentation approach using two-stage generative adversarial network for nuclei image segmentation», *Biomed Signal Process Control*, 2020, doi: 10.1016/j.bspc.2019.101782.

[58] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, і H. Greenspan, «GAN-based Synthetic Medical Image Augmentation for increased CNN Performance in Liver Lesion Classification», *Neurocomputing*, вип. 321, с. 321–331, Груд 2018, doi: 10.1016/j.neucom.2018.09.013.

[59] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, і S. Hochreiter, «GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium», *ArXiv170608500 Cs Stat*, Січ 2018, Дата звернення: 27, Листопад 2020. [Online]. Доступний у: http://arxiv.org/abs/1706.08500

[60] V. P. Crawford, «Learning the Optimal Strategy in a Zero-Sum Game», *Econometrica*, вип. 42, вип. 5, с. 885–891, 1974, doi: 10.2307/1913795.

[61] L. Weng, «From GAN to WGAN». arXiv, 18, Квітень 2019. doi: 10.48550/arXiv.1904.08994.

[62] S. Albawi, T. A. Mohammed, і S. Al-Zawi, «Understanding of a convolutional neural network», в *2017 International Conference on Engineering and Technology (ICET)*, Сер 2017, с. 1–6. doi: 10.1109/ICEngTechnol.2017.8308186.

[63] Z. Zhang, M. Li, і J. Yu, «On the convergence and mode collapse of GAN», в *SIGGRAPH Asia 2018 Technical Briefs*, в SA '18. New York, NY, USA: Association for Computing Machinery, Груд 2018, с. 1–4. doi: 10.1145/3283254.3283282.

[64] M. Arjovsky, S. Chintala, і L. Bottou, «Wasserstein Generative Adversarial Networks», в *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Лип 2017, с. 214–223. Дата звернення: 29, Травень 2022. [Online]. Доступний у: https://proceedings.mlr.press/v70/arjovsky17a.html

[65] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, і A. C. Courville, «Improved Training of Wasserstein GANs», представлена на Neural Information Processing Systems, Бер 2017. Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Improved-Training-of-Wasserstein-GANs-Gulrajani-Ahmed/edf73ab12595c6709f646f542a0d2b33eb20a3f4

[66] R. D. Hjelm, A. P. Jacob, T. Che, A. Trischler, K. Cho, і Y. Bengio, «Boundary-Seeking Generative Adversarial Networks». arXiv, 21, Лютий 2018. doi: 10.48550/arXiv.1702.08431.

[67] D. Berthelot, T. Schumm, і L. Metz, «BEGAN: Boundary Equilibrium Generative Adversarial Networks», arXiv, arXiv:1703.10717, Трав 2017. doi: 10.48550/arXiv.1703.10717.

[68] A. Brock, J. Donahue, і K. Simonyan, «Large Scale GAN Training for High Fidelity Natural Image Synthesis», *ArXiv*, Вер 2018, Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Large-Scale-GAN-Training-for-High-Fidelity-Natural-Brock-Donahue/22aab110058ebbd198edb1f1e7b4f69fb13c0613

[69] T. Karras, S. Laine, і T. Aila, «A Style-Based Generator Architecture for Generative Adversarial Networks». arXiv, 29, Березень 2019. doi: 10.48550/arXiv.1812.04948.

[70] K. He, X. Zhang, S. Ren, і J. Sun, «Deep Residual Learning for Image Recognition», *2016 IEEE Conf. Comput. Vis. Pattern Recognit. CVPR*, с. 770–778, Чер 2016, doi: 10.1109/CVPR.2016.90.

[71] K. Simonyan і A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition», arXiv, arXiv:1409.1556, Квіт 2015. doi: 10.48550/arXiv.1409.1556.

[72] S. Amari, «Backpropagation and stochastic gradient descent method», *Neurocomputing*, вип. 5, вип. 4, с. 185–196, Чер 1993, doi: 10.1016/0925-2312(93)90006-O.

[73] S. Khirirat, H. R. Feyzmahdavian, і M. Johansson, «Mini-batch gradient descent: Faster convergence under data sparsity», в *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Груд 2017, с. 2880–2887. doi: 10.1109/CDC.2017.8264077.

[74] J. Duchi, E. Hazan, і Y. Singer, «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization».

[75] M. D. Zeiler, «ADADELTA: An Adaptive Learning Rate Method». arXiv, 22, Грудень 2012. doi: 10.48550/arXiv.1212.5701.

[76] D. P. Kingma і J. Ba, «Adam: A Method for Stochastic Optimization», *CoRR*, Груд 2014, Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Adam%3A-A-Method-for-Stochastic-Optimization-Kingma-Ba/a6cb366736791bcccc5c8639de5a8f9636bf87e8

[77] O. Berezsky, T. Datsko, і G. Melnyk, «Cytological and histological images of breast cancer». Zenodo, 03, Травень 2023. doi: 10.5281/zenodo.7890874.

[78] P. Liashchynskyi, «Rudi: Lightweight image converter and dataset augmentor». Zenodo, 22, Серпень 2019. doi: 10.5281/zenodo.3374946.

[79] D. J. Im, H. Ma, G. Taylor, і K. Branson, «Quantitatively Evaluating GANs With Divergences Proposed for Training». arXiv, 28, Квітень 2018. doi: 10.48550/arXiv.1803.01045.

[80] A. Kontorovich, S. Sabato, і R. Urner, «Active Nearest-Neighbor Learning in Metric Spaces». arXiv, 31, Жовтень 2018. doi: 10.48550/arXiv.1605.06792.

[81] L. Theis, A. van den Oord, і M. Bethge, «A note on the evaluation of generative models», *ArXiv151101844 Cs Stat*, Квіт 2016, Дата звернення: 14, Листопад 2021. [Online]. Доступний у: http://arxiv.org/abs/1511.01844

[82] Q. Xu *et al.*, «An empirical study on evaluation metrics of generative adversarial networks», *ArXiv180607755 Cs Stat*, Сер 2018, Дата звернення: 14, Листопад 2021. [Online]. Доступний у: http://arxiv.org/abs/1806.07755

[83] S. Barratt і R. Sharma, «A Note on the Inception Score». arXiv, 21, Червень 2018. Дата звернення: 17, Травень 2022. [Online]. Доступний у: http://arxiv.org/abs/1801.01973

[84] A. Borji, «Pros and cons of GAN evaluation measures», *Comput. Vis. Image Underst.*, вип. 179, с. 41–65, Лют 2019, doi: 10.1016/j.cviu.2018.10.009.

[85] K. Shmelkov, C. Schmid, і K. Alahari, «How good is my GAN?», arXiv, arXiv:1807.09499, Лип 2018. doi: 10.48550/arXiv.1807.09499.

[86] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, і S. Gelly, «Are GANs created equal? a large-scale study», в *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, в NIPS'18. Red Hook, NY, USA: Curran Associates Inc., Груд 2018, с. 698–707.

[87] C. Szegedy *et al.*, «Going Deeper With Convolutions», представлена на Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, с. 1–9. Дата звернення: 29, Травень 2022. [Online]. Доступний у: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html

[88] «AN APPROACH TO ACCELERATE THE TRAINING OF CONVOLUTIONAL NEURAL NETWORKS BY TUNING THE HYPERPARAMETERS OF LEARNING». Дата звернення: 03, Квітень 2024. [Online]. Доступний у: https://ouci.dntb.gov.ua/en/works/legQdwK7/

[89] H. Liu, K. Simonyan, і Y. Yang, «DARTS: Differentiable Architecture Search». arXiv, 23, Квітень 2019. Дата звернення: 31, Липень 2022. [Online]. Доступний у: http://arxiv.org/abs/1806.09055

[90] E. Real *et al.*, «Large-scale evolution of image classifiers», в *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, в ICML'17. Sydney, NSW, Australia: JMLR.org, Сер 2017, с. 2902–2911.

[91] B. Zoph і Q. V. Le, «Neural Architecture Search with Reinforcement Learning». arXiv, 15, Лютий 2017. doi: 10.48550/arXiv.1611.01578.

[92] D. Stamoulis *et al.*, «Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours». arXiv, 05, Квітень 2019. doi: 10.48550/arXiv.1904.02877.

[93] A. M. Vincent і P. Jidesh, «An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms», *Sci. Rep.*, вип. 13, вип. 1, с. 4737, Бер 2023, doi: 10.1038/s41598-023-32027-3.

[94] P. Ren *et al.*, «A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions». arXiv, 02, Березень 2021. Дата звернення: 01, Серпень 2022. [Online]. Доступний у: http://arxiv.org/abs/2006.02903

[95] D. Baymurzina, E. Golikov, і M. Burtsev, «A review of neural architecture search», *Neurocomputing*, вип. 474, с. 82–93, Лют 2022, doi: 10.1016/j.neucom.2021.12.014.

[96] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, і K. C. Tan, «A Survey on Evolutionary Neural Architecture Search», *IEEE Trans. Neural Netw. Learn. Syst.*, с. 1–21, 2021, doi: 10.1109/TNNLS.2021.3100554.

[97] B. Baker, O. Gupta, R. Raskar, і N. Naik, «Practical Neural Network Performance Prediction for Early Stopping», *ArXiv*, Трав 2017, Дата звернення: 06, Квітень 2024. [Online]. Доступний у: https://www.semanticscholar.org/paper/Practical-Neural-Network-Performance-Prediction-for-Baker-Gupta/8cb79a3d446af39b72abb24564b0809d23c43f06

[98] B. Baker, O. Gupta, N. Naik, і R. Raskar, «Designing Neural Network Architectures using Reinforcement Learning». arXiv, 22, Березень 2017. doi: 10.48550/arXiv.1611.02167.

[99] J. Zhao, R. Zhang, Z. Zhou, S. Chen, J. Jin, і Q. Liu, «A neural architecture search method based on gradient descent for remaining useful life estimation», *Neurocomputing*, вип. 438, с. 184–194, Трав 2021, doi: 10.1016/j.neucom.2021.01.072.

[100] T. Elsken, J. H. Metzen, і F. Hutter, «Neural Architecture Search: A Survey». arXiv, 26, Квітень 2019. doi: 10.48550/arXiv.1808.05377.

[101] A. Shah, E. Kadam, H. Shah, S. Shinde, і S. Shingade, «Deep Residual Networks with Exponential Linear Unit», в *Proceedings of the Third International Symposium on Computer Vision and the Internet*, в VisionNet’16.

New York, NY, USA: Association for Computing Machinery, Вер 2016, с. 59–65. doi: 10.1145/2983402.2983406.

[102] G. Huang, Z. Liu, L. V. D. Maaten, і K. Q. Weinberger, «Densely Connected Convolutional Networks», представлена на 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, Лип 2017, с. 2261–2269. doi: 10.1109/CVPR.2017.243.

[103] S. Ioffe і C. Szegedy, «Batch normalization: accelerating deep network training by reducing internal covariate shift», в *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, в ICML'15. Lille, France: JMLR.org, Лип 2015, с. 448–456.

[104] G. Kyriakides і K. Margaritis, «An Introduction to Neural Architecture Search for Convolutional Networks». arXiv, 22, Травень 2020. Дата звернення: 01, Серпень 2022. [Online]. Доступний у: http://arxiv.org/abs/2005.11074

[105] E. Real, A. Aggarwal, Y. Huang, і Q. V. Le, «Regularized Evolution for Image Classifier Architecture Search», *Proc. AAAI Conf. Artif. Intell.*, вип. 33, вип. 01, с. 4780–4789, Лип 2019, doi: 10.1609/aaai.v33i01.33014780.

[106] «Gradient-based learning applied to document recognition | IEEE Journals & Magazine | IEEE Xplore». Дата звернення: 07, Квітень 2024. [Online]. Доступний у: https://ieeexplore.ieee.org/document/726791

[107] L. Hertel, E. Barth, T. Käster, і T. Martinetz, «Deep Convolutional Neural Networks as Generic Feature Extractors». arXiv, 06, Жовтень 2017. doi: 10.48550/arXiv.1710.02286.

[108] A. Howard *et al.*, «Searching for MobileNetV3», представлена на 2019 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE Computer Society, Жов 2019, с. 1314–1324. doi: 10.1109/ICCV.2019.00140.

[109] M. E. Plissiti, P. Dimitrakopoulos, G. Sfikas, C. Nikou, O. Krikoni, і A. Charchanti, «Sipakmed: A New Dataset for Feature and Image Based Classification of Normal and Pathological Cervical Cells in Pap Smear Images», в *2018 25th IEEE International Conference on Image Processing (ICIP)*, Жов 2018, с. 3144–3148. doi: 10.1109/ICIP.2018.8451588.

[110]M. Wu, C. Yan, H. Liu, Q. Liu, і Y. Yin, «Automatic classification of cervical cancer from cytological images by using convolutional neural network», *Biosci. Rep.*, вип. 38, вип. 6, с. BSR20181769, Груд 2018, doi: 10.1042/BSR20181769.

[111]K. P. Win, Y. Kitjaidure, K. Hamamoto, і T. Myo Aung, «Computer-Assisted Screening for Cervical Cancer Using Digital Image Processing of Pap Smear Images», *Appl. Sci.*, вип. 10, вип. 5, Art. вип. 5, Січ 2020, doi: 10.3390/app10051800.

[112]«Deep Learning based Classification of Cervical Cancer using Transfer Learning | IEEE Conference Publication | IEEE Xplore». Дата звернення: 07, Квітень 2024. [Online]. Доступний у: https://ieeexplore.ieee.org/document/9783560

[113]«Cervical cell classification using Deep Learning Techniques | IEEE Conference Publication | IEEE Xplore». Дата звернення: 07, Квітень 2024. [Online]. Доступний у: https://ieeexplore.ieee.org/document/10128238

[114]O. Düzyel і M. Kuntalp, «Data Augmentation with GAN increases the Performance of Arrhythmia Classification for an Unbalanced Dataset». arXiv, 24, Лютий 2023. doi: 10.48550/arXiv.2302.13855.

[115]P. Costa *et al.*, «End-to-End Adversarial Retinal Image Synthesis», *IEEE Trans. Med. Imaging*, вип. 37, вип. 3, с. 781–791, Бер 2018, doi: 10.1109/TMI.2017.2759102.

[116]X. Gong, S. Chang, Y. Jiang, і Z. Wang, «AutoGAN: Neural Architecture Search for Generative Adversarial Networks», представлена на 2019 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE Computer Society, Жов 2019, с. 3223–3233. doi: 10.1109/ICCV.2019.00332.

[117]G. Ying, X. He, B. Gao, B. Han, і X. Chu, «EAGAN: Efficient Two-Stage Evolutionary Architecture Search for GANs», вип. 13676, с. 37–53, 2022, doi: 10.1007/978-3-031-19787-1_3.

[118]C. Gao, Y. Chen, S. Liu, Z. Tan, і S. Yan, «AdversarialNAS: Adversarial Neural Architecture Search for GANs», в *2020 IEEE/CVF Conference on*

Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA: IEEE, Чер 2020, с. 5679–5688. doi: 10.1109/CVPR42600.2020.00572.

[119] A. Vaswani *et al.*, «Attention is All you Need», представлена на Neural Information Processing Systems, Чер 2017. Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Attention-is-All-you-Need-Vaswani-Shazeer/204e3073870fae3d05bcbc2f6a8e263d9b72e776

[120] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu, і S. Han, «Differentiable Augmentation for Data-Efficient GAN Training», *ArXiv*, Чер 2020, Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Differentiable-Augmentation-for-Data-Efficient-GAN-Zhao-Liu/670f9d0d8cafaeaeea564c88645b9816b1146cef

[121] J. H. Lim і J. C. Ye, «Geometric GAN», *ArXiv*, Трав 2017, Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Geometric-GAN-Lim-Ye/5e4d2e73e71806e9147afbedcd504424721fa059

[122] T. Miyato, T. Kataoka, M. Koyama, і Y. Yoshida, «Spectral Normalization for Generative Adversarial Networks», *ArXiv*, Лют 2018, Дата звернення: 05, Листопад 2023. [Online]. Доступний у: https://www.semanticscholar.org/paper/Spectral-Normalization-for-Generative-Adversarial-Miyato-Kataoka/84de7d27e2f6160f634a483e8548c499a2cda7fa

ДОДАТОК А. ДОВІДКА ПРО УЧАСТЬ У ВИКОНАННІ НАУКОВО-ДОСЛІДНИХ РОБІТ ЗУНУ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАУКОВО-ДОСЛІДНА ЧАСТИНА
46009, Україна, м. Тернопіль, вул. Львівська, буд. 5-а, тел. (0352) 51-75-52

№ 21/31-2024 «16» 04 2024 р.

ДОВІДКА

Видана **ЛЯЩИНСЬКОМУ Петру Борисовичу** про участь у виконанні науково-дослідних робіт. Зокрема, у 2017 році – держбюджетної розробки «Гібридна інтелектуальна інформаційна технологія діагностування передракових станів молочної залози на основі аналізу зображень» (державний реєстраційний номер 0116U002500) в лабораторії з проблем інформаційних технологій на посаді лаборанта (Наказ від 12.12.2016 р. №329); у 2019 році – госпдоговірної науково-дослідної роботи «Нейромережеві методи і засоби класифікації зображень ауто- та ксеногенних тканин» (державний реєстраційний номер 0119U103227) на посаді лаборанта (Наказ від 06.12.2019 р. №356); у 2023 році – госпдоговірної НДР «Високопродуктивна комп'ютерна система опрацювання біомедичних зображень» (державний реєстраційний номер 0122U201124) на посаді молодшого наукового співробітника (Наказ від 20.12.2022 р. №412). Також у 2021-2024 роках Лящинський П.Б. бере участь у виконанні науково-дослідної роботи, яка виконується професорсько-викладацьким персоналом, докторантами, аспірантами та здобувачами наукового ступеня кафедри комп'ютерної інженерії в межах основного робочого часу «Розробка методів, алгоритмів та програмних засобів синтезу та класифікації біомедичних зображень» (державний реєстраційний номер 0121U108937) в розробці розділу «Синтез біомедичних зображень на основі GAN-мереж».

Начальник науково-дослідної частини Віта СЕМАНЮК

ДОДАТОК Б. ДОВІДКА ПРО УЧАСТЬ У ВИКОНАННІ НАУКОВО-ДОСЛІДНИХ РОБІТ НУЛП

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
Національного університету
«Львівська політехніка»
д.т.н., проф. Іван ДЕМИДОВ
«__» _____ 2024 р.

АКТ
про використання результатів дисертаційної роботи
Лящинського Петра Борисовича
«Синтез біомедичних зображень на основі глибоких нейронних мереж»,
представленої на здобуття наукового ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки»,
при виконанні науково-дослідної роботи за темою:
«Методи та засоби нейронечіткого управління групою мобільних робото-технічних платформ»

Комісія у складі – начальника НДЧ д.т.н., ст. досл. Романа НЕБЕСНОГО та членів: зав. відділу науково-організаційного супроводу наукових досліджень к.т.н. Галини ЛАЗЬКО, завідувача кафедри автоматизованих систем управління д.т.н., проф. Василя ТЕСЛЮКА та заст. начальника планово-фінансового відділу Ірини ФАСТ цим актом підтверджують, що результати дисертаційної роботи здобувача наукового ступеня доктора філософії Лящинського Петра Борисовича на здобуття наукового ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки» використані при виконанні науково-дослідної роботи, яка виконувалась за рахунок коштів загального фонду державного бюджету за темою: «Методи та засоби нейронечіткого управління групою мобільних робототехнічних платформ» (номер державної реєстрації 0123U101688).

Зокрема Петром ЛЯЩИНСЬКИМ розроблено метод автоматичного синтезу архітектур згорткових нейронних мереж для класифікації зображень (Розділ 2. Метод автоматичного синтезу архітектур загорткових нейроних мереж), який за рахунок використання фаз мікропошуку та макропошуку забезпечує створення нейромереж з підвищеною точністю класифікації зображень.

Голова комісії:
Начальник науково-дослідної частини, д.т.н, ст. досл. — Роман НЕБЕСНИЙ

Члени комісії:

Зав.відділу науково-організаційного супроводу наукових досліджень — Галина ЛАЗЬКО

В.о. заступника начальника планово-фінансового відділу — Ірина ФАСТ

Завідувач кафедри автоматизованих систем управління, д.т.н, проф. — Василь ТЕСЛЮК

ДОДАТОК В. АКТ ВПРОВАДЖЕННЯ – ІНСТИТУТ БІОМЕДИЧНИХ ТЕХНОЛОГІЙ

ТОВ "ІНСТИТУТ БІОМЕДИЧНИХ ТЕХНОЛОГІЙ"
46001, м. Тернопіль, вул. Січових Стрільців, 8А,
інд. код 35578106, р/р 26000369000700 в АТ «УкрСиббанк». м. Харків,
МФО 351005, ІПН 355781019181
тел./факс (0352) 43-49-29

АКТ

впровадження результатів дисертації

Лящинського Петра Борисовича

Цим підтверджуємо, що результати дисертаційної роботи «Синтез біомедичних зображень на основі глибоких нейронних мереж» аспіранта кафедри комп'ютерної інженерії П. Б. Лящинського, виконаної у Західноукраїнському національному університеті, впроваджено у ТзОВ «Інститут біомедичних технологій», де використовуються для автоматичного генерування та класифікації ауто- та ксеногенних тканин.

Аспірант брав активну участь у двох госпдоговірних науково-дослідних роботах «Нейромережеві методи і засоби класифікації зображень ауто- та ксеногенних тканин» (2019 р., державний реєстраційний номер 0119U103227), «Високопродуктивна комп'ютерна система опрацювання біомедичних зображень» (2023 р., державний реєстраційний номер 0122U201124).

Аспірантом розроблено метод автоматичного пошуку архітектур нейронних мереж для класифікації біомедичних зображень, що дало можливість підвищити точність класифікації біомедичних зображень. Крім цього розроблено метод автоматичного пошуку архітектур нейронних мереж для синтезу біомедичних зображень, що дало можливість підвищити якість синтезованих зображень.

Також розроблено метод генерування та класифікації біомедичних зображень, дав можливість розширити навчальну вибірку біомедичних зображень і підвищити точність їх класифікації.

Використання розроблених методів дозволило згенерувати синтетичну базу зображень На основі розробленого програмного засобу стало можливим синтезувати та класифікувати ауто- та ксеногенні тканини.

Цей акт не є підставою для фінансових зобов'язань.

Відповідальний виконавець
госпдоговірних науково-дослідних робіт,
к.т.н., доцент кафедри комп'ютерної інженерії
Західноукраїнського національного університету Г. М. Мельник

Директор ТзОВ «Інститут біомедичних технологій»,
заслужений діяч науки і техніки України,
лауреат Державної премії в галузі науки і техніки,
д.м.н., професор В.В. Бігуняк
20.05.2024р.

ДОДАТОК Г. АКТ ВПРОВАДЖЕННЯ – НАВЧАЛЬНИЙ ПРОЦЕС ЗУНУ

ЗАТВЕРДЖУЮ
Проректор
з науково-педагогічної роботи
Західноукраїнського національного університету
канд. економічних наук, доцент
_____ Віктор ОСТРОВЕРХОВ
«__15__» ____06____ 20_2_ р

АКТ
впровадження результатів дисертації

Ми, що нижче підписалися, комісія у складі декана факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету (ЗУНУ), кандидата технічних наук, доцента Ігоря ЯКИМЕНКА, завідувача кафедри комп'ютерної інженерії, кандидата технічних наук, доцента Лесі ДУБЧАК, доцента кафедри комп'ютерної інженерії, кандидата технічних наук Григорія МЕЛЬНИКА, склали цей акт про те, що результати, отримані аспірантом Петром ЛЯЩИНСЬКИМ під час виконання ним дисертації, впроваджено на кафедрі комп'ютерної інженерії факультету комп'ютерних інформаційних технологій ЗУНУ, де вони використовуються у навчальному процесі та науково-дослідній роботі.

Дисертаційна робота Петра ЛЯЩИНСЬКОГО виконувалася в межах таких науково-дослідних робіт ЗУНУ: у 2017 році - держбюджетної розробки «Гібридна інтелектуальна інформаційна технологія діагностування передракових станів молочної залози на основі аналізу зображень» (державний реєстраційний номер 0116U002500); у 2019 році - госпдоговірної науково-дослідної роботи «Нейромережеві методи і засоби класифікації зображень ауто- та ксеногенних тканин» (державний реєстраційний номер 0119U103227); у 2023 році - госпдоговірної НДР «Високопродуктивна комп'ютерна система опрацювання біомедичних зображень» (державний реєстраційний номер 0122U201124). Також у 2021-2024 роках Петро ЛЯЩИНСЬКИЙ брав участь у виконанні науково-дослідної роботи, яка виконується професорсько- викладацьким персоналом, докторантами, аспірантами та здобувачами наукового ступеня кафедри комп'ютерної інженерії в межах основного робочого часу «Розробка методів, алгоритмів та програмних засобів синтезу та класифікації біомедичних зображень» (державний реєстраційний номер 0121U108937).

Петро ЛЯЩИНСЬКИЙ розробив та дослідив нові методи: метод автоматичного синтезу архітектур згорткових нейронних мереж для класифікації біомедичних зображень; метод автоматичного синтезу архітектур генеративно-змагальних нейронних мереж для задач генерування біомедичних зображень; метод генерування та класифікації біомедичних зображень, які забезпечили розширення та доповнення навчальної вибірки біомедичних зображень для навчання згорткових нейронних мереж. Також розроблено модель опису архітектур нейронних мереж, яка забезпечила формалізацію представлення згорткових і генеративно-змагальних нейронних мереж.

Використання розроблених засобів забезпечує автоматичний синтез архітектур згорткових і генеративно-змагальних нейронних для задач класифікації та генерування біомедичних зображень. Розроблені засоби генерування та класифікації біомедичних зображень розширюють та доповнюють навчальні набори біомедичних зображень для навчання згорткових нейронних мереж.

Розроблені Петром ЛЯЩИНСЬКИМ методи забезпечують автоматичний синтез архітектур згорткових і генеративно-змагальних нейронних для задач класифікації та генерування біомедичних зображень. Розроблені засоби генерування та класифікації біомедичних зображень розширюють та доповнюють навчальні набори біомедичних зображень для навчання згорткових нейронних мереж.

Результати наукових досліджень, викладені у дисертаційній роботі, використані при підготовці дисциплін «Методи розпізнавання зображень і комп'ютерний зір», «Теоретичні основи штучного інтелекту», «Технології глибокого машинного навчання», «Дослідження комп'ютерних систем штучного інтелекту» на кафедрі комп'ютерної інженерії Західноукраїнського національного університету.

Декан факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету, кандидат технічних наук, доцент — Ігор ЯКИМЕНКО

Завідувач кафедри комп'ютерної інженерії, кандидат технічних наук, доцент — Леся ДУБЧАК

Доцент кафедри комп'ютерної інженерії, кандидат технічних наук, відповідальний виконавець науково-дослідних тем — Григорій МЕЛЬНИК

ДОДАТОК Д. ПРИКЛАДИ ФАЙЛІВ ДЛЯ РОЗГОРТАННЯ ПРОГРАМНОГО ЗАСОБУ НА ХМАРНІЙ ІНФРАСТРУКТУРІ

```yaml
version: '3.8'

services:
  nestjs-app:
    build:
      context: ./api
      dockerfile: APIDockerfile
    ports:
      - "3000:3000"
    depends_on:
      - mongodb
      - rabbitmq
    environment:
      MONGO_URI: "mongodb://mongodb:27017/nestjs"
      RABBITMQ_URL: "amqp://rabbitmq"

  python-app:
    build:
      context: ./python-worker
      dockerfile: WorkerDockerfile
    ports:
      - "5000:5000"
    depends_on:
      - mongodb
      - rabbitmq
    environment:
      MONGO_URI: "mongodb://mongodb:27017/biomedai"
      RABBITMQ_URL: "amqp://rabbitmq"

  mongodb:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db

  rabbitmq:
    image: "rabbitmq:3-management"
    ports:
      - "5672:5672"   # AMQP protocol port
      - "15672:15672" # RabbitMQ management dashboard
    volumes:
      - rabbitmq-data:/var/lib/rabbitmq

volumes:
  mongo-data:
  rabbitmq-data:
```

```
# Define default make target
.PHONY: all
all: build_frontend run

# Build Docker containers
.PHONY: build_docker
build_docker:
    docker-compose build

# Install frontend dependencies
.PHONY: install_frontend
install_frontend:
    cd frontend && npm install

# Build frontend static files
.PHONY: build_frontend
build_frontend: install_frontend
    cd frontend && npm run build

# Run the entire application
.PHONY: run
run:
    docker-compose up -d

# Stop the application
.PHONY: stop
stop:
    docker-compose down

# Clean up the environment
.PHONY: clean
clean:
    docker-compose down -v # This also removes the volumes
    cd frontend && rm -rf node_modules

# Rebuild everything from scratch
.PHONY: rebuild
rebuild: clean all

# Display help for each command
.PHONY: help
help:
    @echo "Usage: make [target] ..."
    @echo "Targets:"
    @echo "  all              Builds and runs the whole application"
    @echo "  build_docker     Builds Docker containers"
    @echo "  install_frontend Installs frontend dependencies"
    @echo "  build_frontend   Builds frontend static files"
    @echo "  run              Runs all services (in background)"
    @echo "  stop             Stops all services"
    @echo "  clean            Cleans up containers, networks and volumes"
    @echo "  rebuild          Rebuilds all containers and frontend from scratch"
    @echo "  help             Displays this help"
```

```
provider "aws" {
  region = "us-west-2"
}

# ECS Cluster for NestJS Application
resource "aws_ecs_cluster" "nestjs_cluster" {
  name = "nestjs-cluster"
}

# ECS Cluster for Python Application
resource "aws_ecs_cluster" "python_cluster" {
  name = "python-cluster"
}

# Task Definition for NestJS
resource "aws_ecs_task_definition" "nestjs_task" {
  family                   = "nestjs-application"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "256"
  memory                   = "512"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([
    {
      name      = "nestjs",
      image     = "account-id.dkr.ecr.region.amazonaws.com/nestjs-repository:latest",
      cpu       = 256,
      memory    = 512,
      essential = true,
      portMappings = [
        {
          containerPort = 3000,
          hostPort      = 3000
        }
      ]
    }
  ])
}

# Task Definition for Python
resource "aws_ecs_task_definition" "python_task" {
  family                   = "python-application"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "256"
  memory                   = "512"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([
    {
      name      = "python",
      image     = "account-id.dkr.ecr.region.amazonaws.com/python-repository:latest",
      cpu       = 256,
      memory    = 512,
      essential = true,
      portMappings = [
        {
          containerPort = 5000,
          hostPort      = 5000
        }
      ]
    }
  ])
}
```

```yaml
name: Deploy Frontend to AWS S3 and Invalidate CloudFront

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'

    - name: Install dependencies
      run: npm install

    - name: Build
      run: npm run build

    - name: Deploy to S3
      uses: jakejarvis/s3-sync-action@v0.5.1
      with:
        args: --acl public-read --follow-symlinks --delete
      env:
        AWS_S3_BUCKET: frontend
        AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        AWS_REGION: 'us-west-2'
        SOURCE_DIR: 'build'
```

```yaml
name: Deploy Applications to ECS

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        include:
          - app_name: "nestjs"
            ecr_repository: ${{ secrets.ECR_REPOSITORY_NESTJS }}
            ecs_service_name: ${{ secrets.ECS_SERVICE_NAME_NESTJS }}
            dockerfile_path: "./api/APIDockerfile"
            context_path: "./api"
          - app_name: "python"
            ecr_repository: ${{ secrets.ECR_REPOSITORY_PYTHON }}
            ecs_service_name: ${{ secrets.ECS_SERVICE_NAME_PYTHON }}
            dockerfile_path: "./python-worker/WorkerDockerfile"
            context_path: "./python-app"

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v1
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ secrets.AWS_REGION }}

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build, tag, and push Docker image to Amazon ECR
      run: |
        docker build -t ${{ matrix.ecr_repository }}:${{ github.sha }} -f ${{
matrix.dockerfile_path }} ${{ matrix.context_path }}
        docker push ${{ matrix.ecr_repository }}:${{ github.sha }}

    - name: Fill in the new image ID in the ECS task definition
      id: task-def
      uses: aws-actions/amazon-ecs-render-task-definition@v1
      with:
        task-definition: ${{ matrix.ecr_repository }}-task-def.json
        container-name: ${{ matrix.app_name }}
        image: ${{ matrix.ecr_repository }}:${{ github.sha }}

    - name: Deploy ECS task definition
      uses: aws-actions/amazon-ecs-deploy-task-definition@v1
      with:
        service: ${{ matrix.ecs_service_name }}
        cluster: ${{ secrets.ECS_CLUSTER_NAME }}
        task-definition: ${{ steps.task-def.outputs.task-definition }}
```

ДОДАТОК Е. ПРИКЛАД PYTHON КОДУ ДЛЯ ПОБУДОВИ ROC КРИВИХ

```
from itertools import cycle

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc


def roc_from_predictions(y_score, y, title='Some extension of Receiver operating characteristic to multi-class',
                         labels=None, lw=2):
    """
    This function builds ROC curve from model predictions.
        - y_score - model class predictions (as numpy array)
        - y - one-hot encoded ground truth (as numpy array)
        - labels - class labels
        - lw - plot's line width
    """

    n_classes = y.shape[1]

    _roc(n_classes, y_score, y, title, labels, lw)


def roc_from_keras_model(compiled_model, x, y,
                         title='Some extension of Receiver operating characteristic to multi-class', labels=None, lw=2):
    """
    This function builds ROC curve from the Keras classification model.
        - compiled_model is a keras model
        - x - dataset
        - y - one-hot encoded ground truth (as numpy array)
        - labels - class labels
```

```
        - lw - plot's line width
    """

    n_classes = y.shape[1]
    y_score = compiled_model.predict(x)

    _roc(n_classes, y_score, y, title, labels, lw)


def _roc(n_classes: int, y_score, y, title: str, labels=None,
lw=2):
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y.ravel(),
y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    # Plot all ROC curves
    plt.figure(1)

    _plot(fpr, tpr, roc_auc, n_classes, labels, title, lw)


def _plot(fpr, tpr, roc_auc, n_classes, labels, title, lw):
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)
    colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
```

```
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                       ''.format(labels[i] if labels is not None
else i, roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()
```

ДОДАТОК Ж. ПРИКЛАД ІМПЛЕМЕНТАЦІЇ CRUD СЕРВІСУ НА МОВІ ПРОГРАМУВАННЯ TYPESCRIPT

```
import {
  Model,
  Document,
  FilterQuery,
  Types,
  HydratedDocument,
  UpdateQuery as MongooseUpdateQuery,
  AnyKeys as MongooseAnyKeys,
  QueryWithHelpers,
  AggregateOptions,
  isValidObjectId,
} from 'mongoose';

import {
  Pagination,
  CrudQueryOptions,
  PaginationResult,
  WithError,
  WithoutError,
  FindOneOptions,
  FilterOrId,
  FindManyOptions,
  IsExistsOptions,
  UpdateOptions,
  BaseEntity,
  CrudSignature,
  DEFAULT_OPTIONS,
} from '../misc/types';
import { throwErrorCheck } from '../misc/helpers';
import { Inject, Type } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { ClsService } from 'nestjs-cls';
```

```
export function CrudService<T extends Document>(entity: Type<BaseEntity>): Type<CrudSignature<T>> {
  class CrudMixin implements CrudSignature<T> {
    @InjectModel(entity.name) private readonly databaseModel!: Model<T>;
    @Inject(ClsService) private readonly cls!: ClsService;

    async findOne(filter: FilterOrId<T>, options?: FindOneOptions & WithError): Promise<T>;
    async findOne(filter: FilterOrId<T>, options?: FindOneOptions & WithoutError): Promise<T | null>;
    async findOne(filter: FilterOrId<T>, options: FindOneOptions = DEFAULT_OPTIONS): Promise<T | null> {
      const session = this.cls.get('mongoSession');

      let mongooseQuery: QueryWithHelpers<HydratedDocument<T> | null, HydratedDocument<T>>;

      if (isValidObjectId(filter as Types.ObjectId)) {
        mongooseQuery = this.databaseModel.findOne({ _id: filter as Types.ObjectId }, null, options).session(session);
      } else {
        mongooseQuery = this.databaseModel.findOne(filter as FilterQuery<T>, null, options).session(session);
      }

      const queryResult = await mongooseQuery.exec();

      throwErrorCheck(queryResult, options);

      return queryResult;
    }

    async findMany(
      filter?: FilterQuery<T>,
```

```
      options?: FindManyOptions & {
        pagination: Pagination;
      },
    ): Promise<PaginationResult<T>>;
    async findMany(filter?: FilterQuery<T>, options?:
FindManyOptions & WithError): Promise<T[]>;
    async findMany(filter?: FilterQuery<T>, options?:
FindManyOptions & WithoutError): Promise<T[]>;
    async findMany(
      filter?: FilterQuery<T>,
      options: FindManyOptions = DEFAULT_OPTIONS,
    ): Promise<PaginationResult<T> | T[] | []> {
      const session = this.cls.get('mongoSession');

      const mongooseQuery = this.databaseModel.find(filter ?? {},
null, options);

      mongooseQuery.session(session);

      if (options.pagination) {
        const { page, limit } = options.pagination;

        if (page && limit) {
          mongooseQuery.skip((page - 1) * (limit || 0));
          mongooseQuery.limit(limit || 0);
        }

        const queryResult = await mongooseQuery;
        const total = await
this.databaseModel.countDocuments(filter ?? {});

        throwErrorCheck(queryResult, options);

        return {
          data: queryResult,
          total,
```

```
          totalPages: limit && limit !== 0 ? Math.ceil(total /
limit) : 1,
          currentPage: page ? +page : 1,
        };
      }

      const queryResult = await mongooseQuery;

      throwErrorCheck(queryResult, options);

      return queryResult;
    }

    async isExists(filter: FilterQuery<T>, options?:
IsExistsOptions & WithError): Promise<true>;
    async isExists(filter: FilterQuery<T>, options?:
IsExistsOptions & WithoutError): Promise<boolean>;
    async isExists(filter: FilterQuery<T>, options:
IsExistsOptions = DEFAULT_OPTIONS): Promise<true | boolean> {
      const isDocumentExists = await
this.databaseModel.exists(filter);
      throwErrorCheck(isDocumentExists, options);

      return Boolean(isDocumentExists);
    }

    async count(filter: FilterQuery<T>): Promise<number> {
      return this.databaseModel.count(filter);
    }

    async create(data: MongooseAnyKeys<T>): Promise<T>;
    async create(data: MongooseAnyKeys<T>[]): Promise<T[]>;
    async create(data: MongooseAnyKeys<T>[] | MongooseAnyKeys<T>):
Promise<T[] | T> {
      const session = this.cls.get('mongoSession');
```

```
      const [doc] = await this.databaseModel.create(Array.isArray(data) ? data : [data], { session });

      return doc;
    }

    async updateOne(
      filter: FilterOrId<T>,
      data: MongooseUpdateQuery<T>,
      options?: UpdateOptions & WithError,
    ): Promise<T>;
    async updateOne(
      filter: FilterOrId<T>,
      data: MongooseUpdateQuery<T>,
      options?: UpdateOptions & WithoutError,
    ): Promise<T | null>;
    async updateOne(
      filter: FilterOrId<T>,
      data: MongooseUpdateQuery<T>,
      options: UpdateOptions = DEFAULT_OPTIONS,
    ): Promise<T | null> {
      let updatedDocument: T | null;

      const session = this.cls.get('mongoSession');

      if (isValidObjectId(filter as Types.ObjectId)) {
        updatedDocument = await this.databaseModel.findOneAndUpdate(
          { _id: filter as Types.ObjectId },
          data,
          Object.assign(options ?? {}, { new: true, session }),
        );
      } else {
        updatedDocument = await this.databaseModel.findOneAndUpdate(
```

```
        filter as FilterQuery<T>,
        data,
        Object.assign(options ?? {}, { new: true, session }),
      );
    }

    throwErrorCheck(updatedDocument, options);

    return updatedDocument;
  }

  async updateMany(
    filter: FilterQuery<T>,
    data: MongooseUpdateQuery<T>,
    options?: UpdateOptions & WithError,
  ): Promise<T[]>;
  async updateMany(
    filter: FilterQuery<T>,
    data: MongooseUpdateQuery<T>,
    options?: UpdateOptions & WithoutError,
  ): Promise<T[] | []>;
  async updateMany(
    filter: FilterQuery<T>,
    data: MongooseUpdateQuery<T>,
    options: UpdateOptions = DEFAULT_OPTIONS,
  ): Promise<T[] | []> {
    const session = this.cls.get('mongoSession');

    const targetDocuments = await this.databaseModel.find(filter).session(session).exec();

    throwErrorCheck(targetDocuments, options);

    if (!targetDocuments.length) {
      return [];
    }
```

```
      await       this.databaseModel.updateMany(filter,       data,
Object.assign(options ?? {}, { session }));

      const query = this.databaseModel
        .find({
          _id: { $in: targetDocuments.map((d) => d._id) },
        })
        .session(session);

      return query.exec();
    }

    async      deleteOne(filter:      FilterOrId<T>,      options?:
CrudQueryOptions & WithError): Promise<T>;
    async      deleteOne(filter:      FilterOrId<T>,      options?:
CrudQueryOptions & WithoutError): Promise<T | null>;
    async      deleteOne(filter:      FilterOrId<T>,      options:
CrudQueryOptions = DEFAULT_OPTIONS): Promise<T | null> {
      let targetDocument: T | null;
      const session = this.cls.get('mongoSession');

      if (isValidObjectId(filter as Types.ObjectId)) {
        targetDocument  =  await  this.databaseModel.findOne({  _id:
filter as Types.ObjectId }, undefined, { session });
      } else {
        targetDocument  =  await  this.databaseModel.findOne(filter
as FilterQuery<T>, undefined, { session });
      }

      throwErrorCheck(targetDocument, options);

      if                   (targetDocument)                   await
this.databaseModel.findOneAndDelete({ _id: targetDocument._id }, {
session });
```

```
      return targetDocument;
    }

    async deleteMany(filter: FilterQuery<T>, options?:
CrudQueryOptions & WithError): Promise<T[]>;
    async deleteMany(filter: FilterQuery<T>, options?:
CrudQueryOptions & WithoutError): Promise<T[] | []>;
    async deleteMany(filter: FilterQuery<T>, options:
CrudQueryOptions = DEFAULT_OPTIONS): Promise<T[] | []> {
      const session = this.cls.get('mongoSession');

      const targetDocuments = await
this.databaseModel.find(filter).session(session).exec();
      throwErrorCheck(targetDocuments, options);

      await this.databaseModel.deleteMany(filter);

      return targetDocuments;
    }

    async aggregation<ReturnedType>(pipelines: any[], options?:
AggregateOptions): Promise<ReturnedType[]> {
      const session = this.cls.get('mongoSession');

      const aggQuery = this.databaseModel.aggregate(pipelines,
options).session(session);

      return aggQuery.exec();
    }
  }

  return CrudMixin;
}

import {
  QueryOptions as MongooseQueryOptions,
```

```
  FilterQuery,
  Types,
  Document,
  AnyKeys as MongooseAnyKeys,
  UpdateQuery as MongooseUpdateQuery,
  AggregateOptions,
} from 'mongoose';

export type Pagination = {
  page?: number;
  limit?: number;
};

export type PaginationResult<T> = {
  data: T[];
  total: number;
  totalPages: number;
  currentPage: number;
};

export class BaseEntity {}

export type CrudQueryOptions = {
  throwError?: boolean;
  errorMessage?: string;
  pagination?: Pagination;
};

export type WithError = {
  throwError: true;
};

export type WithoutError = {
  throwError: false;
};
```

```
export type IsExistsOptions = Pick<CrudQueryOptions, 'errorMessage' | 'throwError'>;
export type FilterOrId<T> = FilterQuery<T> | Types.ObjectId;
export type FindOneOptions = Omit<CrudQueryOptions, 'pagination'> & MongooseQueryOptions;
export type FindManyOptions = CrudQueryOptions & MongooseQueryOptions;
export type UpdateOptions = Omit<CrudQueryOptions, 'pagination'> & MongooseQueryOptions;

export const DEFAULT_OPTIONS: CrudQueryOptions = {
  throwError: false,
};

export interface CrudSignature<T extends Document> {
  findOne(filter: FilterOrId<T>, options?: FindOneOptions & WithError): Promise<T>;
  findOne(filter: FilterOrId<T>, options?: FindOneOptions & WithoutError): Promise<T | null>;
  findOne(filter: FilterOrId<T>, options: WithoutError): Promise<T | null>;

  findMany(
    filter?: FilterQuery<T>,
    options?: FindManyOptions & {
      pagination: Pagination;
    }
  ): Promise<PaginationResult<T>>;
  findMany(filter?: FilterQuery<T>, options?: FindManyOptions & WithError): Promise<T[]>;
  findMany(filter?: FilterQuery<T>, options?: FindManyOptions & WithoutError): Promise<T[]>;

  isExists(filter: FilterQuery<T>, options?: IsExistsOptions & WithError): Promise<true>;
```

```
  isExists(filter: FilterQuery<T>, options?: IsExistsOptions & WithoutError): Promise<boolean>;
  isExists(filter: FilterQuery<T>, options: WithoutError): Promise<true | boolean>;

  count(filter: FilterQuery<T>): Promise<number>;

  create(data: MongooseAnyKeys<T>): Promise<T>;
  create(data: MongooseAnyKeys<T>[]): Promise<T[]>;
  create(data: MongooseAnyKeys<T>[] | MongooseAnyKeys<T>): Promise<T[] | T>;

  updateOne(filter: FilterOrId<T>, data: MongooseUpdateQuery<T>, options?: UpdateOptions & WithError): Promise<T>;
  updateOne(
    filter: FilterOrId<T>,
    data: MongooseUpdateQuery<T>,
    options?: UpdateOptions & WithoutError
  ): Promise<T | null>;
  updateOne(filter: FilterOrId<T>, data: MongooseUpdateQuery<T>, options: WithoutError): Promise<T | null>;

  updateMany(filter: FilterQuery<T>, data: MongooseUpdateQuery<T>, options?: UpdateOptions & WithError): Promise<T[]>;
  updateMany(
    filter: FilterQuery<T>,
    data: MongooseUpdateQuery<T>,
    options?: UpdateOptions & WithoutError
  ): Promise<T[] | []>;
  updateMany(filter: FilterQuery<T>, data: MongooseUpdateQuery<T>, options: WithoutError): Promise<T[] | []>;

  deleteOne(filter: FilterOrId<T>, options?: CrudQueryOptions & WithError): Promise<T>;
  deleteOne(filter: FilterOrId<T>, options?: CrudQueryOptions & WithoutError): Promise<T | null>;
```

```
  deleteOne(filter: FilterOrId<T>, options: WithoutError):
Promise<T | null>;

  deleteMany(filter: FilterQuery<T>, options?: CrudQueryOptions &
WithError): Promise<T[]>;
  deleteMany(filter: FilterQuery<T>, options?: CrudQueryOptions &
WithoutError): Promise<T[] | []>;
  deleteMany(filter: FilterQuery<T>, options: WithoutError):
Promise<T[] | []>;

  aggregation<ReturnedType>(pipelines: any[], options?:
AggregateOptions): Promise<ReturnedType[]>;
}
```