НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО» МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО» МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

СТАТКЕВИЧ РОМАН ВАДИМОВИЧ

УДК 004.032.26 (043.3)

ДИСЕРТАЦІЯ

МЕТОД СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ

121 Інженерія програмного забезпечення
12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

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

Науковий керівник: Гордієнко Юрій Григорович, д. ф.-м. н., професор

Київ – 2024

АННОТАЦІЯ

Статкевич Р.В. Метод сегментації зображень з використанням глибоких нейронних мереж. – Кваліфікаційна наукова робота на правах рукопису

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – Інженерія програмного забезпечення з галузі знань 12 – Інформаційні технології. – Національний Технічний Університет України «Київський Політехнічний Інститут імені Ігоря Сікорського», Київ, 2024.

Дисертаційна робота присвячена розробці та удосконаленню нейронних мереж для семантичної сегментації зображень, що базуються на архітектурі U-Net, та дозволяє покращити результати та метрики передбачень, у порівнянні з базовою архітектурою.

Аналіз зображень у контексті семантичної сегментації є однією з актуальних задач, що широко використовуються у різних галузях, таких як аналіз та діагностика медичних зображень, автономні автомобілі, тощо. Покращення методів семантичної сегментації дозволяє краще виявляти патології у людському організмі, а для систем управління автомобілем – краще розуміти навколишнє середовища та краще реагувати на виникнення небезпечних ситуацій у процесі дорожнього руху. Саме тому важливо постійно удосконалювати уже наявні методи.

Тема дисертаційної роботи входить в план наукової роботи затвердженому на кафедрі обчислювальної техніки КПІ ім. Ігоря Сікорського, що враховує розпорядження Кабінету Міністрів України від 2 грудня 2020 р. № 1556-р про схвалення Концепції розвитку штучного інтелекту в Україні.

Метою дисертації було покращення існуючих засобів аналізу зображень в контексті задач сегментації зображень, що дозволять отримувати більш точні результати.

Для досягнення цієї мети, було поставлено та вирішено наступні завдання:

- Проведено огляд та описано особливості основних архітектур нейронних мереж для аналізу зображень в контектсі задач класифікації та сегментації;

- У деталях розглянуто сімейство нейронних мереж U-Net;
- Запропоновано та обґрунтовано методи модифікації архітектур U-Net з використанням способу підбору коефіцієнта розширення та способу глибинних роздільних проміжних зв’язків.
- Проведено велику кількість експериментів на різних наборах даних, з використанням різних підходів та запропонованих нововведень і К-кратної перехресної перевірки для підтвердження якісних покращень результатів.
- Проведено виміри впливу запропонованого методу модифікації нейронної мережі U-Net на метрики швидкодії та пам’яті

Запропоновано спосіб підбору коефіцієнту розширення архітектури U-Net, що дозволяє регулювати глибину нейронної мережі та збільшення (чи зменшення) кількості параметрів даної архітектури. Завдяки цьому з’явилася можливість оптимізувати розмір нейронної мережі, та отримати результати, співставні з результатами базової архітектури, при 2.5 меншій кількості параметрів нейронної мережі.

Також було запропоновано спосіб глибинних роздільних проміжних зв’язків архітектури U-Net, що базується на основі глибинних роздільних згорткових шарів. Дана модифікація дозволила покращити точність сегментації при незначному збільшенні кількості параметрів. Разом з цим, ці модифікації дозволяють також покращувати результати не лише базової архітектури U-Net, але і її модифіковані версії, що було показано на прикладі Attention-UNet. Для різних наборів даних, було виявлено щонайменше один з варіантів модулів глибинних роздільних проміжних зв’язків, що дозволив покращити точність сегментації від 1% до 5%. У деяких випадках дане покращення було досягнуте за рахунок збільшення архітектури лише на 1%, що підтверджує якісні властивості даних змін.

На основі запропонованих способів, було розроблено метод модифікації нейронних мереж U-Net для задач сегментації зображень, з використанням мови програмування Python та бібліотеки Tensorflow для експериментального підтвердження доцільності даних модифікацій. Експерименти було проведено у різних доменах знань, таких як аналіз медичних зображень, а також аналіз міського

середовища. Також, запропоновані підходи були перевірені як на двовимірних зображеннях, так і тривимірних об’ємах, що підтверджує практичність застосування запропонованих у роботі способів модифікації нейронних мереж. Для експериментів використовувалися відомі набори даних, такі як UWGIT, BraTS, CityScapes, Synapse. Було також продемонстровано, що запропоновані модифікації дозволяють досягнути, а в деяких випадках, перевершити точність деяких відомих та широковживаних архітектур нейронних мереж.

Окрім того, було проведено аналіз швидкодії та використання пам’яті для запропонованих модифікацій нейронних мереж. Було встановлено, що глибші мережі, які використовують підхід з коефіцієнтом розширення, можуть працювати швидше, аніж базова архітектура, при приблизно однаковій точності сегментації.

Розроблений метод має велике практичне значення та широке поле для застосування у галузі аналізу зображень, що було експериментально підтверджено у ході досліджень.

Ключові слова: нейронні мережі, машинне навчання, сегментація зображень, U-Net, аналіз медичних зображень, аналіз міського середовища, ком’ютерний зір.

ABSTRACT

Statkevych R. V. Methods of image segmentation using deep neural networks – Qualified scientific work on the rights of the manuscript.

Dissertation for the degree of Doctor of Philosophy in the specialty 121 – Software engineering and 12 – Information Technologies – National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv 2024.

This work is dedicated to the development and improvement of the neural networks in a context of semantic segmentation, based on U-Net architecture, which allow to improve overall evaluation and performance metrics, compared to the baseline U-Net model.

The topic of the dissertation was agreed by the Department of Computer Engineering of National Technical University of Ukraine “Ihor Sikorskiy Kyiv Polytechnic Institute”, in accordance with Cabinet of Ministers Order №1556-p on concept of development of Artificial Intelligence in Ukraine.

The goal of dissertation was an improvement of existing semantic image segmentation methods, that allows to improve results an effectiveness of neural networks.

To achieve the stated goal, next tasks and problems were solved:

- Existing semantic image segmentation methods and neural network’s architectures.
- Family of U-Net neural network architectures were studied in detail.
- As part of the proposed method, expansion rate variation and depthwise separable skip connections were proposed as modifications to the baseline U-Net architecture.
- Large number of experiments were conducted on different datasets, using different approaches and proposed novel improvements. To confirm a quality nature of the change, K-fold cross-validation was performed;
- To get inference time and memory usage metrics, benchmark was conducted for the proposed improvements and modifications.

The proposed expansion rate variation method allows the developer to regulate the number of parameters for the U-Net network, enabling deeper networks with fewer

parameters. Using this approach, it is possible to optimize the size and the performance of the model and get the same evaluation results as a baseline model, with 2.5 less parameters.

Another proposal is using Depthwise Separable Skip Connections, based on Depthwise Separable Convolutions. This modification allowed to improve results of the model with relatively small size increase for a model. It is also possible to improve other U-Net-like models, as demonstrated on the Attention-UNet model. For different datasets, at least one of the suggested modifications managed to improve a result of the baseline model, with this improvement measured between 1-5%. It was also shown on the K-fold cross-validation, that these modifications could steadily outperform the baseline model. In some cases, this improvement was achieved by model with just 1% increase of a number of parameters, which proves this is a quality improvement.

Utilizing the proposed modifications, a method for image segmentation using modified U-Net architectures was developed with Python programming language and tensorflow library, to prove a feasibility of these modifications. Experiments were conducted in different knowledge domains, such as Computer Assisted Diagnostics and road environment analysis. Also, these methods were used for analysing both 2 dimensional images and 3 dimensional volumes, which proves the practicality of using these methods. For the experiments, such well-known state-of-the-art datasets were used, as University of Wisconsin Gastrointestinal (UWGIT), Cityscapes, Synapse, and Brain Tumor Segmentation (BraTS). It was also demonstrated that the proposed methods may match, and sometimes outperform some of the State-Of-The-Art models, aside from baseline U-Net.

Besides that, some performance and productivity metrics were measured for the proposed network architectures. It was noted, that a deeper networks with expansion rate approach may have better inference times than a baseline model and match its quality.

The suggested methods have a big practical value and a wide range of applications in a field of image analysis, that was proven during the conducted experiments.

Keywords: Neural Networks, Machine Learning, Image Segmentation, U-Net, Medical Image Analysis, Urban Environment Analysis, Computer Vision.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові праці в яких опубліковано основні наукові результати дисертації:

1. **Statkevych, R.,** Gordienko, Y., Stirenko, S. (2022). Improving U-Net Kidney Glomerulus Segmentation with Fine-Tuning, Dataset Randomization and Augmentations. In: Advances in Computer Science for Engineering and Education. Lecture Notes on Data Engineering and Communications Technologies, vol 134. *ISSN 2367-4520 (electronic)* | Springer, Cham. https://doi.org/10.1007/978-3-031-04812-8_42, - Scopus, Q3
2. **Statkevych, R.,** Gordienko, Y., Stirenko, S. (2023). Expansion Rate Parametrization and K-Fold Based Inference with U-Net Neural Networks for Multiclass Medical Image Segmentation. In: Artificial Intelligence and Soft Computing. Lecture Notes in Computer Science(), vol 14125. *ISSN 0302-9743*, Springer, Cham. https://doi.org/10.1007/978-3-031-42505-9_22, - Scopus, Q3
3. **Statkevych R**, Stirenko S, Gordienko Y. Human kidney tissue image segmentation by U-Net models. *ISSN 2473-2001 (Online)* | *IEEE Xplore digital library*. https://doi.org/10.1109/EUROCON52738.2021.9535599, - Scopus
4. **Statkevych R**, Gordienko Y, Stirenko S. Improving Pedestrian Detection Methods by Architecture and Hyperparameter Modification of Deep Neural Networks. In Advances in Artificial Systems for Logistics Engineering 2021 (pp. 44-53). *ISSN 2367-4512*, Springer International Publishing. https://doi.org/10.1007/978-3-030-80475-6_5, - Scopus, Q3

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADAM – Adaptive Moment Estimation.

API – Application Programming Interface.

DS – Dice Score.

DeSSCo, DSSC – Depthwise Separable Skip connections.

DeSSCoB, DSSCB – Depthwise Separable Skip Connections with Bottleneck.

DeSSCoR, DSSCR – Depthwise Separable Skip Connections with Residuals.

DeSSCoBR, DSSCBR – Depthwise Separable Skip Connections with Bottleneck and Residual.

GB – gigabyte.

IoU – intersection over union.

mIoU – mean intersection over union.

Attn-Unet – Attention U-Net.

BN – Batch Normalization.

IN – Instance Normalization.

GN – Group Normalization.

ReLU – rectified linear unit.

TTA – test time augmentation.

TPU – Tensor Processing Unit.

GPU – Graphical Processing Unit.

CPU – Computational Processing Unit.

RAM – Random access memory.

VRAM – Video random access memory.

FLOP – Floating point operations.

ViT – Visual Transformers.

WT – whole tumor

ET – enhanced tumor

TC – tumor core

МРТ – магнітно-резонансна томографія.

ВСТУП

Актуальність теми. В останні роки глибокі нейронні мережі набули великої популярності завдяки збільшенню обчислювальної потужності та наявністю зручних та доступних інструментів розробки. Це дозволило застосовувати нейронні мережі у різноманітних галузях, такі як аналіз медичних зображень, автономні автомобілі, аналіз супутникових знімків, обробки природньої мови, великі мовні нейронні мережі (англ. *Large Language Models*), генеративні мережі, тощо. Особливого поширення набули нейронні мережі для аналізу зображень та різного роду мультимедійних матеріалів у контексті семантичної сегментації зображень. Найчастіше з даною метою використовують згорткові нейронні мережі, які, не зважаючи на успіхи нейронних мереж на основі інших підходів у останні роки (наприклад, візуальних трансформерів), займають широку нішу завдяки простоті, швидкості тренування та відносно невеликим об’ємам наборів даних, на яких можливо натренувати нейронну мережу до задовільних результатів. Разом з цим, в цій сфері все ще наявне широке поле для покращення точності результатів, оптимізації використання ресурсів і пошуків нових способів застосування.

Збільшення розмірів нейронних мереж та кількості матеріальних ресурсів, що використовуються для тренування нейронних мереж, дозволило досягнути значних успіхів, однак точність результатів нейронних мереж часто обмежується архітектурними особливостями, що призводить до того, що збільшення розмірів та часу тренування не призводить до покращення результатів, або спричиняє їх погіршення, через такі явища, як «перенавчання» мережі, градієнти, що «згасають» або «вибухають», тощо. Саме тому важливими є якісні зміни у архітектурах, що дозволяють покращити точність передбачень за рахунок, наприклад, збільшення контексту охоплення у згорткових шарах (як, наприклад, було зроблено у нейронній мережі DeepLab), чи використання глибинних згорткових шарів (як у архітектурі MobileNet). За останні роки, було запропоновано велику кількість архітектур, які використовують різні структурні компоненти, що дозволяє комбінувати та інтегрувати їх у інші архітектури з метою покращення результатів передбачень цих нейронних мереж.

У даній роботі було запропоновано ряд модифікацій, що використовують глибинні (англ. *Depthwise*) згорткові шари на проміжних зв’язках архітектури нейронної мережі U-Net, а також спосіб підбору коефіцієнту розширення мережі U-Net, з метою її збільшення для покращення результатів або зменшення з метою оптимізації використання ресурсів, а також було показано доцільність даних модифікацій для задач сегментації зображень у різних доменах, таких як розпізнавання міського середовища, та аналіз медичних зображень – теми, що актуальні у поточних науково-технічних дослідженнях.

Зв’язок роботи з науковими програмами, планами, темами.

Тема дисертаційної роботи входить в план наукової роботи затвердженому на кафедрі обчислювальної техніки КПІ ім. Ігоря Сікорського, що враховує розпорядження Кабінету Міністрів України від 2 грудня 2020 р. № 1556-р про схвалення Концепції розвитку штучного інтелекту в Україні. Запропонований у дисертації метод використано у науково-дослідних проектах: "Платформа штучного інтелекту для виявлення та діагностики хвороб людини". №2020.01/0490, профінансовано Національним фондом досліджень України, “Knowledge At the Tip of Your fingers: Clinical Knowledge for Humanity” (KATY), (укр. «*Знання на кінчиках Ваших пальців: клінічні знання для людства*») № 101017453, який фінансується в рамках програми Horizon 2020 Європейського Союзу;

Мета і завдання дослідження. Метою дисертації було підвищення точності семантичної сегментації зображень за рахунок запропонованих у роботі модифікацій архітектури нейронних мереж типу "шифрувальник-дешифрувальник" на прикладі архітектури U-Net з використанням способу підбору коефіцієнта розширення та способу глибинних роздільних проміжних зв'язків, які у порівнянні з базовими архітектурами дозволяють покращити точність сегментації мультимедійних зображень різної розмірності для широкого кола практичних застосувань.

Об’єкт дослідження. – Процес сегментації зображень з використанням модифікованих нейронних мереж типу "шифрувальник-дешифрувальник" для

задач семантичної сегментації зображень.

Предмет дослідження. – Методи модифікації архітектури нейронних мереж типу "шифрувальник-дешифрувальник" для задач семантичної сегментації зображень на прикладі архітектури U-Net.

Методи дослідження. – Основними методами дослідження є теоретичний аналіз існуючих методів з метою використання їх компонентів для покращення базової нейронної мережі, а також експериментальний метод виявлення впливу запропонованих модифікацій на результати базової нейронної мережі

Для досягнення цієї мети, було поставлено та вирішено наступні завдання:

- Проведено огляд та описано особливості основних архітектур нейронних мереж для аналізу зображень в контексті задач класифікації та сегментації зображень;
- У деталях розглянуто сімейство нейронних мереж U-Net;
- Запропоновано та обґрунтовано метод модифікації нейронних мереж архітектури U-Net для сегментації зображень на основі способів підбору коефіцієнта розширення та глибинних роздільних проміжних зв’язків.
- Проведено декілька експериментів на різних наборах даних, з використанням різних підходів та запропонованих нововведень і K-кратної перехресної перевірки для підтвердження якісних покращень результатів.
- Проведено виміри впливу запропонованих модифікації нейронної мережі U-Net на метрики швидкодії та пам’яті.

Наукова новизна отриманих результатів. Вперше було запропоновано спосіб підбору коефіцієнту розширення для архітектури U-Net на основі гіперпараметрів коефіцієнта розширення R та глибини мережі δ, який, на відміну від існуючих підходів, дозволяє регулювати розміри нейронної мережі за рахунок зменшення кількості каналів у згорткових шарах та збільшення глибини архітектури, таким чином надаючи можливість отримати мережі менших розмірів, що працюють швидше, і досягають точність передбачення, співставну з базовою архітектурою, або збільшити глибину та розмір нейронної мережі в рамках обмежених ресурсів для досягнення більшої точності результатів.

Вперше було запропоновано спосіб глибинних роздільних проміжних зв’язків (англ. *Depthwise Separable Skip Connections*) архітектури U-Net на основі глибинних роздільних згорток, який на відміну від існуючих методів модифікації архітектури U-Net дозволяє збільшити точність сегментації з набагато меншим приростом кількості додаткових параметрів (від 1% до 10%), таким чином вдалося покращити точність сегментації зображення при незначному збільшенні розміру нейронної мережі.

Подальшого розвитку набули методи на основі згорткових нейронних мереж типу «шифрувальник-дешифрувальник» сімейства U-Net, що на відміну від існуючих методів мають менші розміри та меншу кількість шарів, а також мають модульну будову, таким чином даючи можливість заміняти компоненти архітектури з метою збільшення точності сегментації, та дозволяють використовувати дані нейронні мережі в умовах обмежених обчислювальних ресурсів.

Практичне значення отриманих результатів. Отримані результати дозволяють розширити арсенал засобів для роботи з нейронними мережами у різноманітних сферах. В даній роботі було розглянуто використання нейронних мереж для аналізу медичних зображень (зокрема, у контексті діагностики та виявлення пухлин), а також аналізу міського середовища в контексті автомобільного руху, що підтверджує можливості використання запропонованого методу модифікації архітектури U-Net у широкому полі практичних застосувань. В результаті проведених досліджень, було встановлено, що завдяки незначним запропонованим в роботі модифікаціям до існуючих архітектур нейронних мереж, використання запропонованого методу може приводити як до збільшення точності сегментації зображень, так і до пришвидшення швидкодії нейронної мережі, в залежності від обраних гіперпараметрів та наявних ресурсів.

Особистий внесок здобувача. Дана робота є результатом індивідуальних зусиль здобувача. В ході науково-дослідної роботи, здобувачем було 1) обрано методологію проведення дослідження, 2) обрано набори даних для проведення експериментів, 3) запропоновано способи покращення існуючих методів (спосіб

Роздільних Глибинних Проміжних Зв’язків в мережах U-Net, спосіб підбору коефіцієнту розширення архітектури U-Net), 4) платформа, апаратні та програмні засоби реалізації дослідження, 5) розроблено метод модифікації нейронних мереж U-Net для сегментації зображень з використанням запропонованих способів.

Апробація результатів дисертації. Основні результати роботи опубліковано та обговорено на міжнародних та всеукраїнских наукових конференціях, зокрема на: International Conference on Computer Science, Engineering and Education Applications 2022 Feb 21, International Conference on Artificial Intelligence and Soft Computing 2023 Jun 18, IEEE EUROCON 2021-19th International Conference on Smart Technologies 2021 Jul 6, International Conference on Computer Science, Engineering and Education Applications, 18-21 Jan 2021

Публікації. . За результатами дисертаційних досліджень опубліковано 4 наукові статі, що входять до наступних наукометричних баз даних з міжнародним індексом цитування: Scopus – 4, з них 3 – в журналах Q3.

Структура та обсяг роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел із 102 найменувань. Загальний обсяг дисертації становить 157 сторінок, з яких 106 сторінок основного тексту, 2 додатки на 27 сторінках, та містить 32 рисунки, 34 формули, 21 таблицю.

РОЗДІЛ 1
ОСОБЛИВОСТІ ВИКОРИСТАННЯ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

1.1. Згорткові нейронні мережі, їх особливості та будова.

З моменту створення перших обчислювальних машин то до сьогоднішнього дня, комп'ютерні технології відіграють велику роль в діяльності людей. Пришвидшення обчислень за допомогою спеціалізованих машин відкрило шлях до багатьох винаходів та спростило багато процесів, задля яких раніше використовувалася людська праця, а також відкрило багато не бачених раніше можливостей для реалізації концепцій, які раніше були лише теоретичним. Відомий закон Мура [1], виведений ще у 1965, висував прогноз, що кількість транзисторів буде подвоюватися кожних 2 роки, відкриваючи нові горизонти та об'єми обчислень.

Ідеї використовувати нові обчислювальні потужності для аналізу даних, прогнозування чисельних величин і створення систем прийняття рішень виникли ще у 1960-70их роках. В цьому контексті часто вживався термін «Розпізнавання закономірностей», який прослідковується у ранніх фундаментальних працях в цій сфері [2] [3].

Ранні методи машинного навчання, незважаючи на певні успіхи, могли вирішувати обмежені задачі з відносно невеликою кількістю даних. Через так зване «Прокляття багатомірності» [4] [5] , при збільшенні вимірності вхідних даних X, складність обчислень виростає на порядки, а також ускладняється процес узагальнення методу на основі машинного навчання, оскільки на порядок збільшується область визначення X, таким чином множина відомих даних стає більш розрідженою. Дана проблема вимагала інших підходів.

Однією із найважливіших концепцій, що заклала початок нейронним мережам і який використовується по сьогодні – *персептрон* (perceptron) [6] $y(x) = f\left(w^T \varphi(x)\right)$. $\phi(x)$ – це вхідні дані, або вектор ознак (найпростіше представлення –

$\phi(x)=x$, тобто вхідні дані передаються в персептрон у первинному вигляді), w^T-матриця ваг нейронної мережі, f() – функція активації. Фактично, персептрон являє собою результат матричного добутку вектору ознак та ваг, у найпростішому вигляді його можна описати як $y(x) = b + \sum_{i=1}^{N} w_i x_i$, де b – це зміщення (bias) для регуляризації мережі.

Основною проблемою персептрона (або одношарового персептрона) була неможливість працювати з нелінійною залежністю, тобто такими, які неможливо розділити множину вхідних значень за допомогою лінійої функції. *Багатошаровий персептрон (multilayer-perceptron)* дещо краще вирішує цю проблему . Він є композицією декількох персептронів $y_i(x) = f\left(w^T \cdot \varphi_i\left(y_{i-1}(x)\right) \right)$. Однак зі збільшенням кількості вхідних параметрів, через вищезгадане «прокляття багатомірності», вимоги до обчислювальних ресурсів та кількості ваг зростають кратно.

Одним із теоретичних підходів до подолання «прокляття багатомірності» була гіпотеза, що кожне входження *x* з множини *X* означає не лише конкретну точку вхідних даних, а й характеризує певну множину точок, що знаходяться у безпосередній близькості до цієї точки. Наслідком цього стало створення концепції *ядрової функції* – функції, яка характеризує подібність вхідних даних, що знаходяться в безпосередній близькості один від одного. Одним з відомих методів, де використовують ядрові функції, є *метод опорних векторів* [7]. Ядрові функції в подальшому відіграють важливу роль у сучасних нейронних мережах, зокрема, в згорткових нейронних мережах.

Із збільшенням обчислювальної потужності та розвитком графічних процесорів GPU, що значно пришвидшують обчислення, область використання нейронних мереж та об’єми даних, що їх можуть обробляти нейронні мережі, значно розширилися, включаючи аналіз та класифікацію фото та відео, розпізнавання голосу, генеративні нейронні мережі для тексту, зображень, тощо. В даній роботі розглядатимуться здебільшого методи роботи з зображеннями.

Перший широковідомий метод використання згорткових нейронних мереж для аналізу зображень (англ. *Convolutional neural-networks, CNN*) був

запропонований в архітектурі LeNet5 [8] для розпізнавання рукописних символів. Саме у цьому дослідженні було стандартизовано підхід до створення нейронних мереж.

Особливістю згорткових нейронних мереж є те, що вони використовують згадані вище «ядрові функції», за рахунок яких вихідна ознака вираховується не як залежність від усіх вхідних ознак, а як залежність від підмножини вхідних ознак, що задається розмірами ядер. Завдяки такому підходу, згорткові нейронні мережі потребують меншої кількості параметрів, що дозволило їм стати одним із основних засобів аналізу зображень.

В даній роботі розглядалася побудова модифікованих нейронних мереж для аналізу зображень. Основну увагу було надано контролю кількості параметрів в результаті застосування запропонованого методу модифікації нейронних мереж, тому важливо розглянути в деталях основні принципи побудови згорткових нейронних мереж, компоненти (шари), з яких складаються нейронні мережі, а також кількість параметрів (ваг), що додаються з кожним таким компонентом.

Кожен шар нейронної мережі приймає на вхід карту ознак Z , виконує певну операцію з нею, та повертає вихідну карту ознак Z'. Карта ознак Z в згорткових нейронних мережах зазвичай є тривимірною матрицею ознак (у контексті тривимірних зображень – чотиривимірною), що має розмірність $W \times H \times C$ (для тривимірних зображень – $W \times H \times D \times C$), які залежать від ширини та висоти (просторових вимірів) вхідного зображення, а також кількістю ознак, визначеного будовою нейронної мережі. Формально, карти ознак можна визначити як двовимірну матрицю векторів $\vec{z}_{ij}$ з розмірністю C (формула 1.1):

$$Z = \begin{pmatrix} \vec{z}_{11} & \cdots & \vec{z}_{W1} \\ \vdots & \ddots & \vdots \\ \vec{z}_{1H} & \cdots & \vec{z}_{WH} \end{pmatrix} \quad , \left|\vec{z}_{ij}\right| = C \tag{1.1}$$

У контексті аналізу зображень індекси i,j відповідають координатам конкретних пікселів чи областей зображень (якщо застосовано операції узагальнення, в результаті яких W та H зменшуються, або збільшуються).

Основними шарами згорткових нейронних мереж прийнято вважати наступні шари:

Згортковий (англ. convolutional) шар – ключовий компонет згорткових нейронних мереж. Основні властивості – просторова розмірність ядра $\mathrm{m}_{\mathrm{m}_1 \times \ldots \times \mathrm{m}_\mathrm{k}}$ та кількість нових ознак – *f*. У літературі часто згадується термін «кількість фільтрів» по відношенню до величини *f*. Математично операція з одним ядром описується формулою як перетворення $\mathrm{S} - \mathrm{S}(\mathrm{i}_1, \mathrm{i}_2 \ldots, \mathrm{i}_\mathrm{k}) = (\mathrm{K} * \mathrm{I})(\mathrm{i}_1, \mathrm{i}_2, \ldots, \mathrm{i}_\mathrm{k}) = \sum_{m_1} \sum_{m_2} \ldots \sum_{m} \mathrm{I}(\mathrm{i}_1 - \mathrm{m}_1, \mathrm{i}_2 - \mathrm{m}_2, \ldots, \mathrm{i}_\mathrm{k} - \mathrm{m}_\mathrm{k})\mathrm{K}(\mathrm{m}_1, \mathrm{m}_2, \ldots, \mathrm{m}_\mathrm{k})$ – де K – ядрова функція розмірності $\mathrm{m}_1 \times \ldots \times \mathrm{m}_\mathrm{k}$, I – k-вимірна . На практиці, згорткові шари, ядра яких мають більше ніж 4 виміри (три просторових виміри, та один вимір для ознак), зустрічаються нечасто. Згорткові шари з k=3 широко використовуються в аналізі зображень (наприклад, LeNet5 використовував ядра розрядністю 5x5), а з k=4 – в аналізі просторових даних. Для спрощення, вживають термін двовимірних та тривимірних згорток відповідно, даючи можливість розробникам задавати лише просторову розмірність ядер, тоді як кількість вхідних ознак вираховується з попередніх шарів. Кількість фільтрів регулює кількість окремих ядрових функцій, що застосовується до заданої карти ознак – наприклад, в LeNet5, шари мали 6 та 16 фільтрів відповідно. Завдяки цьому, кількість параметрів не залежить від розміру вхідних даних, що робить згорткові архітектури гнучкими, а також зменшує кількість параметрів, якщо порівнювати з тою кількістю параметрів персептрона, яка потрібна для отримання проміжної карти ознак аналогічної розмірності. Результатом операції двовимірного згортування з кількістю фільтрів f на карті ознак Z розмірністю $W \times H \times C$ буде карта ознак Z' розмірністю $W \times H \times f$.

Повна формула вирахування кількості параметрів виглядає наступним чином:

$$N_i^P = \left(1 + \prod_j^k m_j\right) \cdot f_i \tag{1.2}$$

Де N_i^P - к-сть параметрів, m_j - розмір ядра у просторовому вимірі j, f_i - кількість ядер у даному згортковому шарі, C – глибина (кількість каналів) вхідної карти ознак. Якщо вхідна карта ознак є продуктом згорткового шару, то $m_k = f_{i-1}$, тобто відповідає кількості фільтрів попереднього згорткового шару.

$$N_i^P = \left(1 + f_{i-1} \cdot \prod_j^{k-1} m_j\right) \cdot f_i \quad (1.3)$$

Повнозв'язні шари (Dense, або Fully-Connected) – шар, що є практичною реалізацією персептрона. Названий таким чином через те, що кожен канал *x* із множини *X* через матричний добуток має свій вплив на вихідні дані *x'*;

Глибинні згорткові шари (depth-wise convolutions) – варіант згорткових шарів, запропонований в [9]. На відміну від звичайних згорткових шарів, даний вид операцій виконує обчислення по кожному окремому каналу вхідної карти ознак, по одному ядру на канал, тобто $m_k = 1$. Звідси кількість параметрів обчислюється:

$$N_i^P = \left(1 + \prod_j^{k-1} m_j\right) \cdot C \quad (1.4)$$

Де C – кількість каналів вхідної карти ознак, m_j – задана розмірність ядер для j-ої розмірності.

Шари об'єднання, або узагальнення (pooling) – використовується для узагальнення груп суміжних ознак з метою зменшення розмірності карти ознак. Найпоширенішими є об'єднання з максимізацією (Max Pooling), або усереднення (Average Pooling)

Нормалізація вибірки (Batch Normalization) [10] – нормалізує значення карти ознак в рамках тренувальної вибірки – $x_i' = \frac{x_i - \mu}{\sqrt{(\sigma^2 + \epsilon)}}$ – де μ, σ- середнє значення та стандартне відхилення.

Шар активації (activation layer) – в даному шарі до кожного елементу карти ознак застосовується певна функція активації $x_i' = f(x_i)$. Поширеними функціями активації є ReLU [11], Sigmoid, тощо.

Шар випадання (drop-out) [12] - простий механізм, мета якого – боротися з «перенавчанням» у процесі тренування. Принцип роботи - з заданою ймовірністю сигнали з попереднього шару нейронної мережі замінюються нулями, таким чином в алгоритм зворотного поширення помилки вимушений шукати інші зв'язки, таким чином сприяючи збільшенню узагальнюваності нейронної мережі.

Шари розширення Upsampling – шар, обернений до шару узагальнення – він дозволяє вдвічі збільшити ширину та висоту карти ознак. Для заповнення значень

новоутворених пікселів використовуються операції інтерполяції, що вираховують значення на основі сусідніх, оригінальних пікселів. Часто використовується метод найближчого сусіда, бі-лінійний, гаусовий, тощо.

Шари оберненого згортування DeConvolution [13] (іноді, Transposed-Convolution) – шар, що використовується з аналогічною до шару розширення метою, з тією різницею, що для заповнення новоутворених пікселів замість інтерполяції використовуються ядрові функції, аналогічні до згорткових шарів. На відміну від шарів розширення, даний шар є параметризованим шаром, тобто його параметри тренуються у ході тренування нейронної мережі.

1.2. Аналіз наявних архітектур нейронних мереж для задач класифікації та сегментації зображень

Перш за все варто виокремити типи завдань, що їх вирішують нейронні мережі для роботи з зображеннями, а саме:

- Класифікація – необхідно визначити, до якого класу з обмеженого набору класів залежить зображення;
- Сегментація – визначення контурів об’єктів на зображеннях; виділяють *семантичну сегментацію* – визначення належності кожного пікселя до різних класів, та *об’єктну сегментацію* - виокремлення окремих унікальних об’єктів, і встановлення приналежності пікселю до конкретного об’єкту. В цій роботі основна увага була приділена семантичній сегментації.

Нейронні мережі для задач сегментації часто є розширенням архітектур класифікації, використовуючи нейронні мережі для класифікації в якості опорної (англ. *Backbone)* мережі, тому важливо розглядати ці архітектури в загальному контексті нейронних мереж для аналізу зображень.

З моменту розробки LeNet5, було запропоновану велику кількість різноманітних нейронних мереж, що зі зростанням обчислювальної потужності значно збільшилися у розмірах, точності та швидкодії. Особливо бурхливий розвиток був зумовлений використанням графічних карт (Graphical Processing

Unit), які за рахунок оптимізації матричних обчислень дозволяють значно швидше проводити обчислення [14]. Розглянемо декілька з них:

AlexNet [15] прийнято вважати однією з найвпливовіших архітектур згорткових нейронних мереж, як одна з перших відомих архітектур, що була натренована з використанням графічних прискорювачів, таким чином показавши потенціал такого методу тренування. Повторюючи підхід LeNet5 зі зменшенням ширини та висоти і збільшенням глибини вихідних даних згорткових шарів, AlexNet використовує в набагато більшу кількість параметрів (близько 60млн), а також додатково 3 додаткових згорткових шари з ядрами 3x3. Саме в цій архітектурі було продемонстровано ефективність використання функції активації ReLU, та шарів відкидання Dropout з коефіцієнтом відкидання 0.5. Це архітектура перемогла на конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 з класифікації 15млн зображень, що належать до 1000 різних класів, на 10% перевершивши результати найближчих конкурентів.

R-CNN [16] [17] (англ. *Region based – convolutional neural network*) – сімейство нейронних мереж, що поєднали у собі методи нейронних мереж і класичні підходи у сфері комп'ютерного зору для задач класифікації, виявлення об'єктів, а пізніше – сегментації та генерації 3D об'єктів з 2D зображення. В основі підходу – пропозиція регіонів інтересу (англ. *Regions of interest*), в яких можуть знаходитися шукані об'єкти. Також метод показав ефективність підходу перенесеного навчання (англ. *transfer learning*) – коли нейронна мережа, натренована на великому об'ємів даних, дотреновується на додаткових даних, дозволяючи добитись прийнятних результатів за умови відсутності достатньої кількості даних для тренування мережі. Ще однією практикою, популяризованою даним методом, став підхід опорної нейронної мережі (англ. *Backbone)* – коли є можливість заміняти одну чи декілька частин нейронної мережі іншою – так, оригінальна [16] робота використовувала AlexNet.

GoogLeNet [18] запропонували Inception модуль, що є комбінацією декількох згорткових шарів з ядрами 1x1, 3x3, 5x5 та шару об'єднання max-pooling з ядром 3x3, що пізніше «складаються» (англ. *concatenate*) разом та передаються в наступні

шари. В результаті отримана архітектура мала 22 шари, при цьому у 12 разів меншу кількість параметрів, ніж AlexNet, при кращій якості класифікації.

VGG16 [19] – ідеологічно схожа на AlexNet, але з більшою кількістю шарів та параметрів (було запропоновано ряд нейронних мереж з глибиною від 13 до 19 шарів)

Однією з проблем, що виникають у процесі тренування нейронних мереж, є *проблема зникаючих та вибухаючих градієнтів* [20]. Вони стають більш відчутними при подальшому збільшення глибини мережі, що не призводить до покращення результатів, і в багатьох випадках навіть приводить до їх погіршення. Для боротьби з цими проблемами було запропоновано концепцію залишкових зв'язків (residual, або skip connections) [21] у моделях ResNets, найбільш оптимальна з яких має 110 шарів.

FCN (fully convolutional network) [22] – одна з перших архітектур, що розроблялася для задачі сегментації фото. Було запропоновано ряд нейронних мереж, що базуються на AlexNet, VGG16 і GoogLeNet, в яких замінили усі сильно-зв'язані шари (що розташовувалися в найнижчих шарах) на згорткові, а останній шар з вектором коефіцієнтів класифікації замінили на згортковий шар з ядром 1x1 та кількістю ядер рівній кількості класів (21 в оригінальній статті) – це стане типовою практикою для нейронних мереж для задач сегментації зображень.

U-Net [23] – ще одна популярна архітектура для сегментації, що розроблялася в контексті аналізу медичних зображень, з міркуванням, що в задачі медичної обробки даних зазвичай набори даних мають обмежені розміри. Також ця мережа є однією з популярних архітектур, що належать до класу нейронних мереж «шифрувальник-дешифрувальник» (англ. *Encoder-Decoder*)– тобто таких, що складаються з 2ох частин, одна з яких проводить декомпозицію карт ознак (шифрування, англ. – *encoding*), а інша – реконструює (дешифрує, англ. *Decoding*) ці карти ознак до розмірності вхідних даних.

MobileNet [9] – варіація нейронної мережі, що розроблялася з метою використання на мобільних портативних платформах, які характеризуються обмеженою кількістю ресурсів та обчислювальних можливостей. В цій архітектурі

було запропоновано використання комбінації глибинних і по-точкових згорток з ядрами 1x1, що дозволило добитися гарних результатів на наборі даних ImageNet при в рази меншій кількості параметрів. Ці поєднання, названі глибинними роздільними згортками, (англ. *Depthwise-separable convolution*), і оберненими залишками (англ. Inverted-Residuals [24]), стали структурними компонентами для багатьох інших архітектур, наприклад EfficientNet [25];

DeepLab [26] – архітектура, яка запропонувала розширені згорткові (dilated convolutions) шари, що дозволяються захопити більше контексту, ніж звичайні згорткові шари, не збільшуючи кількість параметрів, за рахунок збільшення кроку всередині окремого ядра. Комбінація таких згорток (Atrous Spatial Pyramid Pooling) є структурним компонентом мережі DeepLab

ViT [27] (Visual Transformers) – на даний момент популярна архітектура, що використовує принципово інший підхід. В даному методі не використовуються згорткові шари, натомість, пропонується використання трансформерів, чий принцип роботи базується на механізмі уваги [28], що свого був розроблений для задачі обробки природньої мови (NLP). Метод показав непогані результати на багатьох еталонних наборах даних, однак вагомим недоліком є необхідність у наявності великої кількості даних, аби натренувати нейронну мережу до задовільних результатів.

Окремою категорією нейронних мереж є мережі, що працюють з тривимірними зображеннями. Однією з відомих архітектур є U-Net3D [29], що за концепцією аналогічна архітектурі U-Net, за винятком того, що у ній усі згорткові шари з двовимірними ядрами замінені на тривимірні. Ще одна архітектура – V-Net [30] – наслідує ідею U-Net-3D, з різницею в тому, що там використовуються модулі з трьох згорткових рівнів з розмірністю ядра 5x5x5.

1.3. Актуальна проблематика нейронних мереж для сегментації зображень

В цілому, увесь цикл розробки методів, що застосовують підходи машинного навчання, зводиться до наступного алгоритму, який було також застосовано і у цій

роботі – підготовка набору даних, тренування нейронної мережі, та використання нейронної мережі в режимі передбачення (inference). З кожним з етапів асоціюються певні проблеми, які в своїй більшості є фундаментальними для усіх методів, що використовують нейронні мережі - більшість з цих проблем характерна не лише для методів сегментації, але й для нейронних мереж у цілому.

Підготовка набору даних – у комерційній розробці цей етап часто є найбільш витратним з точки зору часу. Оскільки переважна кількість методів сегментації зображення використовує навчання з учителем (англ. *supervised learning*), тренування вимагає великої кількості анотованих даних – тобто таких вхідних даних, для яких відомий очікуваний результат сегментації. В залежності від сфери використання, підготовка даних вимагає рутинної ручної роботи, в деяких випадках, люди, що виконують розмітку даних, повинні мати відповідну кваліфікацію (наприклад, лікарі, що розмічують медичні зображення). Для великих об’ємів «сирих» даних (наприклад, в контексті великих мовних нейронних мереж), розмітку яких можливо автоматизувати, необхідно розробляти комплексні системні рішення для їх обробки, що вимагає додаткової інженерної експертизи. В першу чергу від якості зібраних та розмічених даних залежить якість результатів натренованих методів. Збагачення та покращення набору даних також може бути постійним процесом, наприклад, з використанням підходу навчання з підсиленням на основі людського відгуку (англ. *human feedback reinforced learning*) [31] – однак ці способи збагачення набору даних вимагають великої кількості ресурсів та часу;

Частково вирішити проблему збору даних та їх розмітки допомагають набори даних, що викладені у загальний доступ. Ці набори даних зазвичай уже підготовлені, мають високу якість розмітки даних, і доступні усім бажаючим. Однак зазвичай використання цих наборів обмежено ліцензійними умовами, що обмежують область застосування дослідницькою діяльністю та забороняють використовувати їх для тренування нейронних мереж, що застосовуватимуться у комерційній діяльності. Окрім цього, через високу якість дані набори даних недоцільно використовувати у комерційних цілях, оскільки вони є невеликими за розмірами, і отримувати нові дані подібної якості у ході експлуатації є

нетривіальною задачею, проте завдяки публічно доступним наборам, таким як ImageNet [32], сталося багато наукових проривів у сфері машинного навчання.

Тренування нейронної мережі. В процесі тренування, відповідно до його назви, відбувається процес тренування мережі, тобто модифікації її ваг з метою оптимізації функції втрат. Даний режим є затратним з точки зору ресурсів, оскільки процес зворотного поширення помилки, який є ключовим алгоритмом тренування нейронних мереж [33], вимагає обчислення градієнтів кожного параметру нейронної мережі. Завдяки використанню графічних прискорювачів GPU, що реалізовують програмний інтерфейс CUDA, даний процес вдалося пришвидшити, однак використання даних прискорювачів також є доволі дорогим.

Багато існуючих методів мають багатоетапний режим тренування, що може складатися з декількох кроків, як наприклад у підходах з використанням напівкерованого тренування (Semi-Supervised learning), таких як Masked Autoencoders [34], де тренування мережі, що базується на ViT, складається з двох етапів: спочатку на великому наборі неанотованих зображень, з яких випадковим чином видаляються частини зображення, і нейронна мережа тренується відновлювати видалені частини зображення, а потім – на меншому анотованому наборі даних. Такий підхід дозволяє спростити проблему якісного анотованого набору, однак вимагає значних обчислювальних ресурсів. Так, для попереднього тренування архітектури ViT-L з використанням методу Masked Autoencoders, було використано 128 ядер TPUv3, а сам процес тренування зайняв 31 годину [34]. Варто зазначити, що не завжди є можливість використати таку кількість ресурсів, в першу чергу через вартість хмарних обчислювальних прискорювачів.

Проблема перенавчання (англ. overfitting) - Доволі поширеною є проблема *перенавчання (overfitting)*, коли в результаті тренування нейронна мережа показує високі результати на тренувальному наборі, однак на невідомих даних ці результати незадовільні. Аби уникнути цього явища, використовуються різні методики, найпростішою з яких є виокремлення окремого валідаційного набору даних з тренувального набору, що буде містити відомі пари *(x,y)* (вхідні дані та очікуваний результат), і не буде використаним у процесі навчання, однак для нього

буде окремо пораховано функцію втрат. Тренування виконується протягом циклів, які в літературі називають епохами. В кінці кожної епохи перевіряються значення функцій втрат для тренувального та валідаційного наборів. Якщо в процесі тренування виявиться, що втрати для валідаційного набору починають зростати, тоді як для тренувального вони продовжують зменшуватися – це означає, що має місце «перенавчання» нейронної мережі на тренувальному наборі.

Однією з причин перенавчання є надлишковість нейронних мереж. Через те, що кількість параметрів значно перевищує кількість вхідних зміних, кількість цих параметрів часто є достатньою, аби точно відтворити шум (варіативність), наявний в даних тренувального набору [35]. Через розрідженість множини даних, це приводить до недостатньої узагальнюваності нейронної мережі та отримання незадовільних результатів виконання поставленої задачі на невідомих даних. Зазвичай, при виявлені факту перенавчання, навпаки, вдаються до спрощення архітектури нейронної мережі, намагаючись утримуватися певного балансу варіативності/упередження (covariance/bias). Збільшення кількості тренувальних даних є також способом запобігання перенавчанню, однак проблематика збагачення набору даних була висвітлена раніше.

Режим передбачення. - Режим передбачення (inference) потребує набагато менших ресурсів, що дозволяє експлуатувати їх на пристроях з відносно невеликою потужністю обчислювань, таких як смартфони. Існують також спеціалізовані обчислювальні пристрої (так звані Edge пристрої), що розроблялися з метою їх застосування у вбудованих системах, за умови обмеженого заряду батареї, що також обмежує їх потужність. Нейронні мережі для роботи у режимі передбачення оптимізують та мінімізують (компілюють) під відповідну архітектуру апаратного забезпечення, досягаючи ще більшої швидкодії та оптимальнішого використання ресурсів. Проте обмеженість ресурсів на таких пристроях також обмежує можливості використання нейронних мереж з великою кількістю параметрів. З огляду на це, завжди поставатиме проблема покращення результатів роботи нейронної мережі в межах наявних ресурсів.

Також одним із критеріїв роботи нейронних мереж в режимі передбачення є швидкодія – час, відведений на обробку зображення та отримання результатів. Цей критерій є визначальним для таких сфер застосування, як аналіз відеопотоку, особливо коли мова йде про системи реального часу з вимогою короткого часу відгуку на зміну обставин.

З огляду на наведену проблематику, можливо сформувати актуальні задачі, що можуть вирішуватися новими методами сегментації зображень (та методами аналізу зображень в цілому):

- Можливість тренування нейронної мережі в умовах обмеженого набору даних, що дозволить отримувати кращі результати, аніж існуючі методи за таких самих умов;
- Покращення результати методів, що вже використовуються, за рахунок незначних змін архітектури та кількості параметрів нейронної мережі;
- Пришвидшення швидкодії існуючих рішень, зменшення розмірів нейронної мережі та оптимізація використання ресурсів у рамках наявних ресурсних обмежень;
- Спрощення та пришвидшення процесу тренування нейронної мережі у порівнянні з наявними методами, що дозволяє досягнути точності цих методів за коротший проміжок часу

Відповідно до цих вимог, існує потреба у розробці нових методів, що використовують невеликі нейронні мережі в умовах незначних наявних ресурсів та часових обмежень.

1.4. Обґрунтування використання архітектури U-Net та її будова.

У даній роботі основну увагу було надано архітектурам типу «шифрувальник-дешифрувальник», типовим представником якої є архітектура U-Net [23].

З огляду на наведені вище вимоги до нейронних мереж, архітектура U-Net задовільняє більшість критеріїв, що стосуються використання з малими наборами даних, невеликих розмірів архітектури, швидкості тренувального процесу, швидкодії, тощо.

Архітектура U-Net є однією з найбільш застосовуваних архітектур нейронних мереж – Google Scholar для [23] показує близько 70 тис. цитувань [36]. Така популярність цієї архітектури зумовлена її невеликими розмірами (близько 31 млн. параметрів); успішністю у задачах аналізу медичних зображень та невеликими наборами даних, необхідними для тренування; простотою реалізації; гнучкістю до модифікацій завдяки модульній будові.

Архітектура U-Net складається з таких компонентів:

- «Шифрувальника» (Encoder) – відповідає за обробку вхідних даних та розбиття їх на карти ознак; В оригінальній архітектурі, складається з 4 модулів, які в свою чергу складаються з двох послідовних згорткових шарів з функцією активації ReLU та вдвічі більшою кількістю фільтрів, ніж у попередньому модулі, таким чином збільшується у 2 рази глибина карт ознак. На виході кожного модуля виконується операція узагальнення з використанням більшого (max-pooling) з ядрами розмірами 2x2, після кожної з яких розмірність проміжної карти ознак зменшує свою ширину та висоту у 2 рази; Часто в рамках модифікації «шифрувальник» замінюється на «опорну» нейронну мережу (наприклад, VGG16, ResNet), де проміжні карти ознак виступають в якості проміжних зв’язків до відповідних модулів «дешифрувальника»
- «Звуження» (англ. *Bottleneck*) – Складається з 2 згорткових шарів, на виході з нього виконується операція оберненого згортання (англ. *ConvTranspose*) з ядром 2x2, що збільшує ширину та висоту карти ознак у 2 рази.
- «Дешифрувальник» (англ. *Decoder*) – ця частина відповідає за відновлення карт ознак до оригінальної розмірності; Аналогічно до шифрувальника, складається з 4ьох модулів, що поєднують у собі два послідовних згорткових шари з функцією активації ReLU. Кількість ядер у цих шарах у 2 рази менша, ніж глибина карти ознак; На вхід кожного модуля подається карта ознак, що утворена шляхом об’єднання (конкатенації) карт ознак із відповідного модуля в шифрувальнику, та отриманої з попереднього модуля (або модуля «звуження») шляхом виконання операції оберненого згортання ConvTranspose2x2 (або у різних варіаціях - розширення з інтерполяцією Upsample з ядрами 2x2). На виході з

модуля, до карти ознак застосовується операція ConvTranspose/Upsample, за рахунок чого їх ширина та висота збільшується у 2 рази. На виході з останнього модуля, аналогічно з FCN [22], застосовується згортковий шар з ядрами 1x1 та глибиною k – кількістю класів класифікації;

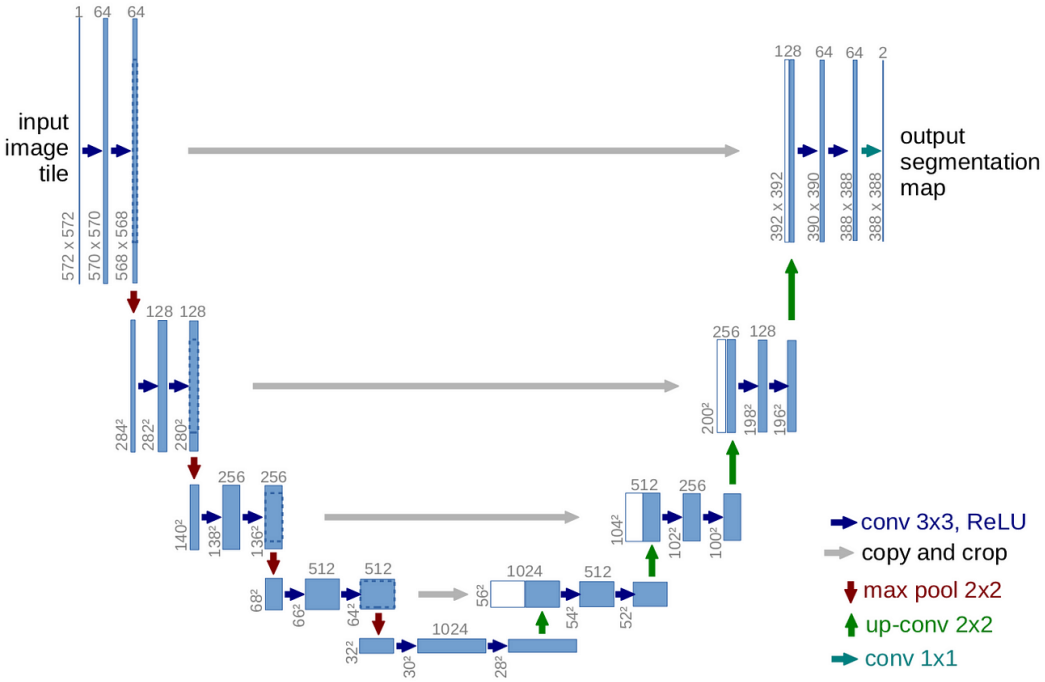

Рисунок 1.1. Візуалізація архітектури U-Net (зображення взяте з [23])

1.5. Поширені модифікації архітектури U-Net

Комбінуючи модифікації модулів «шифрування», «дешифрування», «звуження» та проміжних зв'язків, можливо суттєво покращити результати базової архітектури U-Net.

Наприклад, Attention-U-Net [37], використовуючи механізм «уваги» [28], запропонували модуль «Attention-Gate» (Рисунок 1.2), що є модифікацією проміжних зв'язків на вхід якій подаються як і дані з шифрувальника (query, q), так і дані на виході з операції «розширення» (Upsampling) в дешифрувальнику (gating signal, g), в ході тренування яких відбувається перерахування коефіцієнтів уваги, завдяки яким підсилюються сигнали у тій області зображення, де потенційно знаходиться шукана сегментація.

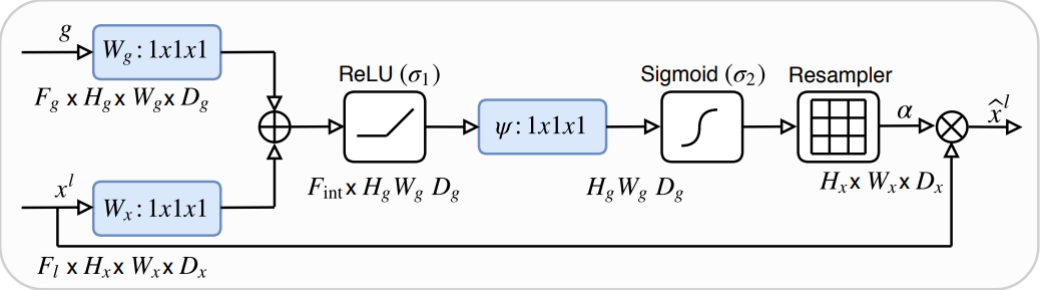

Рисунок 1.2. Схема модуля уваги [37] , що використовується у одній з запропонованих архітектур. (рисунок взято з [37])

V-Net [30] використовують для тривимірних об'ємів, з комплексними залишковими зв’язками, ядрами 5x5x5, трьома згортковими шарами у кожному модулі.

В архітектурі U-Net++ [38] використовуються наявні, попередньо треновані нейронні мережі в якості так званої «опорної» нейронної мережі (наприклад, VGG-16, ResNet, тощо) в якості шифрувальників, використовувати їх внутрішні модулі як проміжні зв’язки, каскадно просумовуючи їх між собою, даючи на вхід проміжні зв’язки з попереднього рівня та проміжні зв’язки з шифрувальника.

В архітектурі ResUNet [39], автори пропонують поелементно додавати вхідні та вихідні карти ознак за допомогою залишкових зв’язків [21] в кожному з блоків шифрувальника та дешифрувальника, таким чином уникаючи «згасання» карт ознак.

У модифікації mU-Net пропонується додавання додаткової комбінації згорткового рівня з ядрами 3x3 та з ідентичною до відповідного модуля шифрувальника кількістю фільтрів до проміжних зв’язків. На вхід цього додаткового модуля дається карта властивості, що отримана як результат поелементного віднімання від проміжних зв’язків карти ознак, що отримана в результаті операції зворотного згортування (англ. *Deconvolution*) над даними, отриманими на виході операції узагальнення з максимізацією (англ. MaxPooling), тобто розмірність цих даних буде відповідати розмірності карти ознак (загальна архітектура показана на Рисунок 1.3). Дана ідея, на думку авторів mU-Net, дозволяла компенсувати просторові втрати в результаті операції MaxPooling, в

результаті яких ширина та висота карти ознак зменшується вдвічі, шляхом виділення контурів на проміжних зв`язках.

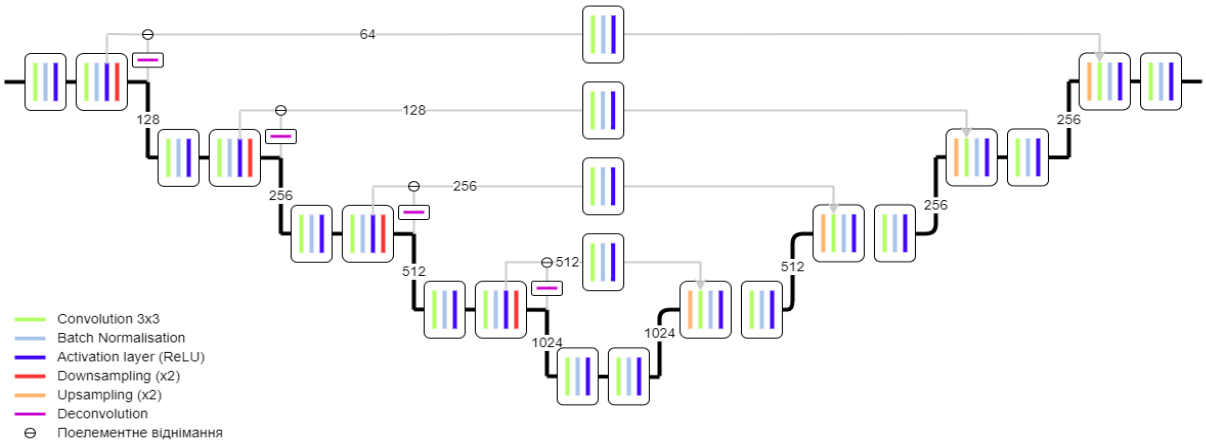

Рисунок 1.3. Візуальне представлення архітектури mU-Net

Відкриття візуальних трансфомерів [27] також зумовило винайдення модифікацій U-Net на їх основі. Так, TransU-Net [40] використали інші архітектури в якості «опорної» нейронної мережі у шифрувальнику, та застосували трансформери у модулі «звуження»; Swin-U-Net [41] повністю замінили згорткові рівні на трансформери.

В плані оптимізації кількостей параметрів, було запропоновано такі архітектури, як Squeeze-U-Net [42], яка використовує в 12 разів менше параметрів, ніж U-Net, при порівнюваній точності передбачень. Ця архітектура була побудована на базі SqueezeNet, що змогла добитися точності, порівнюваної з AlexNet, маючи у 50 разів менше параметрів [43], завдяки запропонованим fire-блокам, які являють собою комбінацію згорткових шарів з ядрами 1x1 та 3x3. Також одним з впливових методів оптимізації є блок «Стискання та збудження» (Squeeze-And-Excitation block), а також нейронна мережа SE-ResNet [44], де пропонувалося використати на проміжних зв’язках будівельного блоку архітектури ResNet операцію глобального узагальнення (Global-pooling) – тобто зведення карти ознак до розмірності 1x1xC.

1.6. Розглянуті прикладні області застосування нейронних мереж для семантичної сегментації

1.6.1. Аналіз медичних зображень

Аналіз медичних зображень став одним із популярних напрямів досліджень у сфері комп'ютерного зору, особливо в контексті таких подій, як пандемія COVID-19, яка спричинила сплеск швидкої діагностики великої кількості пацієнтів на основі різних типи візуальних даних, що надаються за допомогою комп'ютерної томографії, магнітно-резонансної томографії, мікроскопії тощо. Комп'ютерне виявлення (CADe) і комп'ютерне діагностування (CADx) дають можливість прискорити діагностику з такою ж або вищою точністю, ніж лікар-людина, даючи можливість розвантажити лікаря для виконання більш складних завдань. З огляду на велику кількість практичних застосувань, важливо постійно працювати над удосконаленням існуючих методів.

У публічному доступі наявна велика кількість наборів даних з медичними даними. Наприклад:

- набір даних Synapse [45] з МРТ знімками нижньої частини тіла;
- ініціатива HuBMAP (The Human BioMolecular Atlas Program), до якого входять набори з мікроскопічними знімками тканини нирок [46] та знімки тканин різних органів [47];
- набори даних Brain Tumor Segmentation (BraTS) [48] – надає набори даних, що складаються з трьох вимірних МРТ головного мозку пацієнтів, що мають діагностовані ракові пухлини, з багатокласовими анотаціями, що позначають різні ступені ураження тканин;
- Набори даних MedMNIST [49] – містять велику кількість двох-вимірних та трьох вимірних даних для класифікації та діагностики хвороб різних тканин та органів;

Існує велика кількість досліджень, що використовує нейронні мережі у медичних цілях.

Попередні досліджень автора цієї дисертації стосувалися методів сегментації гломерулів серед знімків тканин нирок [50] [51] [52] [53] та сегментації гастроентерологічного тракту зі знімків МРТ [54].

У роботі [55] використано модифіковану архітектуру mU-Net для виявлення пухлини печінки на рентгенівському знімку. В [56] виконували сегментацію ребер рентгенографічних знімків грудної клітини. У досліджені [57] використали нейронні мережі для діагностики пневмонії, у [58] – рак легенів. Дослідження [59] [60] присвячені задачі пошуку та виявлення меланом на шкірі. У досліджені [61] використовувалися мультимодальні дані з метою покращення діагностики діабетичної ретинопатії.

Специфікою аналізу медичних зображень є те, що доволі важко отримати велику кількість даних, а підготовка цих даних (створення анотацій) вимагає багато часу кваліфікованих спеціалістів, що мають експертизу у сфері медицини. Ще одним суттєвим обмеженням є вимоги щодо збереження персональних даних та де-ідентифікація даних, аби унеможливити встановлення особи, чиї медичні дані були використані у наборі даних. Через це, багато наборів даних є приватними, або вимагають додаткових юридичних процедур для отримання доступу до них, що в свою чергу ускладнює відтворення результатів.

1.6.2. Аналіз міського середовища.

Автоматизовані автомобілі, що керуються штучним інтелектом, є однією з дуже популярних галузей застосування нейронних мереж. В останні роки, все більше і більше виробників автомобілів інтегрують елементи штучного інтелекту у свої автомобілі. Лідерами по впровадженню даних технології є автоконцерни-гіганти, такі як Tesla, Toyota, тощо.

Оскільки застосування нейронних мереж в контексті міського середовища набуло комерційних масштабів, а їх практичне застосування вимагає великих технологічних та промислової потужності, велика кількість досліджень захищенні комерційною таємницею.

У контексті міського середовища також існує доволі багато наборів даних. Нижче наведено найбільш відомі:

- CityScapes [62] – набір містить 35 класів, отриманий з 50 міст Німеччини, є одним з наборів даних, на яких перевіряються нові запропоновані State-of-the-art архітектури;
- KITTI [63] – мультимодальний набір даних, який окрім зображень також містить дані багато мультимодальних про хмари точок, рамки об’єктів, тощо. Зображення отримані у центрі та навколо околицях німецького міста Калсруе;
- nuScenes [64] – багатомодальний набір даних, аналогічний до KITTI. Містить анотації для 23 класів. Зображення зроблено у 4ьох містах Сполучених Штатів;

Сегментація зображень – це лише невелика частина цілого комплексу методів та задач, що вирішуються в даному контексті, тому далі буде наведено лише декілька прикладів.

Окрім безпосередньо зображень, в даній сфері часто використовуються дані інших модальностей. Такими даними можуть бути RGB-D (зображення, де інтенсивність кольору відповідає відстані до точки, що відповідає пікселю), та «хмари точок», що отримуються із спеціальних сенсорів (LIDAR — Light Detection and Ranging), і дозволяють отримати відстань до певної точки, даючи можливість отримати повну тривимірну модель навколишнього середовища, та ін. – наприклад, метод MFNet [65] також використовує дані тепловізора у контексті сегментації зображень.

Однією з задач в даній сфері є отримання об’ємного представлення лише із зображень, отриманих з однієї або декількох камер. Наприклад, метод DORN [66] дозволяє отримати карту глибин лише із одного зображення. PSM-Net [67] розроблено з метою отримання карти глибин із стерео-зображення зображення з двох камер. Pseudo-LIDAR [68] використовує стерео-зображення для отримання «хмари точок», що нагадують результати LIDAR. Згадані архітектури нейронних мереж в тому чи іншому вигляді використовують архітектури класу «шифрувальник-дешифрувальник».

Також окремо виділяють задачі із сегментації окремих об’єктів в рамках одного класу, так звану по-об’єктну, або паноптичну сегментацію [69]. Маючи можливість розділяти між собою об’єкти одного класу дозволяє полегшити задачу передбачення їх дій, що особливо актуально в контексті безпеки дорожнього руху.

Особливість аналізу зображень міського середовище полягає у великій кількості та різноманітності об’єктів, що зустрічаються на зображеннях, а також великою різноманітністю середовища (передмістя, центральні райони, різні види доріг, різний час доби, погода, тощо). Через це, тренування адекватної нейронної мережі, що могла б виявляти невеликі об’єкти, а також об’єкти, що зустрічаються нечасто, потребує великих об’ємів даних. Одним із останніх трендів в контексті міського середовища є генерація тренувальних синтетичних даних з використанням генеративних нейронних мереж, таких як GAIA-1 [70], що дозволяють створити велику кількість повністю згенерованих розмічених зображень.

Висновки до розділу 1

В даному розділі було розглянуто основні теоретичні положення глибоких нейронних мереж.

Було описано основну термінологію та основні підходи навчання нейронних мереж, що в подальшому будуть використовуватися в наступних розділах. Наведено опис основних компонентів-шарів, з яких відбувається побудова згорткових нейронних мереж.

Розглянуто основні архітектури нейронних мереж, що застосовуються у контексті класифікації та семантичної сегментації, з коротким описом їх основних підходів та відмінностей.

Описано проблематику глибоких нейронних мереж у контексті сегментації зображень, на прикладі проблеми перенавчання, обмеження ресурсів, складність отримання наборів даних високих даних. Відповідно до розглянутих проблем, було сформовано вимоги до запропонованого методу. Також було аргументовано

використання архітектури U-Net, розглянуто різні її варіанти, а також особливості модифікації цієї архітектури.

Нейронні мережі для аналізу зображень широко використовуються в таких сферах, як аналіз медичних зображень та аналіз міського середовища, що є актуальними задачами на сьогоднішній день. Було розглянуто декілька прикладів досліджень у даних сферах, поширені публічно доступні набори даних для проведення експериментів, а також специфіку та виклики, з котрими доводиться стикатися дослідникам у процесі навчання та використання нейронних мереж у цих доменах знань.

РОЗДІЛ 2
ОСОБЛИВОСТІ МЕТОДУ СЕГМЕНТАЦІЇ З ВИКОРИСТАННЯМ МОДИФІКОВАНИХ НЕЙРОННИХ МЕРЕЖ U-NET

2.1. Запропоновані способи модифікації нейронних мереж U-Net

2.1.1. Спосіб підбору коефіцієнту розширення та глибини нейронної мережі

Припустимо, що розмір вхідних даних W × H × C, де W, H – відповідно ширина та глибина вхідного зображення, а C – глибина, або кількість каналів у цьому зображенні (наприклад, у випадку з чорно-білими зображеннями C=1, у випадку з кольоровими у форматі RGB C=3). Як було згадано вище, архітектура U-Net складається з шифрувальника, дешифрувальника та звуження, кожен з яких складається з однакових модулів. Означимо модулі карти ознак на виході модуля E_i, на виході модуля дешифрувальника – D_j, на виході звуження – B, де i – порядковий номер модуля шифрувальника від вхідного шару, j – порядковий номер модуля де-шифрувальника з останнього класифікаційного шару, $i, j \in [0, d-1]$, де d – це глибина нейронної мережі U-Net, що визначається кількістю операцій узагальнення по найбільшому (англ. *MaxPooling*), що передує «звуженню». Якщо $i = j$, то розміри $shape(E_i) = shape(D_j)$, звідси $C_i = C_j$. У оригінальній архітектурі U-Net [23], розмір вихідних карт ознак можна вирахувати як:

$$C_0 = 64$$

$$shape(E_i) = W_i \times H_i \times C_i = \frac{1}{2}W_{i-1} \times \frac{1}{2}H_{i-1} \times 2 \cdot C_{i-1} \quad (2.1)$$

$$shape(D_j) = W_j \times H_j \times C_j = \frac{1}{2}W_{j-1} \times \frac{1}{2}H_{j-1} \times 2 \cdot C_{j-1} \quad (2.2)$$

$$shape(B) = \frac{1}{2}W_{d-1} \times \frac{1}{2}H_{d-1} \times 2 \cdot C_{d-1} \quad (2.3)$$

W_0, H_0 залежать від розмірів вхідного зображення. При збільшені глибини d, кількість параметрів значно зростає. Варто зазначити, що лише C_i впливає на

кількість параметрів. Користуючись формулою (1.3), можна розрахувати кількість параметрів обох згорткових шарів «звуження» $N_{B_1}^P, N_{B_2}^P$ при збільшені глибини $d' = d + 1 = 6$, якщо слідувати підходу із оригінального дослідження (тобто к-сть ядер обох шарів буде 2048): $N_{B_1}^P = (3 \cdot 3 \cdot 1024 + 1) \cdot 2048 = 1.8 \cdot 10^7$, $N_{B_2}^P = (3 \cdot 3 \cdot 2048 + 1) \cdot 2048 = 3{,}7 \cdot 10^7$

З урахуванням усіх додаткових шарів, це збільшує кількість параметрів майже в 4 рази (з 31 млн. параметр до майже 128 млн. параметрів).

В якості покращення даної архітектури, було запропоновано використання гіперпараметрів - *коефіцієнта розширення архітектури* R (запропонованим автором дисертації у [54]), та *глибини архітектури* δ.

Глибина нейронної мережі $\boldsymbol{\delta = \max(i) + 1}$ **,** де max(i) – дорівнює кількості модулів «шифрувальника» , що передують «звуженню». У класичній архітектурі U-Net, $\boldsymbol{\delta = 5}$**,** що означає наявність 4 модулів шифрувальника, та 4 модулів дешифрувальника.

Коефіцієнти розширення R – коефіцієнт, який регулює кількість ядер згорткових шарів і, відповідно, глибину карт ознак на вході наступного модуля шифрувальника (E_i), де-шифрувальника (D_j) або звуження (B), виходячи з глибини попередніх блоків (E_{i-1}, D_{j-1}), використовуючи формули

$$shape(E_i) = shape(D_j) = W_i \times H_i \times C_i = \frac{1}{2} W_{i-1} \times \frac{1}{2} H_{i-1} \times R \cdot C_{i-1} \qquad (2.4)$$

$$shape(B) = \frac{1}{2} W_{\boldsymbol{\delta}-1} \times \frac{1}{2} H_{\boldsymbol{\delta}-1} \times R \cdot C_{\boldsymbol{\delta}-1} \qquad (2.5)$$

Таким чином надається можливість більш гнучкого регулювання розмірів мережі, а також більш ефективного використання пам’яті та інших ресурсів, що уможливлює тренування нейронної мережі на більш старих або дешевших GPU.

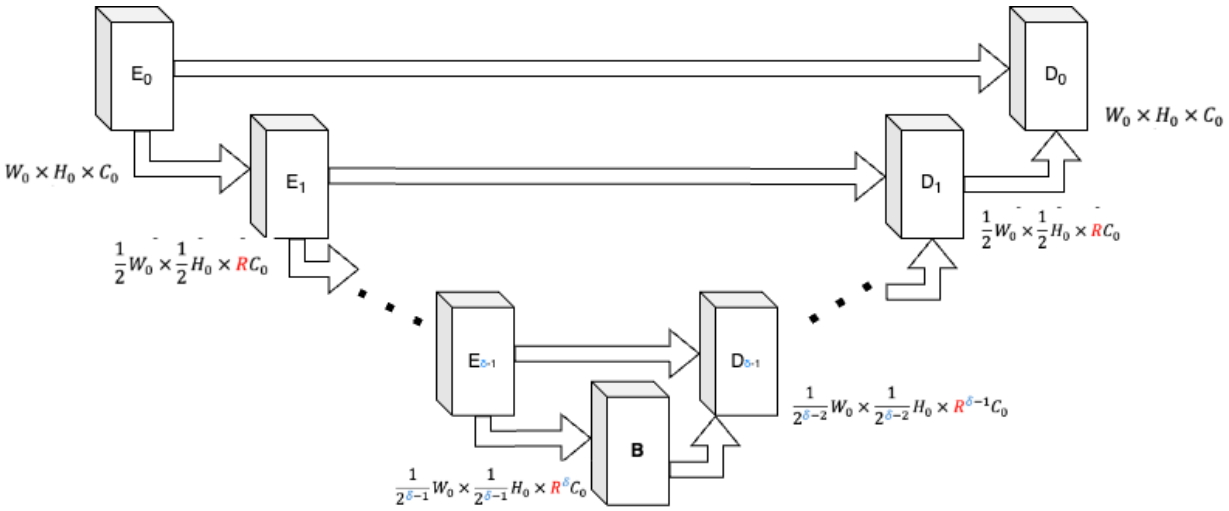

Рисунок 2.1. Узагальнена схема запропонованої архітектури U-Net з використанням коефіцієнту розширення R

Варто зазначити, що глибина мережі є обмеженою розмірністю вхідних даних, в силу того, що з кожною операцією узагальнення, розмірність карти ознак зменшується у 2 рази, тобто максимальна глибина мережі для аналізу двовимірних зображень:

$$\boldsymbol{\delta}_{max} = \lfloor log_2(min(W, H)) \rfloor \tag{2.6}$$

Дане обмеження особливо відчутне для тривимірних об'ємів, оскільки через великий об'єм пам'яті, який споживається при їх обчисленні, розмірність вхідних даних не може бути великою. Наприклад, якщо МРТ апарат дозволяє зробити 100 зрізи, максимальна глибина мережі буде 6.

Значення коефіцієнту розширення R було розглянуто у межах від 1 до 2. При значенні R<1, нейронна мережа надмірно спроститься, оскільки в наступних блоках шифрувальника кількість фільтрів буде зменшуватися. Значення R>2 є теоретично можливими, однак це призведе до значного збільшення розмірів архітектури, що протирічить поставленим у даній роботі завданням.

Схожий підхід було використано в архітектурах сімейства EfficientNet, де також були підібрані гіперпараметри, які регулюють кількість параметрів нейронної мережі, і було отримано табличні значення в результаті декількох експериментів, що дозволяє іншим дослідникам обрати нейронну мережу виходячи з наявних ресурсів та поставлених задач.

2.1.2. Спосіб глибинних роздільних проміжних зв’язків архітектури U-Net.

В ході експериментів нами було запропоновано принципово нову модифікацію U-Net, U-Net з Глибинними Роздільними Проміжними зв’язками - (англ. *Depthwise-Separable-Skip Connections U-Net*, або *DeSSCo-UNe*t). Ключовим компонентом даної модифікації є додатковий модуль DeSSCo, що додається до кожного з проміжних зв’язків U-Net, і який можна описати наступним виразом:

$$\text{x} \in \mathbb{R}^{\text{W}\times\text{H}\times\text{C}}, DeSCCo(f,x): x \to R^{W\times H\times f}$$

$$DeSSCo(f,x) = BN\left(ReLU\left(Conv_{1\times1}\left(f, BN\left(ReLU\left(DConv_{3\times3}(x)\right)\right)\right)\right)\right) \quad (2.7)$$

Де BN – нормалізація вибірки (Batch Normalization) [10], ReLU [11] – функція активації, $DConv_{3\times3}$ – операція глибинного згортування з розміром ядра 3×3, $Conv_{1\times1}$– згорткова операція з ядром 1×1 та кількістю фільтрів f, а x – карта ознак. Аналогічний модуль також було запропоновано для архітектури 3D-U-Net, з ядрами 3×3×3 та 1×1×1 відповідно.

Дана архітектура була мотивована архітектурою MobileNet [9], де було запропоновано глибинні роздільні згорткові шари (англ. *Depthwise Separable Convolution*, зображено на Рисунок 2.2), в якій було запропоновано додавання додаткових згорткових рівнів до проміжних зв’язків. Глибинні роздільні згорткові шари дозволяють додати додатковий вимір обчислювань, що враховує не лише локалізовані в конкретній області вхідного зображення карти ознак, але й глобальний контекст зображення, при цьому вони не потребують великої кількості параметрів (що регулюються розміром ядра і глибиною вхідної карти ознак).

Ідея додати ці модулі саме на проміжні зв’язки архітектури U-Net була почерпнута з архітектури mU-Net [55], де було додано додаткові шари звичайних згорткових шарів. Таким чином автори архітектури намагалися покращити результати за рахунок компенсації втрат просторових ознак в результаті операцій узагальнення.

В результаті додавання глибинних роздільних згорткових шарів для двовимірних вхідних даних до проміжних зв’язків, при $f = C$ (де C – глибина карти

ознак на вході у модуль.), розмірність карти ознак не змінюється, а кількість додаткових параметрів, що додається до мережі, вираховується з використанням формул (1.3) та (1.4) наступним чином.

$$N^P_{DeSSCo} = N^P_{DConv_{3\times3}} + N^P_{Conv_{1\times1}} = (3 \cdot 3 + 1) \cdot C + (1 \cdot 1 \cdot \mathrm{C} + 1) \cdot C = \\ = C^2 + 11C \qquad (2.8)$$

Метою використання даного підходу було покращення результатів (точності) сегментації без суттєвого збільшення розмірів архітектури.

Якщо додати цей модуль до кожного з 4ьох проміжних зв'язків оригінальної U-Net архітектури (з глибинами 64, 128, 256 та 512) – кількість додаткових параметрів буде $64^2 + 128^2 + 256^2 + 512^2 + 11 \cdot (64 + 128 + 256 + 512) = 358720$

Це становить трохи менше ніж 1.5% приросту розміру архітектури. Таким чином, завдяки глибинним згортковим шарам отримана можливість додатково натренувати параметри на основі додаткової вимірності карт ознак.

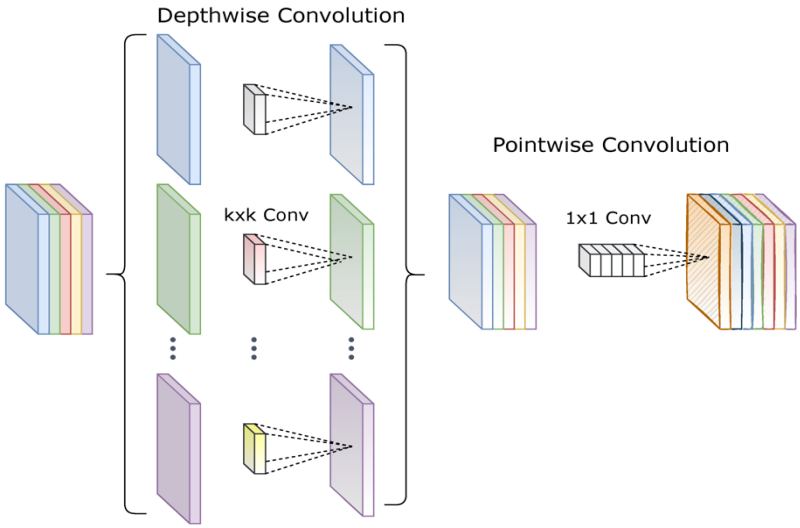

Рисунок 2.2. Візуальне представлення глибинних роздільних згорткових шарів, запропонованих в архітектурі MobileNet [9]. (рисунок взято з [9])

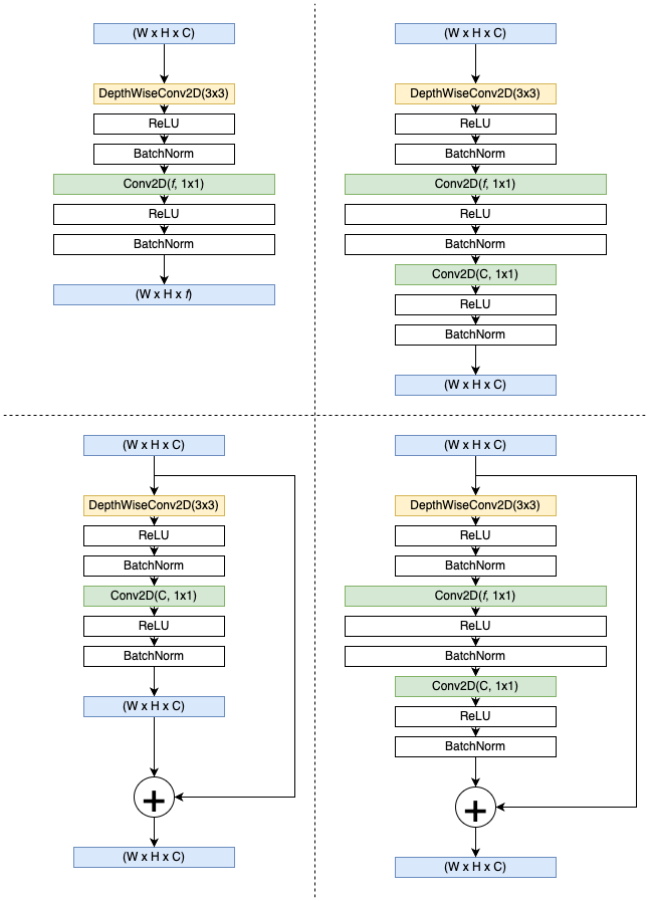

Рисунок 2.3. Візуальне зображення блоку глибинних роздільних проміжних зв'язків (зліва) та модифікованих глибинних роздільних проміжних зв'язків із звуженням. C – глибина вихідних даних, f кількість фільтрів у згортковому шарі. В базовому вигляді, $f = C$, тобто глибині вхідних даних. Знизу – додаткова модифікація запропонованих шарів залишковими зв'язками.

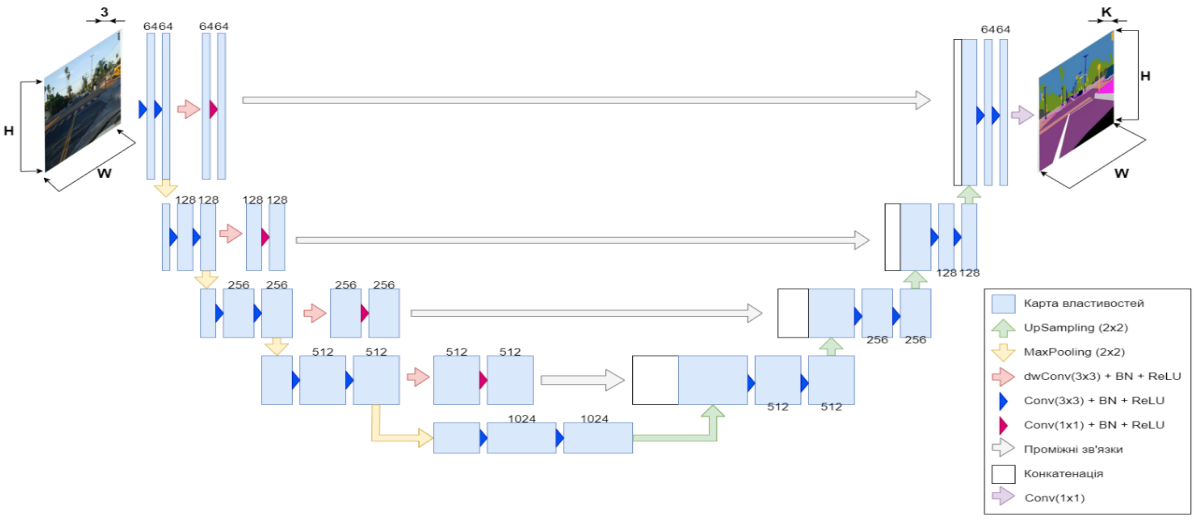

Рисунок 2.4. Запропонована архітектура нейронної мережі з Глибинними Роздільними Проміжними зв'язками DeSSCo

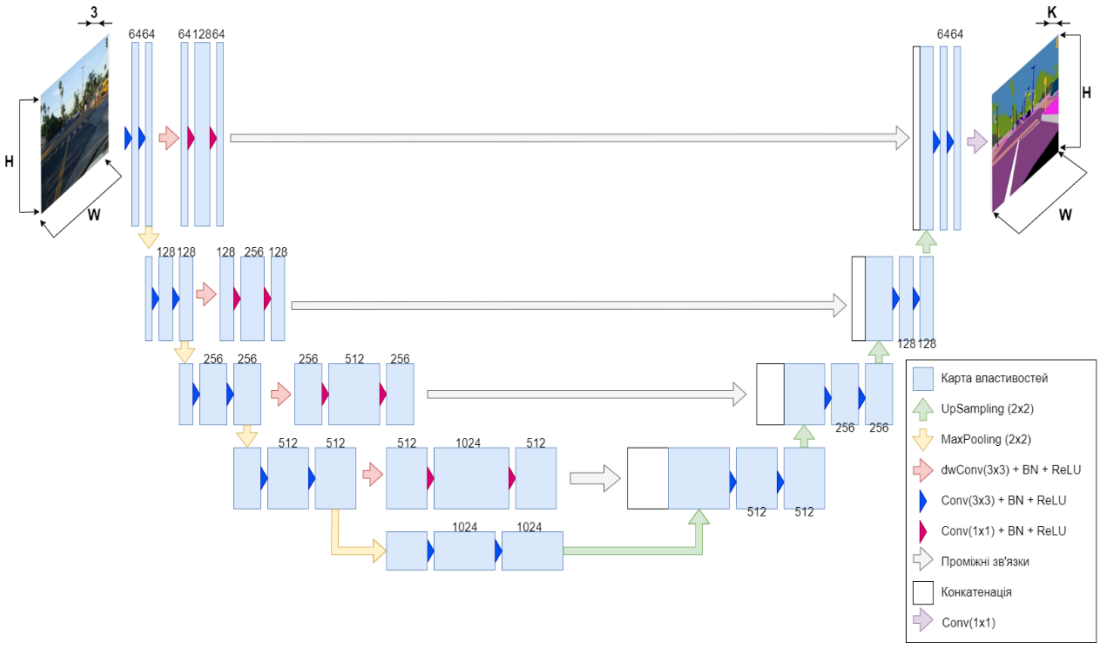

Рисунок 2.5. Запропонована архітектура з використанням глибинними роздільними проміжними зв’язками із звуженням DeSSCoB.

Також була розглянута можливість змінювати кількість ядер у глибинному роздільному згортковому шарі (англ. *Depthwise Separable convolution*) – тобто коли $f \neq \mathrm{C}$. Додаткова складність при зміні розмірності карти ознак проміжних зв’язків заключається у тому, що при цьому змінюється розмірність вхідної карти до модулів дешифрувальника – відповідно, змінюється загальна кількість параметрів дешифрувальника. Якщо в класичній архітектурі $C_j = 3C_{j-1}$, то у випадку зі зміною проміжних зв’язків $C_j = 2C_{j-1} + f$, відповідно додаткова зміна кількості параметрів при $f \neq \mathrm{C}$ для кожного модуля дешифрувальника:

$$\Delta N_j^P = \left(1 + \left(2C_{j-1} + f\right) \cdot \prod_j^{k-1} m_j\right) C_{j-1} - \left(1 + 3C_{j-1} \prod_j^{k-1} m_j\right) C_{j-1} = \\ = 9 \cdot \left(f C_{j-1} - C_{j-1}^2\right) \quad (2.9)$$

Кількість додаткових параметрів на проміжному зв’язку:

$$N_{DSC_j}^P = f\ C_{j-1} + 11C_{j-1} \quad (2.10)$$

Звідси повна зміна кількості параметрів вираховується як:

$$\Delta^{\mathrm{NP}} = \sum_{j=1}^{\delta-1} \left(N_{DSC_j}^{P} + \Delta N_j^{P} \right) = 10 f C_{j-1} + 11 C_{j-1} - 9 C_{j-1}^2 \qquad (2.11)$$

Для класичної архітектури, якщо $f = 2 \cdot C_{j-1}$, $C_0 = 64$, це означатиме розмірів нейронної мережі на 3.8 млн. параметрів.

Аби зменшити цю зміну кількості параметрів, запропоновано додати ще один згортковий шар з ядрами 1x1 і кількістю ядер $f = C$, аби зберегти розмірність проміжних шарів. Дана модифікація отримала назву «Глибинні Роздільні Проміжні зв'язки із Звуженням», (англ. *Depthwise-Separable Skip Connections with Bottleneck (DeSSCoB)*):

$$DeSSCoB(C, f, x) = BN\left(ReLU\left(Conv_{1\times1}\left(C, DeSSCo(f, x) \right) \right) \right) \qquad (2.12)$$

Загальна кількість параметрів додаткових параметрів в цьому випадку буде наступною:

$$\Delta^{\mathrm{NP}} = \sum_{j=1}^{\delta-1} \left(N_{DeSSCoB_j}^{P} \right) = \sum_{j=1}^{\delta-1} \left(N^{P}_{DConv_{3\times3}^{(C_j)}} + N^{P}_{Conv_{1\times1}^{(f)}} + N^{P}_{Conv_{1\times1}^{(C_j)}} \right)$$

$$= \sum_{j=1}^{\delta-1} \left(f^2 + f + f C_j + 11 C_j \right) \qquad (2.13)$$

Для $f = 2C$ формула зведеться до виду:

$$\Delta^{\mathrm{NP}} = \sum_{j=1}^{\delta} \left(6 \mathrm{C}_j{}^2 + 13 C_j \right) \qquad (2.14)$$

Ця ідея нагадує блок ConvNext [71], у якому також використовується глибинний згортковий шар з ядрами 7x7, внутрішній згортковий шар з ядрами 1x1, збільшує глибину проміжної карти у 3 рази, та зовнішній блок 1x1, що слугує «звуженням» та повертає карту ознак до вхідної глибини. Отримана архітектура зображена на Рисунок 2.5.

Ще одна модифікація – модифікації модулів DeSSCo і DeSSCoB з додаванням залишкових зв'язків, таким чином комбінуються оригінальні проміжні зв'язки

нейронної мережі U-Net із запропонованими новими модулями шляхом по елементного додавання. Ця ідея є розвитком підходу, запропонованого в сімействі архітектури ResNet [21]. Ці модифікації отримали назву «Залишкові Глибинні Роздільні Проміжні зв'язки» (англ. *Depthwise Separable Skip Connections with Residuals, DeSSCoR*, у варіанті із звуженням - *DeSSCoBR*). В теорії, така будова модуля дозволяє уникнути згасаючих градієнтів, і зберегти вплив оригінальних проміжних зв'язків, тоді як запропоновані модулі слугують своєрідними підсилювачами сигналів. Варто зазначити, що у випадку *DeSSCoR*, ця модифікація має сенс лише у випадку $f = C$, оскільки для операції поелементного додавання обидві карти ознак (тобто вхідна та вихідні карти) мають бути однакової розмірності. Дані модулі зображені на Рисунок 2.3 знизу, та характеризуються формулами:

$$DeSSCoBR(C, f, x) = x + DeSSCoB(C, f, x) \tag{2.15}$$

$$\mathrm{DeSSCoR}(C, x) = x + DeSSCo(C, x) \tag{2.16}$$

Однією з переваг модифікації проміжних зв'язків є можливість додавати ці зміни модульним шляхом до усіх можливих модифікацій архітектур без суттєвих їх змін. Для демонстрації практичності даного підходу, було також випробувано модифікацію мережі Attention-UNet [37], що зображена на Рисунок 2.6.

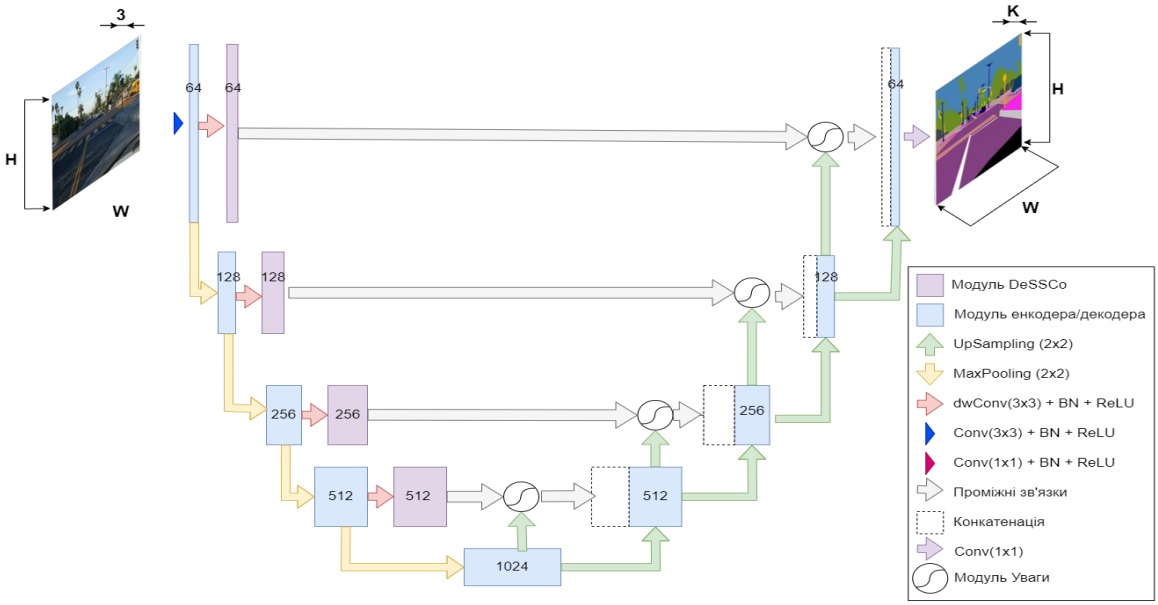

Рисунок 2.6. Запропонована модифікація Attention-U-Net з додаванням глибинних роздільних проміжних зв`язків (DeSSCo).

Аналогічно, запропонований спосіб використання глибинних роздільних згорткових шарів можливо застосувати і у вирішені задачі тривімірної сегментації з використанням архітектури U-Net 3D. Для цього необхідно використовувати глибинні згорткові шари з тривимірними ядрами 3x3x3 (замість двовимірних 3x3) та точкові згорткові шари з тривимірними ядрами 1x1x1 (замість двовимірних 1x1).

2.2. Метод сегментації з використанням модифікованих нейронних мереж U-Net

2.2.1. Загальна схема методу сегментації.

Для узагальнення основних положень запропонованого методу сегментації зображень з використанням модифікований нейронних мереж U-Net нижче наведено його повну схему (Рисунок 2.7).

В результаті аналізу відомих варіантів архітектури U-Net, стандартну методологію модифікації нейронних мереж U-Net можна звести до таких основних кроків:

- Підбір блокової архітектури, чи архітектури, що використовує опорні мережі (backbone) в шифрувальнику або дешифрувальнику
- Підбір модифікації проміжних зв’язків

Блокова архітектура характеризується тим, що вона складається з декількох блоків (модулів), що в свою чергу складаються з однакової послідовності шарів, і відрізняються між собою певним гіперпараметром, найчастіше – кількістю фільтрів згорткових шарів. Такою архітектурою є базова архітектура U-Net. Архітектура на базі опорні мережі використовує окрему архітектуру (зазвичай, для класифікації зображень, як наприклад було зроблено в модифікаціях UNet++ та TransUNet) в шифрувальнику або дешифрувальнику, тобто кількість шарів, та їх послідовність задана параметрами цієї мережі. До цієї категорії можна відносити усі архітектури, що не належать до блокових.

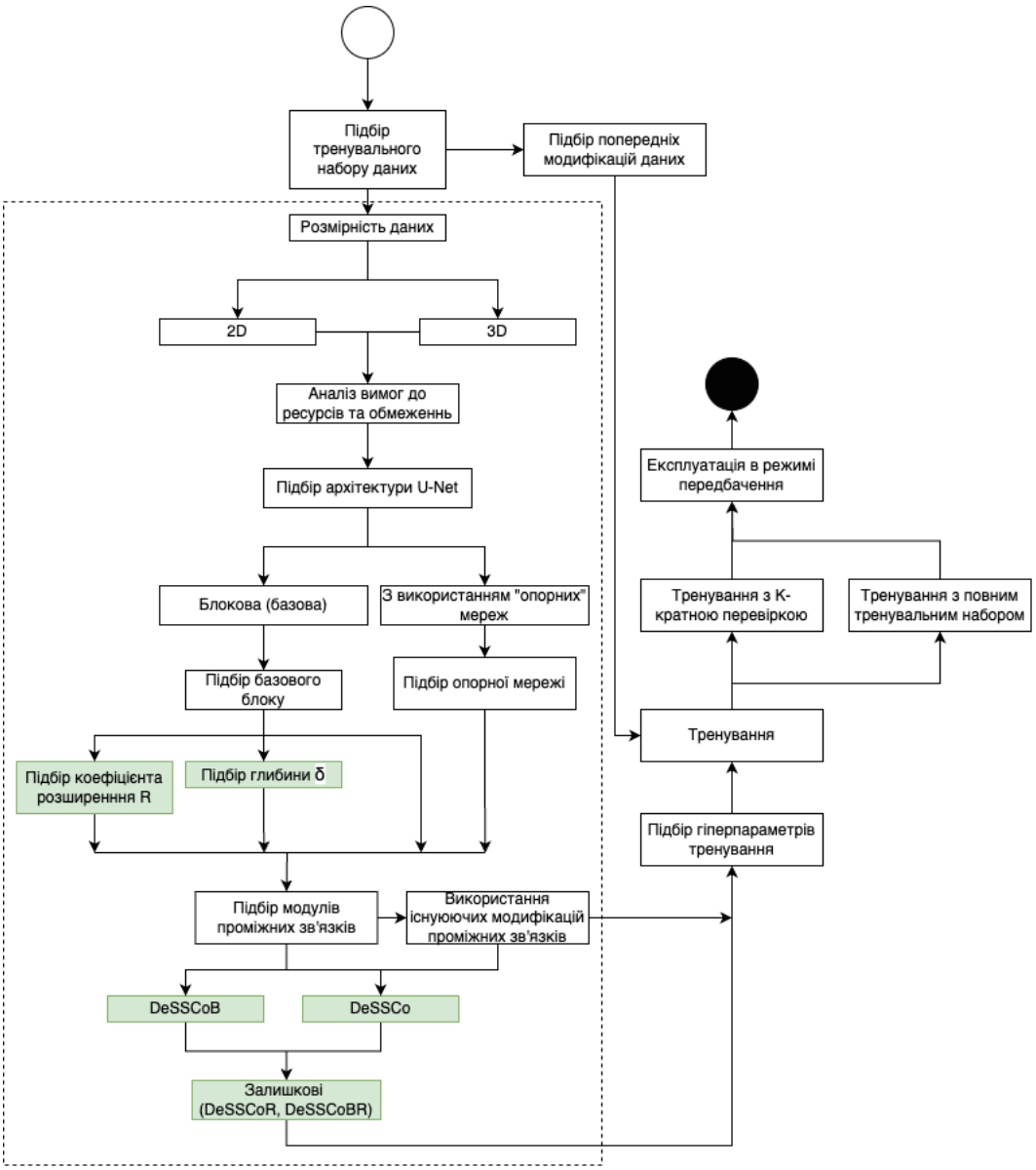

Рисунок 2.7. Запропонований метод сегментації зображень з використанням способів модифікації нейронних мереж типу U-Net (обведено штрихом) в рамках ширшого методу тренування нейронних мереж, що використовувався в експериментах. Зеленим виділено нові способи, що були запропоновані в даній роботі.

Така методологія дозволяє комбінувати між собою різноманітні модифікації, таким чином надаючи можливість створювати велику кількість комбінованих архітектур.

В роботі обґрунтовано два нові способи модифікації архітектури – спосіб підбору коефіцієнта розширення та спосіб глибинних роздільних проміжних зв’язків. Варто зазначити, що спосіб підбору коефіцієнта розширення за

визначенням може застосовуватися в архітектурах блокового типу, тобто таких, чию будову можливо параметризувати з використанням запропонованих гіперпараметрів R та δ. Хоча в цій роботі розглядалася базова архітектура U-Net, що використовує в якості блоків два згорткових шари Conv 3x3, даний спосіб модифікації можливо застосувати до будь-яких варіантів блоків шифрувальника або дешифрувальника.

Спосіб глибинних роздільних проміжних зв'язків можливо застосувати до будь-якого варіанту мережі U-Net, що було показано на прикладі Attention-UNet, де комбінується два види модифікацій проміжних зв'язків.

Цей метод також дозволяє утворити гібридну архітектуру шляхом використання обох запропонованих способів модифікації архітектури U-Net.

2.2.2. Оцінки узагальнення нейронної мережі на основі К-кратної перехресної перевірки.

Як було згадано раніше, проблема "перенавчання" є однією з фундаментальних проблем машинного навчання. Навіть якщо якась з оглянутих вище архітектур дає можливість досягнути кращих результати в тестах, на реальних наборах даних деякі з вони будуть показувати по різному. Завдяки перевірці точності передбачень нейронної мережі за допомогою валідаційного набору є змога перевірити якість нейронної мережі перед тим, як використовувати її на невідомих даних. Проте вибірка даних для валідаційного набору теж відграє важливу роль. Поняття *упередженності вибірки* (selection bias), що бере своє коріння з соціології [72], також відіграє критичну роль у машинному навчанні [73], оскільки вибір нерепрезентативної вибірки для валідаційного набору може створити хибне уявлення про узагальнення нейронної мережі.

Для того, аби мінімізувати фактор валідаційного набору даних, в даній роботі використовується метод k-кратної перехресної перевірки [74]. Його суть полягає в тому, щоб розбити тренувальний набір на k-різних пронумерованих груп, за можливості, однакових за розмірами, і натренувати k нейронних мереж, використовуючи групу під відповідним номером в якості валідаційного набору,

тоді як решта набору використовується для тренування. Для кожної k-ої групи отримуються метрики точності сегментації відповідною мережею і аналізується їх середнє значення та стандартне відхилення, на основі чого робиться висновок про стабільність роботи обраної архітектури та її здатність до узагальнення. Чим менше відхилення, тим стабільніше працює нейронна мережа. К-кратна перехресна перевірка також дозволяє перевірити, чи дійсно одна нейронна мережа показує кращу точність результатів за іншу, оскільки часто різниця між результатами двох нейронних мереж часто може лежати у межах статистичної похибки. Якщо одна з архітектур показує результат, що виходить за рамки стандартного відхилення іншої, є підстави говорити про якісну різницю між двома архітектурами.

Якщо представити нейронну мережу як функцію $f(\theta_k, x): X \to Y$, (де θ_k – ваги нейронної мережі, натреновані на групі з порядковим номером k, X – множина вхідних даних, Y – множина результатів), очікуваний результат як $g \in Y$, функцію метрики перевірки як $\nu\big(g, f(\theta_k, x)\big): Y \to \mathbb{R}$, то результат К-кратної перехресної перевірки можна описати як середнє значення $\mu^{(CV)}$ та стандартний розподіл $\sigma^{(CV)}$ серед К результатів метрики перевірки ν для різних θ_k:

$$\mu^{(CV)} = \frac{1}{K}\sum_{k=1}^{K} \nu(f(\theta_k, x), g) \tag{2.17}$$

$$\sigma^{(CV)} = \sqrt{\frac{1}{K}\sum_{k=1}^{K} (\nu(f(\theta_k, x), g) - \mu^{(CV)})^2} \tag{2.18}$$

2.2.3. Модифікації набору даних.

Окрім архітектури та гіперпараметрів, часто для покращення результатів застосовують метод попередніх модифікацій набору даних (англ англ. *dataset augmentation)*. Існує два види попередніх модифікацій даних – модифікації в процесі тренування (англ. *train time augmentations*), та модифікації в процесі тестування (англ. *test time augmentations*) [75]. Метою модифікацій даних в процесі тренування є збільшення варіативності та розміру даного набору. Це дозволяє

частково вирішувати описану в Розділі 1 проблему перенавчання нейронної мережі, оскільки цей метод додає додатковий «шум» у тренувальний набір, таким чином нейронна мережа з меншою вірогідністю зможе пристосуватися до цього шуму і буде краще узагальнюватися на невідомих даних. При модифікації даних в режимі тестування, до вхідного зображення застосовуються декілька з наведених вище модифікації, і до кожного із зображень застосовується натренована мережа. Отримані результати пізніше приводяться до початкового положення (наприклад, в разі повороту зображення) і отримується по-піксельне середнє значення. В даній роботі використовувався метод попередньої модифікації даних тестового набору (англ. *Train time augmentation, TTA*).

Одними з поширених маніпуляцій над зображеннями в рамках попередньої модифікації даних є наступні:

- Рандомізація – зміна порядку елементів в тренувальному наборі доволі сильно впливає на результати передбачень нейронної мережі. Це є одним із проявів упередженості відбору (selection bias) [73], адже у разі незмінного порядку елементів в тренувальному наборі алгоритм оптимізації ваг може стати «звикнути» (стати упередженим) до цього порядку, що призведе до перенавчання. Практичний вплив упорядкованості набору було показано в [52];
- Інверсія та повороти зображень – поворот зображення під випадковим кутом чи інверсія допомагають створити абсолютно нове зображення;
- Накладання фільтрів – до зображення застосовуються фільтри, які змінюють кольорову гаму. Фільтри зазвичай мають невелику розмірність, наприклад, фільтр Собеля [76] має розмірність матриці 3x3. Його часто використовують для виокремлення контурів;
- Додавання шуму – розмиття зображення, підсилення контрастності, видалення частини зображень також може застосовуватися в рамках даного підходу;

Комбінація декількох типів модифікацій з використанням випадкових чисел дозволяє досягати великої варіативності даних при невеликих розмірах набору даних, що сприяє покращенню узагальнення нейронної мережі.

Висновки до розділу 2

В даному розділі було розглянуто теоретичні та практичні аспекти запропонованого методу.

Було запропоновано та обґрунтовано способи модифікації архітектур U-Net та особливості їх реалізації, а саме - спосіб підбору коефіцієнту розширення, спосіб глибинних роздільних згорткових зв’язків та їх варіантів (глибинних роздільних згорткових зв’язків із звуженням, залишкових глибинних роздільних проміжних зв’язків). Запропоноване використання коефіцієнту розширення ставить за мету контроль розмірів та глибини нейронної мережі шляхом зменшення кількості ядер на наступних блоках шифрувальника та дешифрувальника. Спосіб глибинних роздільних проміжних зв’язків (DeSSCo), в свою чергу, додає додаткові згорткові шари, при цьому збільшуючи нейронну мережу на незначну кількість параметрів, що теоретично дозволить виявляти більш складні закономірності у вхідних даних, аніж базова архітектура. Також було продемонстровано можливість використання роздільних проміжних зв’язків разом з іншими модифікаціями U-Net на прикладі з архітектурою Attention-U-Net. Варто зазначити, що дані способи також можуть бути використані у контексті сегментації тривимірних об’єктів та відповідними архітектурами нейронної мережі U-Net 3D для роботи з ними.

Було наведено загальну схему запропонованого методу сегментації зображень з використанням модифікованих архітектур U-Net, що описує використання описаних способів, а також описує ширшу методологію тренування нейронних мереж, використану в даній роботі, що включає в себе такі аспекти, як попередню модифікацію даних, та K-кратну перехресну перевірку.

РОЗДІЛ 3
ЕКСПЕРИМЕНТИ З МЕТОДОМ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МОДИФІКОВАНИХ АРХІТЕКТУР U-NET

3.1. Набори даних.

Набори даних є ключовою частиною будь-якого методу у машинному навчанні, що визначає контекст та доменну область використання нейронних мереж. Важливою властивістю нейронної мережі є універсальність її використання незалежно від області застосування. Аби перевірити практичність та універсальність запропонованого методу, було обрано декілька наборів даних, що використовуються в контексті медичних зображень та аналізу міського середовища. Аналіз результатів, отриманих на декількох різних наборах даних, дозволить виявити сильні та слабкі сторони запропонованого методу.

3.1.1. Набір даних Maddison University of Wisconsin Gastro-Intestinal.

Maddison University of Wisconsin Gastro-Intestinal (UWGIT) - набір даних [77], що містить зображення, отриманні за допомогою магнітно-резонансної томографії (МРТ) кишково-шлункового тракту. Він складається з 240 повних МРТ знімків, узятих з 85 пацієнтів, кожен знімок містить 144 (або 80) зрізів, мають однакову висоту H та ширину W, таким чином отримуючи об’єм розміром $H \times W \times 144$ вокселів.

Сегментація знімків складається з 3-ох класів

- Шлунок (stomach);
- Малий кишківник (small bowel);
- Великий кишківник (large bowel);

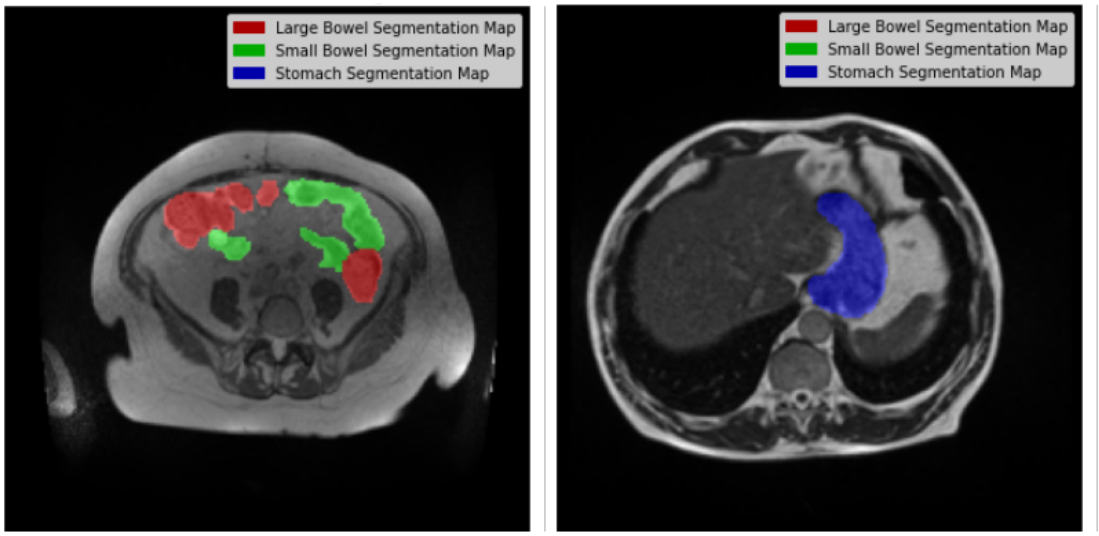

Рисунок 3.1. Зображення з набору даних UWGIT. Маски для відповідних класів відображені різними кольорами.

3.1.2. Набір даних Cityscapes.

Набір даних Cityscapes [62] – один з найпопулярніших наборів даних для тестування методів сегментації. Даний набір даних було отримано шляхом ручної сегментації об'єктів, які належать до 35 класів, на знімках, отриманих з камери, розміщеної за фронтальним скло авто, у 50 різних містах Німеччини. В даному досліджені було використано підмножину даних, що складає 2975 знімків анотованих тренувальних даних, та 500 знімків анотованих валідаційних даних. Для спрощення, знімки було стиснуто до розміру 256x512.

Оскільки міське середовище є доволі різноманітним, у наборі CityScapes для кінцевої перевірки результатів пропонується вираховувати метрики лише для 20 типових класів, оскільки решта класів є або службовими (наприклад, на Рисунок 3.2 видно капот автомобіля, який виконує фотографування середовища перед ним, і сегментація даної частини повинна належати до фонового (нульового класу), і є такою, що не несе у собі практичного значення).

Рисунок 3.2. Зображення з набору даних Cityscapes, з накладеними на нього масками сегментації

3.1.3. Набір даних Synapse.

Даний набір даних [45] являє собою 30 анотованих МРТ зображень черевної порожнини об’ємом 512 x 512 x 153. Наявні анотації належать до 13 класів, що відповідають відповідним органам –

- Селезінка
- права нирка
- ліва нирка
- жовчний міхур
- стравохід
- печінка
- шлунок
- аорта
- нижня порожниста вена
- ворітна і селезінкова вени
- підшлункова залоза
- права надниркова залоза

- лівий наднирник

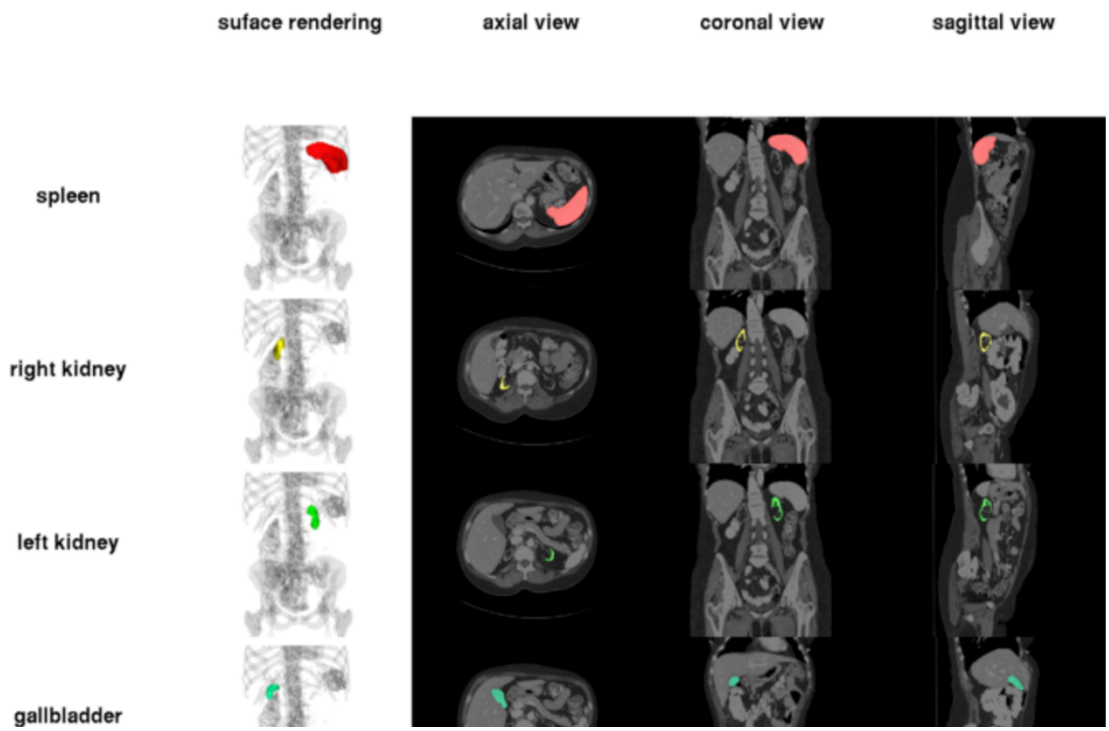

Рисунок 3.3. Зображення із набору даних Synapse (взято з [45])

Для дослідження, було враховано сегментацію 8 органів - підшлункова залоза, шлунок, селезінка, печінка, аорта, жовчний міхур, ліва та права нирки. Даний набір було обрано через те, що він використовувався для оцінки точності інших модифікацій нейронної мережі, таких як Attention-UNet, TransUNet, SwinUNet, що дозволить порівняти результати запропонованого методу з іншими, поширеними нейронними мережами.

3.1.4. Набір даних Brain Tumor Segmentation Challenge 2020.

Набори даних Multimodal Brain Tumor Segmentation Challenge (BraTS 2020) - публічно доступні набори даних, що використовується для щорічних конкурсів науковців. В даному досліджень використовується набір даних 2020 року [48]. Він складається МРТ знімків головного мозку для 367 пацієнтів, що містять ракові пухлини. Для кожного пацієнта зроблено 4 МРТ з використанням процедур T1-weighted (T1), post-contrast T1-weighted (T1ce), T2-weighted (T2), and T2 Fluid-Attenuated-Inversion-Recovery (FLAIR). Кожен МРТ знімок являє собою 155 знімків

розміру 240 на 240 пікселів, тобто об’єм 240×240×155. Сегментація виконана вручну спеціалістами, та містить три категорії:

- збільшення пухлини (ET);
- перитуморальний набряк (ED);
- некротичне ядро пухлини без збільшення (NCR/NET);

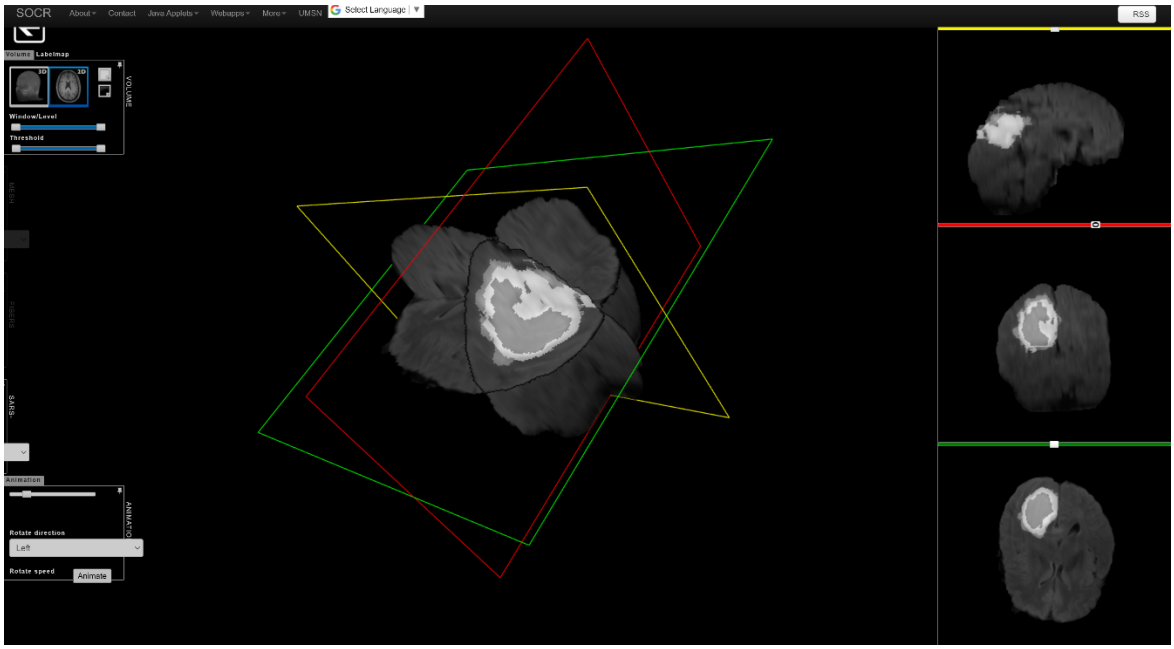

Рисунок 3.4. Приклад МРТ об’єму з набору BRaTS. Відображено з використанням програмного забезпечення SOCR Brain Viewer [78]

Даний набір було обрано з причини його відносно невеликих розмірів, багатокласовості, мультимодальності, а також з метою демонстрації роботи запропонованих модифікації на тривимірних об’ємах, що є актуальним у таких контекстах, як аналіз знімків МРТ.

Важливо також зазначити, що на відміну від інших наборів даних, згаданих вище, у даному наборі даних поставлена задача діагностики пухлин головного мозку, що має важливе практичне значення. Складність даного набору даних також у тому, що пухлини мають довільну форму та розташування, що посилює вимоги щодо варіативності тренувальних даних, тоді коли в контексті сегментації органів (як в наборах Synapse та UWGIT) точки інтересу знаходяться у приблизно

визначених локаціях, що спрощує узагальнення нейронної мережі у процесі тренування.

3.2. Використані програмні і апаратні засоби розробки методу сегментації зображень з використанням модифікованих архітектур U-Net.

3.2.1. Середовище та інструменти розробки.

Для експериментів у різний період часу було застосовано бібліотеки Tensorflow [79] та PyTorch [80], запропоновані свого часу у 2015 та 2019 роках відповідно. Дані бібліотеки надають простий програмний інтерфейс для створення нейронних мереж, керуванням процесу тренування, розподіленням обчислень між різними серверами, тощо.

Більшість програмного коду на пізніх етапах дослідження було створено з використанням бібліотеки TensorFlow. Вибір на користь даної бібліотеки був зумовлений такими факторами:

- старша бібліотека, що означає більш стабільний програмний код, більшу екосистему навколо неї, менше внутрішніх помилок та більше прикладів використання;
- простота – завдяки методу нейронної мережі Model.fit(), що реалізовує у собі цикл тренування та перевірки, підкласам для роботи з даними tf.data і великій внутрішній бібліотеці швидкість розробки більша;
- декларативне оголошення нейронних мереж за допомогою підсистеми Keras [81], що спрощує розуміння будови описаного методу;

Основним інструментом розробки є Jupyter notebook – інтерактивне середовище для мови програмування Python, що надає зручний інтерфейс, використовуючи веб браузер користувача. Серед функціоналу системи – можливість писати програмний код, комбінуючи текстові пояснення, з можливістю друку у PDF, часткове виконання коду, відображення графіків, запуск скриптів за допомогою bash тощо.

Jupyter Notebook лежить в основі багатьох хмарних платформ для роботи з нейронними мережами та машинним навчанням. В дисертації використовувалися 2 таких платформи:

- Kaggle – платформа для проведення змагань серед дослідників та ентузіастів аналізу даних. Надає програмні та апаратні засоби для написання коду, слугує бібліотекою різноманітних наборів даних та соціальною платформою для науковців з усього світу; Надає можливість використовувати GPU Nvidia P-100 до 40 годин на тиждень безкоштовно;
- Google Colab – хмарна платформа від Google. Надає можливість підлючати до віртуального середовища Google Drive, та безкоштовну можливість запускати код з GPU Nvidia T4 по мірі їх наявності. Також має платну підписку, завдяки якій дається доступ до потужніших GPU Nvidia V100 та Nvidia A100;

Список використаного обладнання з його характеристиками наведено у Таблиця 3.1.

Таблиця 3.1. Порівняльна характеристика використаного обладнання

GPU	VRAM	К-сть ядер CUDA	Пропускна здатність пам'яті
NVIDIA T4	16GB	2560	320GB/s
NVIDIA P100	16GB	3584	720GB/s
NVIDIA V100	16GB	5120	900GB/s
NVIDIA A100	40GB	6912	2TB/s

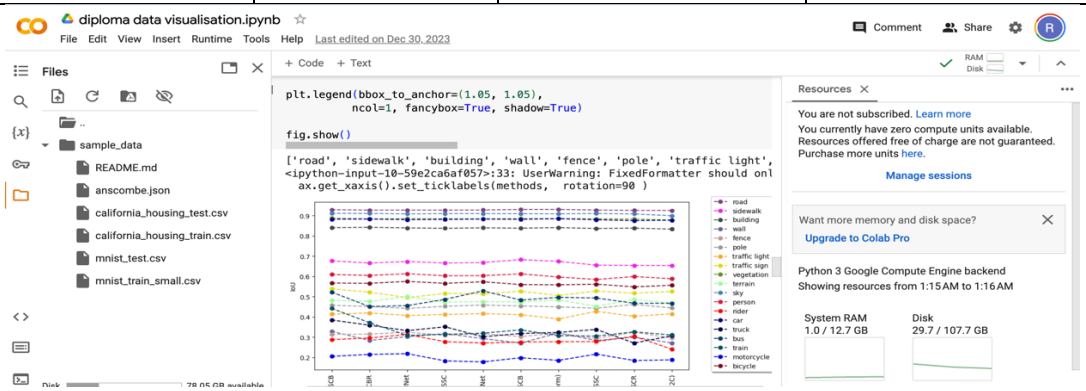

Рисунок 3.5. Інтерфейс платформи Google Colab

Серед інших інструментів розробки, варто зазначити такі бібліотеки, як:

- Numpy, pandas – для роботи з масивами даних;
- Matplotlib – для візуалізації зображень та графіків;
- Nibabel, - для зчитування NIFTI [82] зображень – одного з основних форматів зберігання МРТ сканів;

3.2.2. Програмна реалізація запропонованого методу з використанням бібліотеки Tensorflow.

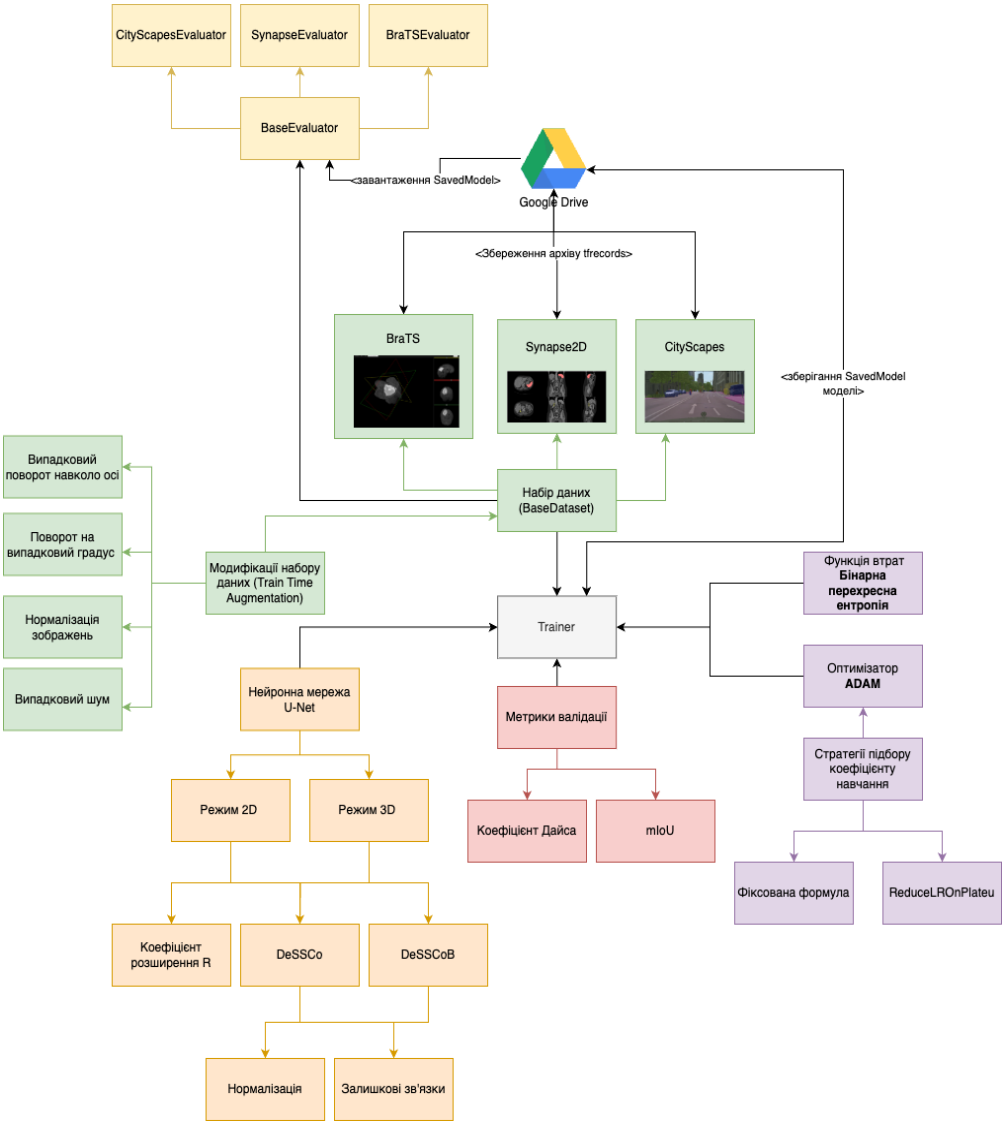

Рисунок 3.6. Запропонована високорівнева організація програмного забезпечення, що використовувалося для проведення експериментів.

Окрім вибору інструментарію, гіперпараметрів та архітектури нейронної мережі, важливим елементом будь-якого програмного забезпечення, що використовується для машинного навчання, є програмний код, який поєднує усі компоненти, необхідні для тренування, в одне ціле. Ще одним важливим фактором є зручність, можливість розширення програмного коду, його повторного використання та відтворення отриманих результатів.

Програмний код у повній мірі наведено у Додатку А.

Основні програмні модулі для роботи включають у себе наступні структури:

Класи, що наслідують BaseDataset. Робота з даними є однією з визначальних факторів для тренування. У роботу з даними входить наступний функціонал:

- Зчитування даних, їх конвертації та приведення їх представлення до формату, що використовуються такими бібліотеками, як tensorflow. Таким форматом для Tensoflow є формат TFRecord [83];
- Розбиття на тренувальний та валідаційні набори;
- Виконання попередньої обробки даних - нормалізація, додавання додаткових відступів, зміна розмірів, а також модифікації вхідних даних для збільшення їх варіативності;

В досліджені використовується 3 різновиди даних

- Зображення у форматі PNG
- тривимірні об’єми у форматі NIFTI
- двовимірні зображення, отримані з NIFTI файлів.

Під ці потреби було створено базовий клас BaseDataset, від якого унаслідуються класи NiBabelDataset (для NiFTi), ImageDataset (для зображень) та DatasetNibabelToImage, від яких уже унаслідувалися конкретні набори даних (BRaTsDataset від NiBabelDataset, SynapseDataset від DatasetNibabelToImage). BaseDataset отримує шлях до архівів, що містять оригінальні тренувальні дані, і відповідно до наслідуваного класу, виконує їх завантаження та конвертацію у TFRecord. Кожен отриманий TFRecord файл додається у архів, разом з файлами train.txt та val.txt, що містять імена файлів, які будуть використовуватися як тренувальний чи валідаційний набір. Отримана директорія з TFRecord файлами

архівується у форматі zip, та завантажується в хмарне сховище Google Drive. Вже в процесі використання набору для тренування, даний архів завантажується, розпаковується, і через використання tf.data.TFRecordDataset передається на вхід алгоритмам навчання. Даний підхід дозволяє привести різні формати до уніфікованого формату, пришвидшити процес попередньої підготовки даних, за зменшити використання пам’яті.

Програмний код, що стосується роботи з наборами даних, описано в Лістингах А.4-А.7 Додатку А.

Створення нейронних мереж. Для створення запропонованих архітектур нейронних мереж з використанням підсистеми Keras було реалізовану функцію get_unet_model, що приймає наступні параметри:

- Ім’я нейронної мережі – під цим іменем нейронна мережа зберігається на диску;
- Коефіцієнт розширення та глибину – за замовчуванням, 2 та 5 відповідно
- Розмірність даних, котрі нейронна мережа приймає на вхід да видає на виході – в залежності від того, чи аналізуються зображення (двовимірний режим) чи об’єми (тривимірний режим);
- Кількість класів класифікації – залежить від використаного набору даних;
- Список модифікацій – усі модифікації перелічено в UnetModifications (Attention-UNet, DeSSCo, DeSSCoB, DeSSCo(2C), тощо). Ці комбінації можна поєднувати одна з одною для отримання гібридних архітектур;

Реалізація даної логіки наведена у Лістингу А.9 Додатку А.

Клас Trainer. Клас, що безпосередньо відповідає за контроль над процесом тренування нейронної мережі. До основного функціоналу входить:

- В якості вхідних параметрів приймає обраний набір даних та архітектуру мережі з наперед заданою архітектурою та іменем;
- Можливість задавати гіперпараметри, такі як оптимізатор, функція втрат, метрики перевірки, кількість епох, додатковий функціонал (наприклад,

tf.keras.callbacks.Callback, що дозволяє додавати додаткову логіку на різних етапах тренувального процесу);

- Можливість завантажувати уже натреновані нейронні мережі з використанням формату SavedModel [84];

Формат SavedModel дозволяє зберігати натреновані ваги нейронної мережі, а також її архітектури, що забезпечує простий спосіб завантаження та розгортання даних нейронних мереж без необхідності повторювати їх архітектуру в програмному коді. Кожна з нейронних мереж, що натренована в процесі дослідження, зберігається в даному форматі, що дає можливість додаткового тренування, а також використовується для проведення етапу передбачення (Inference) – тобто запуску та отримання результуючих метрик на тестовому наборі даних, що власне відображуються в таблицях та графіках у наступних пунктах.

Основна ідея даного класу в тому, аби зафіксувати у простому та доступному вигляді параметри кожного експерименту, з метою їх відтворення у майбутньому.

Таким чином, завдяки невеликій кількості програмного коду можливо повністю визначити параметри тренування. Нижче наведено приклад визначення експерименту для архітектури DeSSCoB на наборі даних CityScapes:

```
def train_cityscapes_desscob():
    cscapes = CityScapes(pad_ratio=32, image_shape=(256, 512),
training_mode=True)

    model = get_unet_model(cscapes.pixel_channels, cscapes.max_classes,
                           expansion_rate=2, depth=5, start_filters=64,
                           modifications=[UnetModifications.DESSCO2],
                           mode=cscapes.mode, name='cityscapes-desscob')

    opts = SchedulerOpts()
    trainer = Trainer(
        model=model,
        dataset=cscapes,
        batch_size=8,
        epochs=50,
        scheduler=opts,
        monitor_metric='val_iou',
    )

    return trainer
```

Програмна реалізація класу Trainer наведена у Лістингу А.12 Додатку А.

Класи BaseEvaluator. Дані підкласи виконують безпосередньо процедуру підрахунку метрик для тестових наборів даних. Кожен експеримент має власний клас для проведення даного процесу. Дані класи підтримують вичитування збереженої архітектури та її ваг з SavedModel.

Програмний код для цих класів наведено у Лістингу А.13 Додатку А.

3.3. Результати способу підбору коефіцієнта розширення.

3.3.1. К-кратна перехресна перевірка на наборі UWGIT.

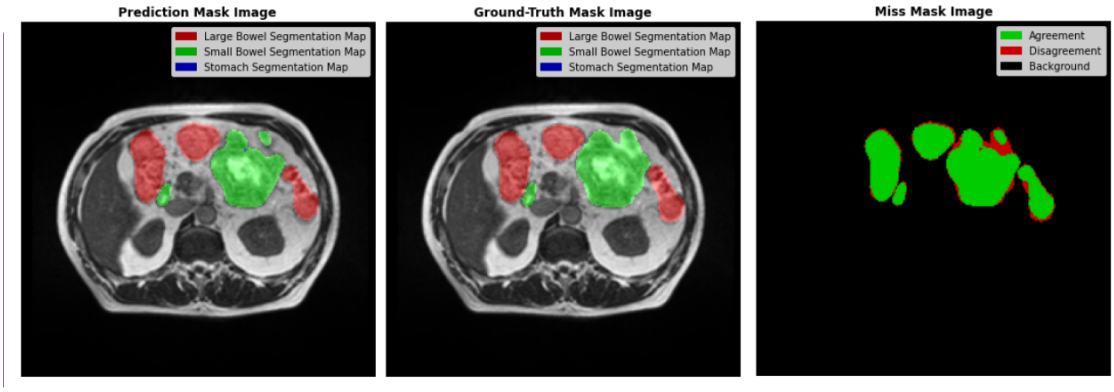

Рисунок 3.7. Демонстрація результатів сегментації. Зліва – результат сегментації, по середині – істинна анотація, справа – зеленим кольором позначено співпадіння між істиною та сегментацією.

В даному під-розділі наведено результати аналізу впливу запропонованих гіперпараметрів – коефіцієнту розширення (R), та глибини ($\boldsymbol{\delta}$) у порівнянні з базовою архітектурою U-Net.

Розглядалися варіанти з поглибленням мережі до $\boldsymbol{\delta = 6}$ та $\boldsymbol{\delta = 7}$. Також було розглянуто архітектури, які мають кількість ядер (фільтрів) першого модуля шифрувальника $\boldsymbol{C_0} = 64$ та $\boldsymbol{C_0} = 32$. Параметри R були підібрані таким чином, аби для двох архітектур з різною глибиною загальна кількість параметрів була приблизно однаковою.

Для аналізу стабільності, було проведено експерименти з К-кратною перехресною перевіркою.

Експерименти було проведено з використанням середовища Kaggle IPython Notebook з GPU – Nvidia P100 (16GB VRAM)

В якості оптимізатора було обрано ADAM [85] з початковим коефіцієнтом навчання 0.0001. В ході тренування, якщо одна з заданих метрик не показала тенденції до покращення (наприклад, функція втрат на валідаційному наборі) протягом 2 епох – крок навчання зменшується на 0.5. Якщо покращення не відбулося незалежно від зменшення коефіцієнта протягом більшого періоду 4 – навчання припиняється;

Метрикою перевірки було обрано коефіцієнт Дайса [86] – одна з поширених метрик серед методів сегментації зображень.

$$\mathrm{DS} = \frac{2 \cdot \sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \tag{3.1}$$

Де p_i- множина пікселів $p_i^{(x,y)} \in [0,1]$, отриманих нейронною мережею в режимі перевірки для і-того зображення з набору даних, g_i – множина пікселів з оригінальної анотації $g_i^{(x,y)} \in [0,1]$ для і-ого зображення. Таким чином значення чисельника та знаменника даної метрики обчислювалося сумарно по усім зображенням з набору даних. Даний підхід було використано в зв'язку з тим, що набір даних містив дуже велику кількість «пустих» зображень – тобто таких, де відсутні анотовані дані. Відповідно, в разі використання коефіцієнту Дайса як середньому по усім зображенням $mDS = \frac{1}{N}\sum_i DS(p_i, g_i)$ призвело б до великої кількості результатів, для яких $DS(p_i, g_i) = 1$, що в свою чергу впливало на остаточний результат і «приглушувало» б вплив більш значимих зображень.

Таблиця 3.2. Результати в залежності від глибини та коефіцієнту розширення нейронної мережі (для набору даних UMWGIT). DS– середнє значення коефіцієнту Дайса для K-их мереж по тестовому наборі, в дужках – стандартне відхилення.

C_0	R	δ	К-сть параметрів	DS
64*	2*	5*	31M	0.701(±0.003)
32	1.5	7	12.8M	0.666(±0.01)

64	1.52	6	13.4M	0.678(±0.008)
64	1.4	7	14M	0.69 (±0.01)
64	1.56	6	16.4M	0.684 (±0.008)
64	1.6	6	20M	0.688(±0.004)
32	1.8	6	20M	0.687(±0.009)
64	1.5	7	27.5M	0.702 (±0.006)
64	1.52	7	31M	0.703 (±0.007)
64	1.7	6	32.4M	0.697 (±0.001)
32	2	6	31M	0.674(±0.01)
64	1.56	7	40M	0.705(±0.004)
64	1.75	6	41M	0.696 (±0.005)
64	1.6	7	51M	0.715(±0.0005)
64	1.8	6	51M	0.704 (±0.002)

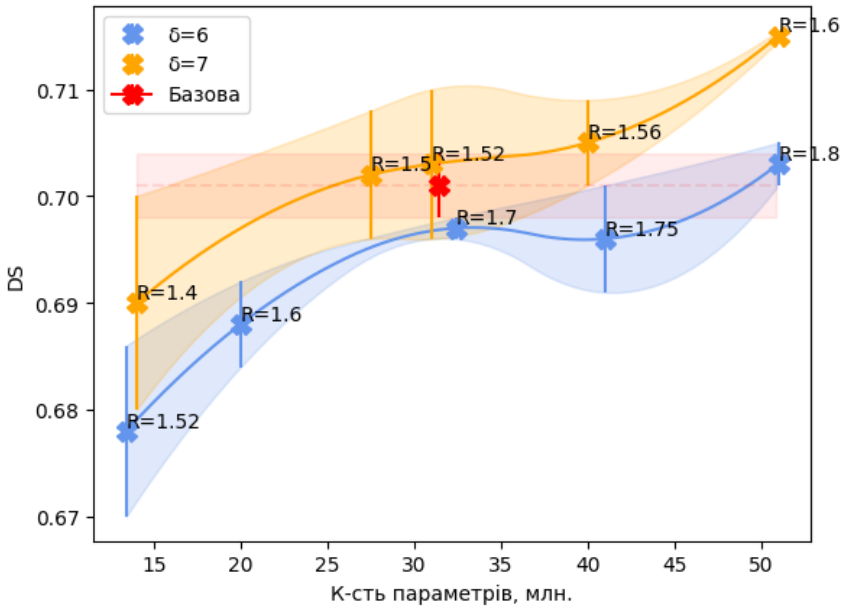

Рисунок 3.8. Графік залежності метрики перевірки (коефіцієнту Дайса), від кількості параметрів архітектури, на основі результатів перехресної перевірки. Червоним кольором позначено базову архітектуру U-Net та розподіл результатів на основі перехресної перевірки для кращого розуміння результатів.

3.3.2. Результати ансамблів мереж з K-кратної перехресної перевірки для способу підбору коефіцієнта розширення.

В рамках додаткових експериментів, було застосовано метод ансамблю нейронних мереж [87] [54], де використовувалися натреновані у рамках попереднього експерименту з K-кратною перехресною перевіркою нейронні мережі. Якщо представити мережу як функцію $g(\theta_k, x): X \to Y$, (де θ_k – ваги нейронної мережі, натреновані на групі з порядковим номером k), результат сегментації від ансамблю мереж як $y(x): X \to Y$, то цей метод можна описати наступним чином:

$$y(x) = \frac{1}{K} \cdot \sigma\left(\sum_{k=1}^{K} g(\theta_k, x)\right)_i \tag{3.2}$$

Де – $\sigma(\cdot)_i$ – softmax функція, описана формулою:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j^K e^{z_j}} \tag{3.3}$$

Де z – це вектор коефіцієнтів класифікації $z \in \mathbb{R}^K$, K – кількість класів класифікації, i – індекс класу у векторі коефіцієнтів.

У випадку, коли вихідними даними є матриці, тобто $y(x) \in \mathbb{R}^{m_1 \times \cdots \times m_n \times K}$, сумація відбувається по кожному елементу. На практиці, наприклад, в задачі сегментації, коли результат має розмірність $W \times H \times C$, де C – кількість класів класифікації і розмірність вектору коефіцієнтів класифікації, це означає, що ці підсумовуються для кожного пікселя зображення і виводиться їх середнє значення.

Метою даного експерименту було покращити результати сегментації, а також перевірити, чи дозволяє збільшення кількості параметрів також покращити результат ансамблів за межами стандартного відхилення отриманих у результаті K-кратної перехресної перевірки, і чи зберігається співвідношення результатів одиночних мереж, та їх ансамблів в залежності від кількості параметрів.

Даний експеримент було виконано у середовищі, аналогічному з попереднім. Оскільки для перехресної перевірки використовувалася кількість відрізків K=4, то в даному експерименті відображатимуться результати для K=4.

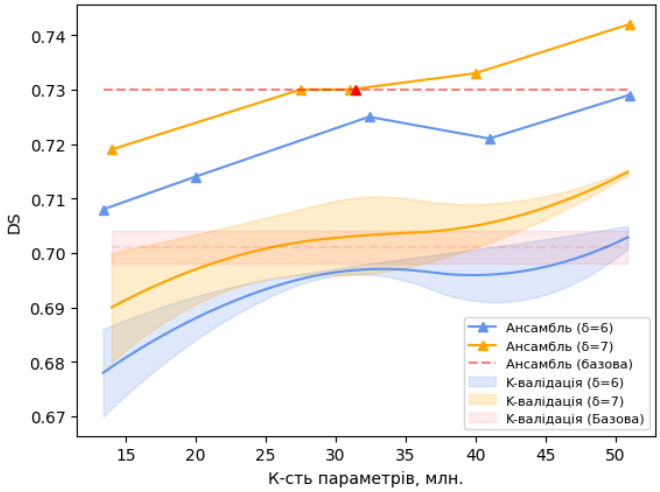

Рисунок 3.9. Графік залежності метрики перевірки (коефіцієнту Дайса) для Ансамблів К-перехресної перевірки, від кількості параметрів архітектури. Для демонстрації покращення, результати перехресної перевірки також наведені на графіку.

Таблиця 3.3. Результати в залежності від глибини та коефіцієнту розширення для ансамблів К-кратної перевірки нейронної мережі (для набору UMWGIT). DS – коефіцієнт Дайса.

C_0	R	δ	К-сть параметрів, млн	DS
64*	2*	5*	31M	0.73
64	1.52	6	13.4M	0.708
64	1.4	7	14M	0.719
64	1.56	6	16.4M	0.712
64	1.6	6	20M	0.714
32	1.8	6	20M	0.713
64	1.5	7	27.5M	0.73
64	1.52	7	31M	0.73
64	1.7	6	32.4M	0.725
32	2	6	31M	0.702
64	1.56	7	40M	0.733
64	1.75	6	40M	0.721
64	1.8	6	51M	0. 729
64	1.6	7	51M	**0.742**

3.4. Результати способу глибинних залишкових проміжних зв’язків.

3.4.1. Експеримент з перехресною перевіркою на наборі даних UMWGIT.

Дані експерименти були проведені на декількох різних наборах даних. Для набору UMWGIT було проведено K-кратну перехресну перевірку для розуміння стабільності використання способу глибинних залишкових проміжних зв’язків (англ. *Depthwise-Separable Skip Connections, DeSSCo*). Також в експериментах використовувалися декілька різних модифікацій DeSSCo.

Основні метрики експерименту – середнє значення коефіцієнта Дайса та його стандартне відхилення серед K мереж (K=4) на тестовому наборі. Рисунок 3.10. візуалізує значення кожної з окремих нейронних мереж на відповідному k-ому відрізку з тренувального набору, а також середнє значення по усім моделям для кожній з варіацій запропонованого способу. Даний графік дозволяє розуміти поведінку кожної модифікації на конкретному k-ому відрізку тестового набору, а також стабільність передбачень нейронних мереж загалом.

Таблиця 3.4 містить результати експериментів, а також результати коефіцієнту Дайса по окремим класам.

Окремим експериментом було отримано значення для відповідних нейронних мереж на повному наборі даних, тобто для тренування використовувався весь тестовий набір. Результати цих експериментів наведено у Таблиця 3.5. Дані результати важливі для розуміння важливості впливу розмірів набору даних на остаточні результати перевірки.

Решта параметрів (середовище виконання, оптимізатор тощо) аналогічні з наведеними у експерименті вище.

Таблиця 3.4. Результати експериментів на наборі даних UMWGIT з перехресною перевіркою. Значення в дужках – стандартне відхилення результатів окремих нейронних мереж.

Архітектура	**К-сть параметрів**	**DS (загальний)**	**DS по категоріям**		
			Шлунок	**Нижній кишківник**	**Верхній Кишківник**
U-Net	31.4	0.701 (±0.003)	0.797 (±0.007)	0.736 (±0.007)	0.605 (±0.006)
DeSSCo(C)	31.7	0.694 (±0.022)	0.79 (±0.019)	0.73 (±0.017)	0.591 (±0.04)
DeSSCo(2C)	35.7	0.713 (±0.007)	0.817 (±0.008)	0.747 (±0.004)	0.61 (±0.011)
DeSSCoB	32.8	0.711 (±0.003)	0.812 (±0.01)	0.745 (±0.004)	0.61 (±0.011)

Таблиця 3.5. Результати експериментів на наборі даних UMWGIT з повним тренувальним набором. DS – Коефіцієнт Дайса

Архітектура	**К-сть параметрів**	**DS (загальний)**	**DS по категоріям**		
			Шлунок	**Нижній кишківник**	**Верхній Кишківник**
U-Net	31.4	0.718	0.829	0.75	0.617
DeSSCo(C)	31.7	0.727	0.827	0.759	0.632
DeSSCo (2C)	35.7	0.722	0.814	0.753	0.631
DeSSCoB	32.8	0.727	0.830	0.756	0.63

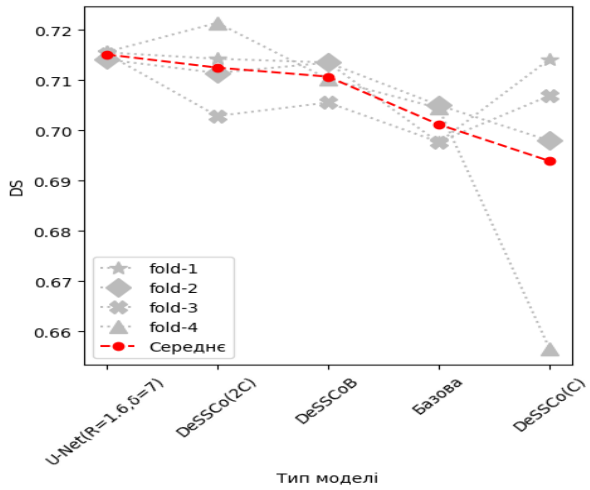

Рисунок 3.10. Середнє значення кожної з окремих нейронних мереж, та значення по кожній з K-их мереж

3.4.2. Експерименти з набором даних CityScapes.

CityScapes є зручним набором даних для порівняння, оскільки він містить заздалегідь визначений набір тестувальних (2975 знімків) та валідаційних даних (500 знімків).

Для порівняння з іншими методами, які не базуються на архітектурах U-Net, було також реалізовано архітектуру DeepLabV3+, що використовує архітектуру ResNet101 в якості опорної (backbone) мережі з вагами, що були ініційовані випадковим чином. Такий варіант DeepLabV3+ було обрано через те, що він має приблизно однакову кількість параметрів (31 млн.), як і базова архітектура U-Net . Мережа DeepLabV3+ показала найкращі результати для даного набору в оригінальному дослідженні [26], і є однією з найпоширеніших нейронних мереж у літературі, однак відтворення результатів експериментів є доволі нетривіальною задачею, яка також включає попереднє тренування нейронної мережі на різних наборах даних, таких як ImageNet, додаткові модифікації набору та велику кількість додаткових гіперпараметрів та модифікацій алгоритму тренування. В рамках експериментального дослідження, було вирішено використати найпростіші підходи та засоби, та натренувати нейронні мережі протягом короткого часу.

Оригінальний набір містить знімки розмірністю 2048x1028, однак для пришвидшення експериментів, зображення було стиснуто до розмірів 512x256 пікселів.

Експерименти було проведено з використанням GPU – Nvidia V100 (16GB VRAM) в середовищі Colab PRO+ IPython Notebook.

Основною метрикою перевірки для даного експерименту є середнє пересічення над об'єднанням (англ. *mean intersection over union, mIoU*). Нехай K – кількість класів, що наявні у наборі даних, N – кількість зображень у валідаційному наборі даних, $p_j^{(k)}$ – маска сегментації для k-ого класу, отримана нейронною мережею, $g_j^{(k)}$ – очікувана маска сегментації для k-ого класу, тоді:

$$\text{mIoU} = \frac{1}{K}\sum_{k=1}^{K}\frac{\sum_j^N p_j^{(k)} g_j^{(k)}}{\sum_j^N p_j^{(k)} + \sum_j^N g_j^{(k)}} \tag{3.4}$$

Тренування нейронної мережі займає 50 епох. В процесі тренування використовувався оптимізатор ADAM з початковим коефіцієнтом навчання 0.001, котрий зменшував у процесі навчання за наступною формулою:

$$lr = lr_0 - i \cdot \frac{lr_0 - lr_{max}}{i_{max}} \tag{3.5}$$

Де $lr_0 = 10^{-3}$ – початковий крок навчання, $lr_{max} = 10^{-6}$ – кінцевий крок навчання, i – поточна епоха, $i_{max} = 50$ – загальна кількість епох.

Як було зазначено вище, до уваги у перевірки береться 20 основних класів. Для кожного класу вираховується значення IoU (наведені на Рисунок 3.11 та Рисунок 3.12)

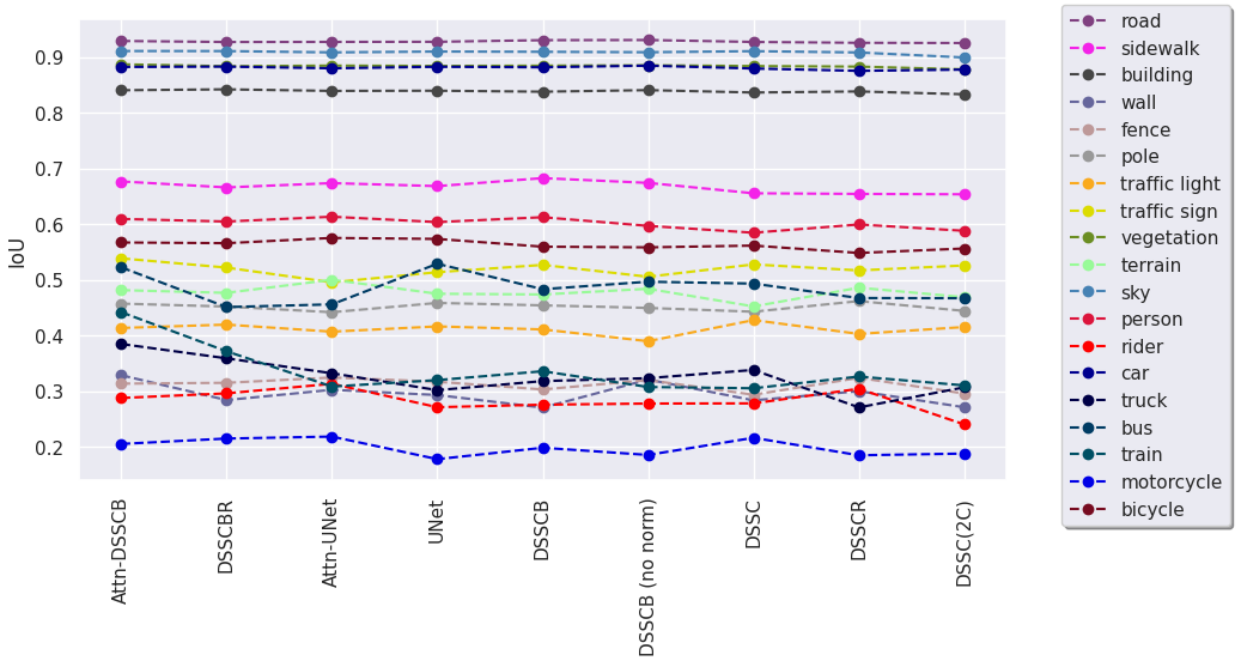

Рисунок 3.11. Покласове значення метрик IoU та коефіцієнта Дайса для набору даних CityScapes

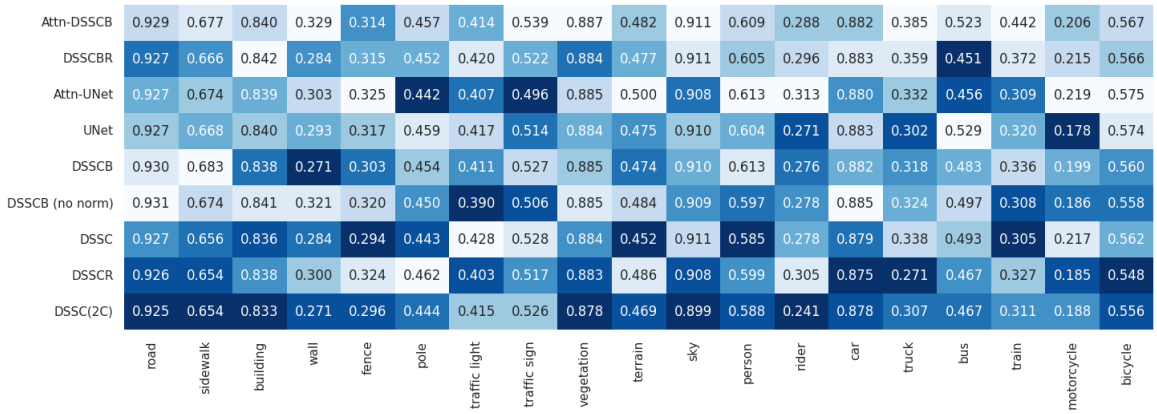

Рисунок 3.12. Таблиця значень метрики IoU по кожному з класів. Світлішим кольором показані найкращі результати серед усіх архітектур, темнішим – найгірші

Остаточні значення метрик перевірки запропонованих модифікацій та базових архітектур наведені у Таблиця 3.6

Таблиця 3.6. Узагальнені результати по усім архітектурам, що були натреновані на наборі даних CityScapes.

Архітектура	**К-сть параметрів**	**mIoU**
U-Net	31.4	0.5444

Att.U-Net	33.1	0.5475
DeSSCo-Attn	33.5	0.5452
DeepLabV3+ resnet101	31.1	0.4436
DeSSCo	31.7	0.5421
DeSSCoR	31.7	0.5410
DeSSCoB	32.8	0.5471
DeSSCoB(NoNorm)	32.8	0.5444
DeSSCo(2C)	35.4	0.5330
DeSSCoBR	32.8	0.5486
DeSSCoB-Att	34.5	0.5621

3.4.3. Експерименти з набором даних Synapse.

Дані експерименти були проведені на наборі Synapse [45]. Експериментальна конфігурація повторює конфігурацію з роботи Trans-U-Net [40] та Swin-U-Net [41], реалізовану з бібліотекою TensorFlow, а саме:

- Набір розбито на тренувальний та валідаційний набори, з 18 та 12 МРТ знімків відповідно;
- З кожного МРТ знімку виокремлено зображення - поперечні зрізи, стиснуті до розмірності 224x224 пікселі;
- Усі пікселі було приведено до значень $[m^-, m^+] = [-125, 275]$, а потім нормалізовано до проміжку значень [0, 1]

$$x' = \frac{\max(m^-, \min(m^+, x)) - m^-}{m^+ - m^-} \tag{3.6}$$

- До кожного знімку та його анотації у тренувальному наборі випадковим чином застосовувалися трансформації повороту на випадковий кут у діапазоні від -20 до 20 градусів, а також поворот на 90 градусів у випадкову сторону;

- В якості анотацій було використано 8 з 13 класів, наявних у наборі, а саме – права нирка, ліва нирка, шлунок, селезінка, аорта, підшлункова залоза, жовчний міхур та печінка;
- В якості метрик перевірки було використано середнє значення коефіцієнту Дайса по всім класам по всім МРТ з валідаційного набору

$$DS_{mean} = \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K} DS\left(\left(Y_p\right)_{ik}, \left(Y_t\right)_{ik}\right) \quad (3.7)$$

З відмінного, було змінено функцію втрат на функцію бінарної крос-ентропії, а також використано оптимізатор ADAM [85]. Також було застосовано стратегію вибору кроку навчання з використанням формули (3.3).

Таблиця 3.7. Порівняння результатів експериментів з результатами інших архітектур, на основі проведених іншими дослідниками експериментів (* - перетренована базова U-Net на заданому середовищі – для порівняння)

Архітектура	**DS**	**DS – по категоріям**							
		Аорта	**Жовчний міхур**	**Л. Нирка**	**П.Нирка**	**Печінка**	**Підшлункова залоза**	**Селезінка**	**Шлунок**
Результати, отримані іншими дослідниками									
V-Net	0.6881	0.7534	0.5187	0.771	**0.8075**	0.8784	0.4005	0.8056	0.5698
DARR [88]	0.6977	0.7474	0.5377	0.7231	0.7324	0.9408	0.5418	0.8990	0.4596
R50 U-Net	0.7468	0.8774	0.6366	0.8060	0.7819	0.9374	0.5690	0.8587	0.7558
U-Net	0.7685	0.8907	**0.6972**	0.7777	0.6860	0.9343	0.5398	0.8667	0.7558
R50 Att-UNet	0.7557	0.5592	0.6391	0.7920	0.7271	0.9356	0.4937	0.8719	0.7495
Att-Unet	0.7777	**0.8955**	0.6888	0.7798	0.7111	0.9357	0.5804	0.8730	0.7575
R50 ViT	0.7129	0.7373	0.5513	0.7580	0.7220	0.9151	0.4599	0.8199	0.7395
TransUnet	0.7748	0.8723	0.6313	0.8187	0.7702	0.9408	0.5586	0.8508	0.7562
SwinUnet	0.7913	0.8547	0.6653	0.8328	0.7961	0.9429	0.5658	**0.9066**	0.7660

Результати експериментів, отриманих запропонованими модифікаціями									
U-Net *	0.7657	0.8765	0.5894	0.8194	0.7767	0.9405	0.4973	0.8846	0.7413
DeSSCoBR	0.7781	0.8636	0.5513	**0.8544**	0.7821	0.9398	0.5625	0.8923	0.7793
DeSSCoB	0.7521	0.8878	0.5927	0.7531	0.6865	0.9102	**0.5947**	0.8596	0.7323
DeSSCo	0.7638	0.8896	0.6234	0.7618	0.6912	0.9345	0.5817	0.8404	**0.7881**
DeSSCoR	0.7669	0.8763	0.6097	0.8206	0.7675	0.9349	0.5064	0.8714	0.7485
DeSSCo(2C)	0.7674	0.8933	0.5837	0.8215	0.7710	0.9364	0.5332	0.8609	0.7359
DeSSCoBR -Attn	0.7699	0.8921	0.5509	0.8245	0.7747	0.9416	0.5269	0.8788	0.7696
DeSSCo-Attn	0.7747	0.8914	0.6037	0.7849	0.7548	**0.9430**	0.5876	0.8821	0.7501

Результати наведено у Таблиця 3.7. В даній таблиці наведено як результати, отримані іншими дослідниками, так і отримані з використанням запропонованого способу модифікації нейронних мереж. Оскільки використане експериментальне середовище дещо відрізняється від середовища, використаного іншими методами (різниця в оптимізаторі, обладнані, фреймворку), було також натреновано базову архітектуру U-Net, аби показати, що отримане середовище дозволяє отримати схожі результати, що дозволить порівняти запропонований метод із існуючими.

Експерименти було проведено з використанням наступного середовища:

- GPU – Nvidia V100 (16GB VRAM)
- Colab PRO+ IPython Notebook

3.4.4. Експерименти з набором даних BraTS.

Методологія даних експериментів базується на напрацюваннях учасників конкурсу BraTS попередніх років [89] [90].

Складність задачі виявлення пухлин в тому, що їх розташування та форма може бути різною, тому розміри та різноманіття набору даних відіграє важливу роль, аби уникнути перенавчання нейронної мережі. Для цього у дослідженнях [89]

[90] пропонуються наступні модифікації, котрі було вирішено використати у експериментальній частині.

Об'єднання багатомодальних даних. В оригінальному наборі даних [48] для кожного пацієнта було отримано 4 окремих МРТ знімків, отримані за допомогою процедур T1-weighted (T1), post-contrast T1-weighted (T1ce), T2-weighted (T2), and T2 Fluid-Attenuated-Inversion-Recovery (FLAIR). Кожна з даних процедур використовується для виявлення різних видів патологій тканин, що в свою чергу дозволяє спеціалістам прийняти рішення щодо ідентифікації даної тканини як ракової пухлини. В рамках попередньої обробки даних, усі 4 представлення розмірністю 240×240×155 поєднуються в одне, утворюючи таким чином чотиривимірний об'єкт розмірності 240×240×155×4, тому усі нейронні мережі, натреновані в ході експериментів, приймають об'єкти з вхідною кількістю каналів 4, замість традиційного для МРТ одного каналу.

Попередня модифікація даних – доволі великий об'єм інформації у вхідних даних займають точки без жодної інформації (що виражається чисельно як 0 у тривимірному масиві, що представляють чорно-білий МРТ знімок), розташовані по краях об'ємів, що отримуються в результаті томографії. Усі об'єми було обрізано до найближчих ненульових значень у масиві, а потім рівномірно розширено до таких розмірів, аби усі виміри ділилися на 8 – це зроблено з метою уникнення проблеми розбіжності розмірності карт ознак при операціях розширення з інтерполяцією (Upsampling). Для тренувального набору, з отриманих об'ємів випадковим чином виокремлюється об'єм 128×128×128, для тестового набору ці об'єми подаються на вхід нейронної мережі незмінними.

Окрім цього, відбувається відкидання значень 1-го та 99-ого персентилю, і після цього відбувається по канальна нормалізація значень з використанням формули (3.6).

В рамках попередній модифікацій тестових даних, також виконуються наступні маніпуляції:

- У випадковому порядку виконується поворот на 90, 180 або 270 градусів навколо 3-ох осей, а також дзеркальне відображення по 1, 2 чи 3-ох осях;

- З ймовірністю 80% додається шум до кожної точки простору (x,y,z) кожного k-ого представлення $M_k = \{T1,\ T1ce, T2, FLAIR\}_k$. Нехай N - функція вибірки числа з нормального розподілу, U – функція вибірки числа з рівномірного розподілу, $x_k^{x,y,z}$ – значення k-ого представлення в точці простору (x,y,z), x_k – вхідна карта ознак для k-ого представлення, $\sigma(X)$ – функція обчислення стандартного відхилення. Тоді формула нормалізації матиме наступний вигляд:

$$x_k^{x,y,z} = x_k^{x,y,z} + N\left(0, 0.1 \cdot \sigma\left(x_k \cdot U(0.9,1.1)\right)\right) \quad (3.8)$$

- З ймовірністю 20% один з чотирьох вхідних каналів повністю замінюється нулями

Архітектурні модифікації – оскільки при аналізі тривимірних зображень розмір вхідних даних на порядок перевищує розмір двовимірних зображень, при обмежених ресурсах VRAM розмір тренувальної ітерації зазвичай обмежений 1 об'єктом, такі шари нейронної мережі, як нормалізація вибіки (англ. *Batch Normalization*) [10], втрачають свій сенс. Саме тому серед існуючих методів замість шарів Batch Normalization використовується групова нормалізація (англ. *Group Normalization*) [91], що виконує нормалізацію серед груп каналів в рамках однієї карти властивості, розмір яких задано параметрично (у випадку даного експерименту - 16). Групову нормалізацію з метою експерименту було також інтегровано в базову U-Net архітектуру, та в усі запропоновані похідні архітектури (DeSSCo, DeSSCoB, тощо). По аналогії з дослідженнями [89], було прийнято рішення обмежити глибину мережі 3 модулями шифрування, шириною першого шару 48 та коефіцієнтом розширення 2.

Деталі реалізації глибинного згорткового шару. У зв'язку з програмною реалізацією бібліотеки Keras, що використовується TensorFlow для визначення архітектури нейронної мережі, глибинний згортковий шар (англ. *Depthwise Convolution*) для тривимірних мереж не було реалізовано в базовій бібліотеці, оскільки дані шари використовуються доволі нечасто. Для повноцінної інтеграції глибинного згортування у Keras, що була б оптимальною з точки зору часу виконання та ресурсів, необхідно реалізувати дану операцію на нижчому рівні API

бібліотеки з використанням мови програмування C++ та скомпілювати модифіковану версію бібліотеки Keras. Такі маніпуляції є трудомісткими та вимагають додаткової експертизи. На щастя, дане обмеження можливо обійти, використавши звичайні тривимірні згорткові шари Conv3D, використовуючи групування фільтрів. API бібліотеки Keras keras.layers.Conv3D дозволяє задати для шару Conv3D параметр groups. Якщо параметр groups буде ідентичним параметру filters, даний шар буде функціонувати як глибинний згортковий шар:

```
        dconv = layers.Conv3D(filters, 3,
                              groups=filters,
                              padding='same', name=f'…')
```

Дана реалізація дозволяє використовувати модулі DeSSCo для тривимірних архітектур нейронних мереж.

В якості метрик перевірки в рамках змагання [48] обрано коефіцієнт Дайса [86], і відстань Хаусдорфа [92] по 3-ом окремим підкласам: Enhancing Tumor (ET) – розширення пухлини, Tumor Core (TC) – ядро пухлини, та Whole tumor (WT) – уся пухлина. Ці класи дещо відрізняються від класів, що сегментовані в анотаціях для тренування (збільшення пухлини (ET), перитуморальний набряк (ED) та некротичне ядро пухлини без збільшення (NCR/NET)), та вираховуються відповідно як $TC = ET \cup NCR$, та $WT = ET \cup NCR \cup ED$. Через це, замість традиційного softmax шару активації в якості останнього шару активації, що для кожного точки повертає вектор довжиною K+1 (K – кількість класів) та сума усіх елементів цього вектору становить 1, було використано шар активації sigmoid $\sigma(x) = \frac{e^x}{1+e^x}$, де $\sigma(x) \in [0, 1)$, що повертає K-елементів – у випадку даного набору, 3. Саме ці дані передаються в функцію втрат.

Відстань (дистанція) Хаусдорфа [92] є величиною, яку використовують для оцінки різниці між двома контурами. У контексті сегментації, її використовують для порівняння між очікуваною маскою сегментації та маскою сегментації, отриманою з використанням нейронної мережі. Відстань Хаусдорфа, описана рівнянням (3.8), визначає найбільшу евклідову відстань між двома точками, що належать порівнюваним контурам $A = \{a_1, a_2, \dots, a_q\}$ та $B = \{b_1, b_2, \dots, b_q\}$:

$$h(A,B) = \max_{a \in A} \min_{b \in B} ||a - b|| \quad (3.9)$$

$$H(A,B) = \max\big(h(A,B), h(B,A)\big) \quad (3.10)$$

Результати наведено у Таблиця 3.8 та Таблиця 3.9. Варто зазначити, що більшість архітектур з даних таблиць використовують Group Normalization на рівні шифрувальника, дешифрувальника та модифікацій проміжних зв'язків. Дужки біля назви архітектури позначають ті архітектури, де використовується інший тип нормалізації лише на модифікаціях проміжних зв'язків – тобто це можуть бути або шари Group Normalization, або Instance Normalization, або без нормалізації взагалі.

Експерименти було проведено з використанням середовища Colab PRO+ IPython Notebook та графічних процесорів Nvidia V100 (16GB VRAM), Nvidia A100 (40GB VRAM);

Таблиця 3.8. Значення коефіцієнту Дайса для запропонованих модифікацій.

Архітектура	**DS**	**DS – по категоріям**		
		ET	**TC**	**WT**
U-Net	0.8009	0.7246	0.8059	0.8724
DeSSCo	0.8015	0.7187	0.8035	0.8823
DeSSCo (Instance Normalization)	0.7676	0.7176	0.7676	0.8178
DeSSCo(2C)	0.7886	0.7153	0.7899	0.8617
DeSSCo (без нормалізації)	0.8057	0.7336	**0.8089**	0.8747
DeSSCoB	0.7931	0.7325	0.7741	0.8696
DeSSCoBR	0.7995	0.7246	0.7992	0.8749
DeSSCo-Attn	0.7893	0.7259	0.7919	0.8501
DeSSCoR	0.7874	0.7174	0.7898	0.8551
DeSSCoB (Instance Normalization)	0.7943	**0.7402**	0.7902	0.8526
DeSSCoB (без нормалізації)	0.7909	0.7194	0.7756	0.**8775**

Таблиця 3.9. Значення дистанції Хаусдорфа для запропонованих архітектур нейронних мереж.

Архітектура	**Дистанція Хаусдорфа – по категоріям**		
	ET	**TC**	**WT**
U-Net	9.915	10.79	**20.73**
DeSSCo	**7.93**	9.99	106.16
DeSSCo (Instance Normalization)	13.83	12.16	58.86
DeSSCo(2C)	15.61	17.94	32.60
DeSSCo (без нормалізації)	11.835	12.078	121.52
DeSSCoBR	9.59	12.21	26.71
DeSSCoB	13.47	14.28	31.40
DeSSCo-Attn	17.24	18.08	55.67
DeSSCoR	12.25	10.786	118.81
DeSSCoB (Instance Normalization)	8.5362	**8.419**	24.46
DeSSCoB (без нормалізації)	9.707	12.576	30.4769

3.5. Результати гібридного способу глибинних роздільних проміжних зв'язків з варіацією коефіцієнта розширення.

Комбінації різних модифікацій архітектури часто можуть призводити до покращення результатів. В рамках додаткових досліджень, було поставлено за мету перевірити можливість комбінації кількісних покращень (на прикладі запропонованого в даному дослідженні коефіцієнту розширення) та якісних – з використанням глибинних роздільних проміжних зв'язків. Задля даного експерименту, розглядалися архітектури з $R = 1.6$, $\boldsymbol{\delta = 7}$ (що містить 51 млн. параметрів) та з $R = 1.4$, $\boldsymbol{\delta = 7}$ **(**14 млн. параметрів **)**. Згідно з Таблиця 3.2 мережа R=1.4, при розмірі в більш ніж 2 рази менше ніж базова архітектура, показала результат DS=0.69 при перехресній перевірці, а архітектура з R=1.6 (в 1.8 рази більша за базову) – DS=0.714. В той же час, базова мережа показала DS=0.70).

Поєднавши меншу архітектуру з модулями DeSSCo, точність метрик перевірки отриманої мережі очікується на рівні базової архітектури, тоді як у поєднанні з великою мережею, результат повинен покращитися. Дані експериментальні мережі було натреновано на повному тренувальному наборі даних UWGIT.

Окремо також було натреновано декілька додаткових нейронних мереж на наборі даних CityScapes. Аналогічно з попереднім експериментом, було обрано архітектуру з параметрами R = 1.4 $\boldsymbol{\delta = 7}$ та її комбінації з модулями DeSSCo і DeSSCoB. Параметри цього експерименту аналогічні до описаного в секції 3.5.2. Результати наведені у Таблиця 3.11.

Ще один експеримент було проведено для набору даних Synapse. Через те, що було обрано мережу з глибиною $\boldsymbol{\delta = 7}$ (в зв'язку з обмеженням, що залежить від глибини мережі, описаним формулою 2.8) нейронні мережі для цього експерименту були натреновані на зображеннях розміром 256×256 (замість 224×224, результати для яких наведено в Таблиця 3.7). Оскільки в попередніх експериментах результати базової архітектури наведено для архітектури, що приймає на вхід зображення розмірності 224×224, окремо від результату на зображеннях 224×224, була також натренована базова архітектура U-Net на зображеннях 256×256. Решта параметрів експерименту, а також методологія його проведення, аналогічні до експерименту, описаного в підпункті 3.4.3. Результати цього експерименту наведені у Таблиця 3.12.

Таблиця 3.10. Комбінація модуля DeSSCo(C) та U-Net з різними коефіцієнтами розширення на наборі даних UWGIT. Для архітектур з коефіцієнтом розширення вигористовується глибина мережі $\boldsymbol{\delta = 7}$. * - результат базової архітектури.

Архітектура	**К-сть параметрів, млн**	**DS**
U-Net*	31.4	0.718
$R = 1.4$	14.4	0.719
$R = 1.6$	51.4	0.723
$R = 1.6$ + **DSSC(C)**	52.2	0.732
$R = 1.4$ + **DSSC(C)**	14.7	0.727

Таблиця 3.11. Результати гібридного способу модифікації з використанням комбінації модулів DeSSCo та способу варіаціє коефіцієнту розширення для набору даних CityScapes.

Архітектура	К-сть параметрів, млн	mIoU
U-Net*	31.4	0.5444
R = 1.4 $\boldsymbol{\delta = 7}$	14.4	0.5441
R = 1.4 $\boldsymbol{\delta = 7}$ + **DSSC**	14.7	0.5338
R = 1.4 $\boldsymbol{\delta = 7}$ + **DSSCB**	15.4	0.5368

Таблиця 3.12. Результати гібридного способу модифікації з використанням комбінації модулів DeSSCo та способу варіаціє коефіцієнту розширення для набору даних Synapse. * - результат, узятий з Таблиця 3.7, ** - результат для базової архітектури, натренованої на зображеннях з розширенням 256×256

Архітектура	К-сть параметрів, млн	DS
U-Net*	31.4	0.7654
U-Net**	31.4	0.7587
R = 1.4 $\boldsymbol{\delta = 7}$	14.4	0.7690
R = 1.6 $\boldsymbol{\delta = 7}$	51.4	0.7535
R = 1.4 $\boldsymbol{\delta = 7}$ + **DSSC**	14.7	0.7761
R = 1.4 $\boldsymbol{\delta = 7}$ + **DSSCB**	15.4	0.7806

Висновки до розділу 3

В даному розділі наведено практичні результати дисертаційного дослідження з використанням запропонованих модифікацій архітектури U-Net.

Для експериментів було обрано 4 набори даних – UWGIT, BraTS, Synapse, що представляють доменну область аналізу медичних зображень, а також набір CityScapes, що репрезентує задачу семантичної сегментації об’єктів у контексті міського середовища. Набір BraTS також є тривимірним, що дозволяє перевірити

ефективність запропонованих модифікацій також і на вирішення задачі сегментації об’ємів, а також діагностики захворювання – пухлин мозку.

Безпосередньо опису експериментів належить три підрозділи, присвячені запропонованим способам :

- Способу підбору коефіцієнту розширення R та глибини нейронної мережі, з проведеною К-кратною перевірки результатів на наборі даних UWGIT;
- Способу Глибинних Роздільних проміжних зв’язків у модифікаціях DeSSCo (C), DeSSCo(2C), DeSSCoB, модифікаціях з використанням залишкових зв’язків (DeSSCoBR, DeSSCoR, DeSSCoR(2C)), і їх використанні в поєднанні з архітектурами U-Net та Attention-UNet. Було проведено запропонованих модифікацій К-Кратну перевірку на наборі UWGIT, і отримано результати на наборах BraTS, Synapse, Cityscapes
- Гібридному способу, що поєднує у собі спосіб підбору коефіцієнту розширення та спосіб глибинних роздільних проміжних зв’язків;

Експерименти було виконано з використанням декількох наборів даних, що містять у собі фото у контексті дорожнього руху у міському середовищі (CityScapes), а також у контексті медичних зображень (Synapse, UWMGIT).

Для деяких експериментів було додатково використано методи попередньої модифікації даних, що були описані в інших досліджень, з метою отримання результатів, котрі можна було б порівняти з широковживаними методами, аби показати практичність запропонованих нововведень.

Експерименти були виконані як для двовимірних зображень, так і для тривимірних (у контексті медичних зображень, отриманих в результаті МРТ) аби показати практичність та корисність використання запропонованих модифікацій з метою покращення метрик перевірки.

Результати експериментів наведені у декількох таблицях та графіках, що дозволяє провести порівняльний аналіз запропонованих способів модифікації нейронної мережі U-Net, на основі яких можливо приймати рішення щодо доцільності використання тих чи інших модифікацій.

РОЗДІЛ 4
АНАЛІЗ РЕЗУЛЬТАТІВ ТА ДОДАТКОВІ ЕКСПРИМЕНТИ З МЕТОДОМ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МОДИФІКОВАНИХ АРХІТЕКТУР U-NET

4.1. Аналіз результатів способу підбору коефіцієнта розширення.

4.1.1. Аналіз результатів К-кратної перехресної перевірки

Використання коефіцієнта розширення було вмотивоване інтуїтивною ідеєю модифікувати використаний у оригінальній архітектурі U-Net підхід у збільшені глибини карти ознак у 2 разу у кожному наступному модулі шифрувальника. Такий підхід веде до значного приросту параметрів нейронної мережі при збільшені глибини цієї мережі. Наприклад, при збільшені глибини базової архітектури з 5 до 6 призводить до приросту кількості параметрів в 4 рази (32 млн до 128 млн.), що накладає певні обмеження на апаратні ресурси, а також може призвести до надлишковості архітектури. Завдяки коефіцієнту розширення, з'являється додаткова гнучкість для створення нейронних мереж, що дозволяє створювати глибші мережі, при меншому прирості кількості параметрів. Більше того, даний підхід також дає можливість зменшення розмірів мережі при збільшені її глибини, що також дозволяє теоретично компенсувати недостачу карт ознак вищого рівню завдяки глибшим картам ознак. Наприклад, мережа R=1.4 , $\boldsymbol{\delta = 7}$, згідно значенням коефіцієнту Дайса (*DS*) (Таблиця 3.2), досягає результатів, котрі можна порівняти з базовою архітектурою (*DS=0.69* проти *DS=0.701* на перехресній перевірці), маючи при цьому у 2 рази менше параметрів (14 млн, порівняно до 32 млн.). Серед більших нейронних мереж, R=1.6 , $\boldsymbol{\delta = 7}$ (51 млн. параметрів) досягла кращих результатів – *DS=0.715* проти *DS=0.701.*

Цікавим є спостереження, що мережі з глибиною 7 краще справляються із задачею, аніж мережі з глибиною 6 та аналогічною кількістю параметрів (що помітно на Рисунок 3.8). Так, серед нейронних мереж, розмірами у 51 млн.

параметрів, мережа з коефіцієнтом розширення 1.6 та глибиною 7 перевершила мережу з коефіцієнтом 1.75 та глибиною 6 (*DS=0.715* проти *DS=0.704*).

Ще одним спостереженням є те, що архітектури з початковою кількістю фільтрів $\boldsymbol{C_0 = 32}$ справляються з задачею сегментації значно гірше, ніж $\boldsymbol{C_0 = 64}$ з аналогічною кількістю параметрів. Так, архітектура $\boldsymbol{C_0 = 32, R = 2, \delta = 7}$ яка містить 31 млн. параметрів показала гірший результат (*DS=0.674*), ніж архітектура з аналогічною кількістю параметрів та глибиною $\boldsymbol{C_0 = 64, R = 1.7, \delta = 6}$ (*DS=0.697*), тому в подальшому в дослідженні використовувалися архітектури з кількістю фільтрів $\boldsymbol{C_0 = 64}$.

4.1.2 Аналіз результатів ансамблів К-кратної перехресної перевірки.

Ансамблі К-кратної перехресної перевірки виявилися доволі ефективним способом покращення фінальних результатів, не змінюючи архітектуру нейронної мережі. Комбінування результуючої сегментації декількох нейронних мереж, натренованих на К-різних варіацій набору даних, дозволило покращити (Таблиця 3.3) середнє значення коефіцієнту Дайса (*DS*) К-кратної перевірки на 4.7% для базової архітектури U-Net (*DS=0.733* у порівнянні з *DS= 0.7*). Для більших архітектур (таких як R=1.6, δ=7) даний результат збільшується на 6% (*DS= 0.742* до *DS= 0.7*). Показово (Рисунок 3.9), що ансамблі перевищили середні результати К-кратної перевірки для усіх запропонованих архітектур, перевищуючи результати кожної одиночної нейронної мережі, про що свідчить те, що результати ансамблів знаходяться вище за верхню межу стандартного відхилення.

Також, показовим є результат даного способу у порівнянні з результатами нейронних мереж, що були натреновані на усьому тренувальному наборі. У випадку з базовою U-Net та $\boldsymbol{R = 1.6\,, \delta = 7}$ значення коефіцієнта Дайса покращилося (*DS=0.733* проти *DS=0.718* та *DS=0.742* проти *DS=0.723*) у порівняні з окремими мережами, натренованими на усьому тренувальному наборі (див. Таблиця 4.1). Варто врахувати, для експериментів на наборі даних UWGIT тренувальні та валідаційні вибірки мають розміри 68 та 17 пацієнтів відповідно, у випадку тренування з використанням перехресної перевірки для кожної з К (в

даному випадку - 4) мереж використовується підмножина тренувального набору у 50 пацієнтів, тобто дані втрачається деяка варіативність тренувальних даних, через що архітектура, натренована на повному наборі, показує кращі результати, аніж кожна окрема з К-мереж, про що у Таблиця 4.1 свідчать середні результати К-кратної перевірки у порівнянні до результатів нейронних мереж, натренованих на повному наборі даних.

Таблиця 4.1. Порівняння коефіцієнту Дайса для різних архітектур та для різних способів тренування нейронних мереж

Архітектура	**Повний набір (67 пацієнтів)**	**К-кратна перевірки (K=4, по 50 пацієнтів)**	**Ансамбль (K=4, по 50 пацієнтів)**
U-Net	0.718	0.701 (±0.003)	0.73
$R = 1.4$	0.719	0.69 (±0.01)	0.719
$R = 1.6$	0.723	0.715(±0.0005)	0.742

Дані результати підтверджують теорію про те, що ансамблі К-кратної перехресної перевірки дозволяють компенсувати алеаторну невизначеність [93]. Варто зазначити, що при підборі кращої архітектури для даної задачі, або покращенню якості чи варіативності даних, різниця між результатами окремих нейронних мереж і ансамблю буде зменшуватися. Це було показано в одному з попередніх досліджень автора [52] на наборі даних HUBMAP [46] для сегментації тканин нирок (див. Таблиця 4.2), де ансамблі показали статистично ідентичні результати з окремими нейронними мережами. Що також показово, використані там модифікації вхідних даних (повороти та інверсія зображень), а також генерація зображень із зсувом (зображення для тренування отримані шляхом отримання зображень розмірами 1024 на 1024 пікселів із більшого зображення тканини із кроком у 512 пікселів, що збільшило у 4 рази варіативність тренувального набору порівнянні з базовим, де зображення отримувалися з кроком 1024) не змогли покращити результат. Це свідчить про те, що алеоторна невизначеність, зумовлена

варіативністю та якістю набору даних, в цьому випадку має мінімальний вплив на результат.

Таблиця 4.2. Результати для базової U-Net з дослідження [52]. TTA – попередня модифікація тренувальних даних (англ. *Train-Time-Augmentation*), DS - коефіцієнт Дайса

Архітектура	**DS (без TTA)**	**DS (з TTA)**
Повний набір	0.934	0.937
Ансамбль (K=4)	0.929	0.931
Ансамбль (K=6)	0.933	0.933

З цього можливо зробити висновок, шо за умови вдалого підбору архітектури мережі, використання повного набору даних іноді може бути більш доцільним, аніж цілеспрямоване тренування К нейронних мереж для способу ансамблів К-кратної перевірки, проте це жодним чином не виключає практичність даного способу в інших випадках, особливо тоді, коли виконується К-кратна перехресна перевірка. Саме практичність даного способу зумовила його розповсюдження серед «конкурсних» методів, мета яких – за будь-яку ціну збільшити фінальну метрику перевірки.

4.1.3. Аналіз впливу способу підбору коефіцієнту розширення для нейронних мереж для аналізу тривимірних зображень.

Окрім експериментів на двовимірних зображеннях, було також проведено невеликий експеримент для тривимірних архітектур, натренованих на наборі даних UWGIT, який було попередньо підготовлено шляхом створення тривимірних об’ємів, розмірністю 224 x 224 x 128, з двовимірних зображень, наведених в даному наборі. Через часові обмеження, а також необхідність використання великої кількості ресурсів (для тренування використовувався графічний прискорювач

NVIDIA A100), було отримано незначну кількість експериментальних даних, що наведені в Таблиця 4.3.

Таблиця 4.3. Результати для тривимірного набору UWGIT в залежності від коефіцієнту розширення та глибини мережі

Ширина першого шару – C_0	**Коефіцієнт розширення - R**	**Глибина мережі - δ**	**К-сть параметрів, млн.**	**DS**
32	2	5	23	0.778
32	1.6	6	15	0.754
32	1.7	6	23	0.776
32	1.8	6	38	0.770

З наведених результатів видно, що зміна коефіцієнту розширення і збільшення глибини не змогла покращити результати передбачень. Причиною цьому може бути замала розмірність вхідних даних, що призводить до втрати просторових ознак у ході збільшення кількості операцій узагальнення. Дана проблема, зокрема, зумовила те, що багато нейронних мереж, що брали участь у змаганнях BraTS, мали глибину мережі не більше 4 [90].

4.2. Аналіз впливу способу глибинних роздільних проміжних зв’язків на результати.

4.2.1. К-кратна перехресна перевірки.

Для визначення впливу модулів, що базуються на глибинних роздільних проміжних зв’язках, було проведено доволі велику кількість експериментів, що включали також в себе і експеримент з К-кратною перехресною перевіркою. Ідея експерименту з К-кратною перехресною перевіркою у необхідності підтвердження того, що саме запропоновані модулі покращують результат, а не випадкові процеси, що виникають в результаті тренування. Дані результати наведені в Таблиця 3.4.

Цей експеримент підтверджує, що середні результати з урахуванням стандартного відхилення для модуля DeSSCoB стабільно перевищують результати базової архітектури U-Net відповідно до значення коефіцієнту Дайса (DS) (*DS=0.701±0.003* проти *DS=0.711±0.003*).

Результати на графіку з Рисунок 3.10, а також Таблиця 3.5 показують деяку невідповідність між результатами К-кратної перевірки та мереж, натренованих на повному наборі даних. Так, хоча і модифікація DeSSCo(2C) показала дещо кращі результати, аніж навіть DeSSCoB (*DS=0.713±0.007*), на повному наборі даних вона уже показала гірший результат (*DS=0.722* до *DS= 0.727* для DeSSCo та DeSSCoB). Для модуля DeSSCo, результат на повному наборі даних виявився аналогічним до DeSSCoB, та кращим, аніж для базової архітектури (*DS= 0.727* до *DS= 0.718*), проте для експерименту з К-кратною перевірку результат виявився дещо гірший, ніж для базової архітектури *(DS=0.694±0.022).* Пояснення даного результату видно на Рисунок 3.10 – мережа на 4ому відрізку з тренувального набору показала значно гірші результати, ніж решта мереж (*DS=0.66*) – саме цей результат суттєво вплинув на загальний результат перехресної перевірки для модуля DeSSCo. Якщо проігнорувати мережу, натреновану на цьому відрізку, середнє значення для цієї модифікації буде на рівні *DS=0.71*, що перевершує базову архітектуру (*DS=0.701±0.003*) . Природа даного відхилення потребує подальшого дослідження. Найбільш ймовірною причиною є «перенавчання» мережі, що не дозволило вийти тренувальному процесу з локального мінімуму, а також пов’язані з цим стохастичні процеси, що супроводжують процес тренування. Цілком ймовірно, перетренування мережі або вибір дещо інших гіперпараметрів може виправити даний результат.

Вартим уваги є те, що усі ці модифікації досягли точності результатів, що порівнюється з найкращою архітектурою, що базується на коефіцієнті розширення (R=1.6, $\boldsymbol{\delta = 7}$), середнє значення якої *DS=0.715* (проти *DS=0.711* та *DS=0.713* для DeSSCoB, DeSSCo(2C) – див. Таблиця 3.2 для результатів архітектур з коефіцієнтом розширення та Таблиця 3.4 для модулів DeSSCo), при тому, що вони значно менші (51 млн. параметрів проти 32.4 млн. та 34 млн. для вищезгаданих

архітектур), проте більша мережа дає дещо стабільніші результати, якщо взяти до уваги стандартне відхилення результатів (0.0007 проти 0.003).

Дані результати не дозволяють однозначно стверджувати про перевагу однієї модифікації над іншою, однак кожна з модифікацій в тому чи іншому експерименті покращила результат базової архітектури. Якщо взяти до міркування приріст кількості параметрів та приріст результуючої метрики, то модифікацію DeSSCo можна вважати компромісним варіантом, оскільки вона збільшує розмір мережі лише на 300 тисяч параметрів (приблизно 1%), дозволивши покращити результат з *DS=0.718* до *DS=0.727* для повного набору, а також показала непогані результати для 3-ьох з 4-ьох мереж К-кратної перевірки.

4.2.2. Порівняння впливу різних варіацій способу глибинних роздільних проміжних зв'язків.

Зведена таблиця для порівняння різних запропонованих модифікацій проміжних зв'язків відображена у Таблиця 4.4. Дана таблиця відображає вплив тої чи іншої модифікації на фінальну метрику перевірки у порівнянні з базовою архітектурою U-Net. Результати зі знаком «+» відображають покращення метрики, тоді як з результати зі знаком «-» - погіршення.

Для результатів з набору даних UWGIT використовується результат нейронних мереж, натренованих на повному наборі. Для результатів набору CityScapes, використовуються результати лише модифікацій базової U-Net (тобто без модифікацій Attention-UNet). Для результатів BraTS – відображається кращий результат між архітектурами з нормалізацією, і без.

Таблиця 4.4. Порівняння результатів додавання різних покращень до базової архітектури U-Net

Архітектура	**% - відносно до результатів базової архітектури**			
	UWGIT	**Synapse**	**CityScapes**	**BraTS**
DeSSCo	+1.21%	-0.25%	-0.045%	+0.42%

DeSSCo(2C)	+0.55%	+0.22%	-2.03%	-1.56%
DeSSCoB	+1.21%	-1.78%	0.0%	-0.99%
DeSSCoBR	-1.64%	+1.61%	+0.909%	-0.33%
DeSSCoR	+0.46%	+0.16%	-0.063%	-1.85%

Було також окремо виокремлено Таблиця 4.5, де містяться порівняння додаткових варіацій запропонованого способу. У даній таблиці результати наведені для архітектур з певною модифікацією, і без неї, аби була можливість виявити закономірність впливу на результат. Також у дану таблицю занесені результати для набору даних UWGIT, що не були наведені у відповідних таблицях Розділу 3.

Таблиця 4.5. Порівняння результатів в залежності від наявності чи відсутності деякої модифікації. GN – групова нормалізація (Group Normalization), BN – нормалізація вибірки (Batch Normalization), IN – нормалізація вхідного вектору ознак (instance normalization), Residual – залишкові зв’язки

Набір даних	**Тип модифікації**	**Різновид модуля**	**Без модифікації**	**З модифікацією**
UWGIT	Residual	DeSSCo	0.727	0.7213
UWGIT	Residual	DeSSCoB	0.727	0.7009
CityScapes	Residual	DeSSCoB	0.5444	0.5486
UWGIT	BN	DeSSCo	0.7215	0.727
CityScapes	BN	DeSSCoB	0.5444	0.5471
BraTS	GN	DeSSCo (3D)	0.8055	0.8015
BraTS	IN	DeSSCo (3D)	0.8055	0.7676
BraTS	IN	DeSSCoB (3D)	0.7909	0.7943

З Таблиця 4.4 можливо зробити висновок, що для кожного з 4ьох експериментальних наборів даних, принаймні одна із запропонованих модифікацій дозволила покращити результат базової архітектури. В той же час, різноманітність

результатів не дозволяє однозначно стверджувати, що якийсь із запропонованих модулів є кращим за інші. Важливо також розуміти, що деякі з модифікацій можуть навіть відчутно погіршувати результати. Дана поведінка не є чимось критичним, оскільки багатьом архітектурам властиво показувати себе краще в одних випадках, і набагато гірше в інших (прикладом цього є нейронна мережа DeepLab, результат якої наведено в Таблиця 3.6, що показала себе значно гірше (*mIoU=0.4436*), ніж базова архітектура U-Net – *mIoU=0.5444* – за тих самих умов тренування).

Модифікації архітектури Attention-UNet. Варто окремо розглянути модифікації, що базуються на Attention-UNet. Серед цих модифікацій, найбільше виділяється результат Attention-DeSSCoB для набору даних Cityscapes, що відчутно покращив результат до *mIoU=0.5621*, що більше від базової архітектури UNet (*mIoU=0.5444*) і базової архітектури Attention-UNet (*mIoU=0.5475*). Виходячи з даних, наведених на Рисунок 3.12, дана нейронна мережа спромоглася добитися значно кращих результатів для класів, що зустрічаються нечасто в тренувальному наборі, або такими, що можуть бути переплутаними з іншими, схожими класами – наприклад, класи «wall» (стіна), «train» (потяг), «truck» (вантажівка), «traffic signal» (світлофор). Цікавим є той факт, що модифікація DeSSCo для Attention-UNet не привела до покращення результатів. Для набору даних Synapse, DeSSCo-Attention спромоглась отримати результат, дещо кращий від базової архітектури (коефіцієнт Дайса - *DS=0.7743* проти *DS=0.7654*), однак цей результат не зміг перевершити результат звичайної архітектури Attention-UNet (*DS=0.7777*), отриманий іншими дослідниками [40].

Вплив нормалізації. Що стосується впливу нормалізації (наявність або відсутність нормалізації вибірки) – то в цілому, як і очікувалося відповідно до оригінального дослідження, в якому був запропонований даний шар [10], для тренувальних процесів, де використовуються вибірки вхідних даних, це призвело до дещо кращих результатів для наборів UWGIT та Cityscapes (розміри вибірки 12 та 8 відповідно). В свою чергу, для експерименту з тривимірними об’ємами (BraTS), через обмеження по ресурсам, зображення надаються вибіркою з 1 об’єму, нормалізація вибірки (Batch Normalization) перетворюється на нормалізацію

вхідного вектору ознак (Instance Normalization). Як альтернативу, було використано також групову нормалізацію (Group Normalization). Для модуля DeSSCo, відсутність нормалізації показала кращі результати (*DS=0.8055* проти *DS=0.8015*), в той час як у нормалізація вхідного вектору ознак погіршила результати (до *DS=0.7676*). Для DeSSCoB Instance Normalization дещо покращила результат (з *DS=0.7904* до *DS=0.7943*), однак в обох випадках результати не перевершили базову архітектуру (0.8009).

Наявність залишкових (residual) зв'язків. Судячи з даних, наведених для модулів DeSSCoR, DeSSCoBR у Таблиця 3.10 та Таблиця 3.11, також важко стверджувати, що залишкові зв'язки однозначно впливають на результат в кращу або гіршу сторону. Найкраще залишкові зв'язки проявили себе на наборі Synapse, де вдалося досягнути значення коефіцієнту Дайса *DS=0.7787*, однак вони також можуть погіршити результат на 1-1.5%, як видно з результату DeSSCoBR на наборі UWGIT та DeSSCoR на наборі BraTS.

Доцільність збільшення кількості фільтрів модуля DeSSCo. Однією з запропонованих модифікацій була параметризація базового модуля DeSSCo (позначається як DeSSCo(2C)), а також модуля DeSSCoB, з кількістю фільтрів 1x1 згорткового шару $f = 2C$. Судячи з отриманих результатів, модифікація DeSSCo(2C) не дає відчутного покращення, при тому вимагає більше параметрів. Також вона значно погіршила результати в 2 з 4ьох експериментів. З огляду на це, модифікація DeSSCo(2C) не є рекомендованою, і кращою альтернативою буде використання модифікацій DeSSCoB, яка також дозволяє збільшувати кількості параметрів у модулі (в усіх дослідах використовувався $f = 2 \cdot C$), однак за рахунок «звуження» не впливає на решту шарів мережі за межами проміжних зв'язків), оскільки розмірність вхідної та вихідної карт ознак даного модуля є однаковою. Також завдяки «звуженню», модифікація DeSSCoB має можливість використовувати додатково залишкові (residual) зв'язки (звідси назва модифікації – DeSSCoBR). Ще однією перевагою є те, що DeSSCoB також інтегрується разом з такими модифікаціями, як Attention-UNet.

Результати для набору даних Synapse. Даний набір даних було обрано у контексті порівняння результатів запропонованих модифікацій нейронних мереж з широковживаними популярними архітектурами, оскільки їх результати були отримані іншими дослідниками, що спрощує порівняння. За результатами відтворення результатів, вдалося отримати результат *DS=0.7787* для нейронної мережі, що використовує модуль DeSSCoBR, що є доволі компромісним варіантом, враховуючи розміри та складність архітектури. Дана мережа конкурує з наведеними у Таблиця 3.7 архітектурами , що базуються на трансформерах (ViT, Trans-UNet, Swin-UNet), при тому, що ці архітектури використовують значно більше параметрів (50-100 млн. параметрів), та вимагають значно більше ресурсів та часу для їх тренування. Для достовірності отриманих результатів, було проведено тренування базової архітектури UNet, в результаті яких було отримані співставні результати (*DS=0.7658* проти *DS=0.7685*), що дозволяє впевнитися в оптимальності відтвореного в даному досліджені середовищі.

Тривимірні архітектури (набір даних BraTS). Незважаючи на те, що вдалося досягнути покращення результату для набору даних BraTS (0.8055 проти 0.8009), різниця між результатом є недостатньо значною, аби однозначно стверджувати ефективність модифікацій. У ході експериментів виникли певні труднощі з тим, аби виконати тренування для архітектур, що використовують модифікації DeSSCoB, оскільки пам’яті графічного процесора V100 (16 гігабайт) виявилося недостатнім, тому довелося використовувати прискорювач A100 (40 гігабайт), що є значно дорожчим, при цьому швидкість тренування суттєво не змінилася. Це призвело до надлишкового використання ресурсів, що варто враховувати у випадку використання даної мережі у інших цілях. Дане обмеження можливо обійти з використанням різноманітних додаткових технік оптимізації, однак цього не було зроблено у зв’язку з часовими рамками.

Цікавими є результати для метрики дистанції Хаусдорфа (*H*) (Таблиця 3.9). Даний коефіцієнт показує середню найбільшу відстань між контурами правильної сегментації та сегментації, отриманої нейронною мережею. Чим менший цей коефіцієнт, тим краще нейронна мережа передбачує контури. З наведених

результатів видно, що архітектура, яка використовує модуль DeSSCo, має кращий результат по коефіцієнту Дайса та краще за базову архітектуру визначає контури розширеної пухлини (*ET)* та ядра пухлини (*TC)* (H_{ET} *=7.93,* H_{TC} *=9.99* проти H_{ET} *=9.915*, H_{TC} *= 10.79*), проте схильна допускати помилки у визначенні контурів усієї пухлини (*WT*) (H_{WT} =106.16 проти H_{WT} = 20.73). З огляду на це також варто відзначити результат нейронної мережі, що використовує модуль DeSSCoB з Instance Normalization, яка хоч і показала дещо гірші результати по коефіцієнту Дайса (*DS=0.7943*), однак змогла отримати кращі результати коефіцієнта Хаусдорфа для класів ET та TC (H_{ET} *= 8.53,* H_{TC} *=8.41*), а також не набагато гірші результати для WT (H_{WT}*=24.46*). Дані результати показують певний потенціал використання запропонованих модулів з тривимірними даними, проте для повноти необхідно проводити додаткові дослідження.

4.2.3. Візуальний аналіз впливу способу глибинних роздільних проміжних зв’язків.

Поширеним способом аналізу нейронних мереж є візуальний аналіз результатів передбачень різних архітектур. Спостереження за результатами передбачень на валідаційних і тестових наборах даних дозволяє зрозуміти, як саме одна мережа робить кращі передбачення ніж інші.

Так як вихідними даними зі згорткових шарів є тривимірні карти ознак, розмірністю $W \times H \times f$ де W, H – ширина та висота вхідної карти ознак, а f – кількість фільтрів у згортковому шарі, є можливість отримати усереднене значення карти ознак для кожного пікселя (x. y)

$$\mathrm{p}^{(x,y)} = \frac{1}{F}\sum_{f=1}^{F} p_f^{(x,y)} \tag{4.1}$$

Дане представлення дозволяє отримати приблизну уяву про те, на яких ділянках нейронна мережа дає більший сигнал, а на яких, сигнал нейронів затихає.

Рисунок 4.2 показує усереднені карти ознак, що подається на вхід шару конкатенації від проміжних зв’язків, для знімку, що належить набору даних

CityScapes, для різних запропонованих модифікацій DeSSCoB, DeSSCoB-Attention, а також для базових архітектур U-Net і Attention-UNet. Дані карти ознак дозволяють зрозуміти, яким саме чином запропоновані модулі впливають на проміжний стан мережі в процесі передбачення.

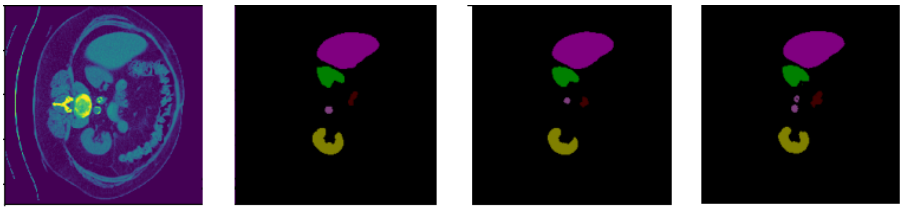

Рисунок 4.1. Порівняння сегментації на наборі даних Synapse

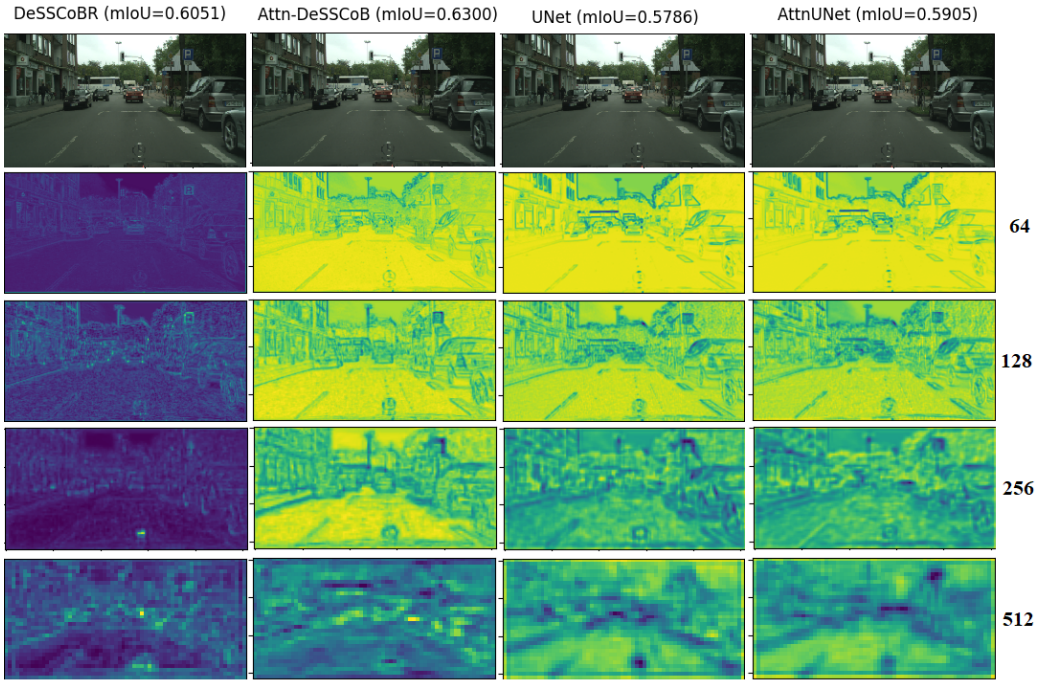

Рисунок 4.2. Візуалізація усереднених карт ознак на проміжних зв’язках у процесі передбачення на даних з набору Cityscapes.

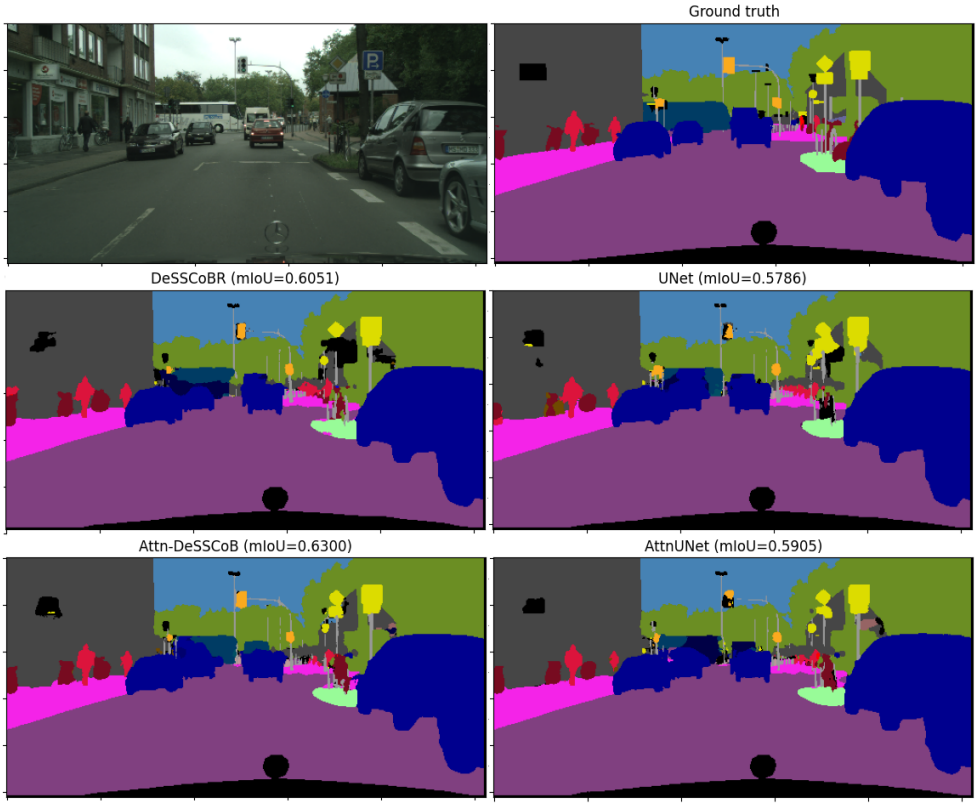

Рисунок 4.3. Результати сегментації з використанням різних архітектур (в дужках – значення метрики перевірки, mIoU)

Судячи з візуальних представлень, модулі DeSSCoB виокремлюють і підсилюють контури об’єктів, подібно до фільтрів, таких як фільтр Собеля. Така поведінка, теоретично, сприяє кращому виокремленню менших об’єктів і краще виявляти краї області сегментації. Дане припущення частково підтверджуються візуальним порівнянням результатів нейронних мереж з модулями DeSSCo і без для задачі сегментації на наборі даних Cityscapes. Наприклад на . Рисунок 4.3, архітектура DeSSCoB-Attention краще визначила такі малі об’єкти як дорожні знаки, стовпи та стовпчики. Ці результати підтверджуються і результатами на валідаційному наборі (див. Рисунок 3.11).

Також, на Рисунок 4.4 показано усереднені значення карт ознак на вході до модуля DeSSCO, та виході, де можна спостерігати ефект збільшення різкості та виокремлення контурів.

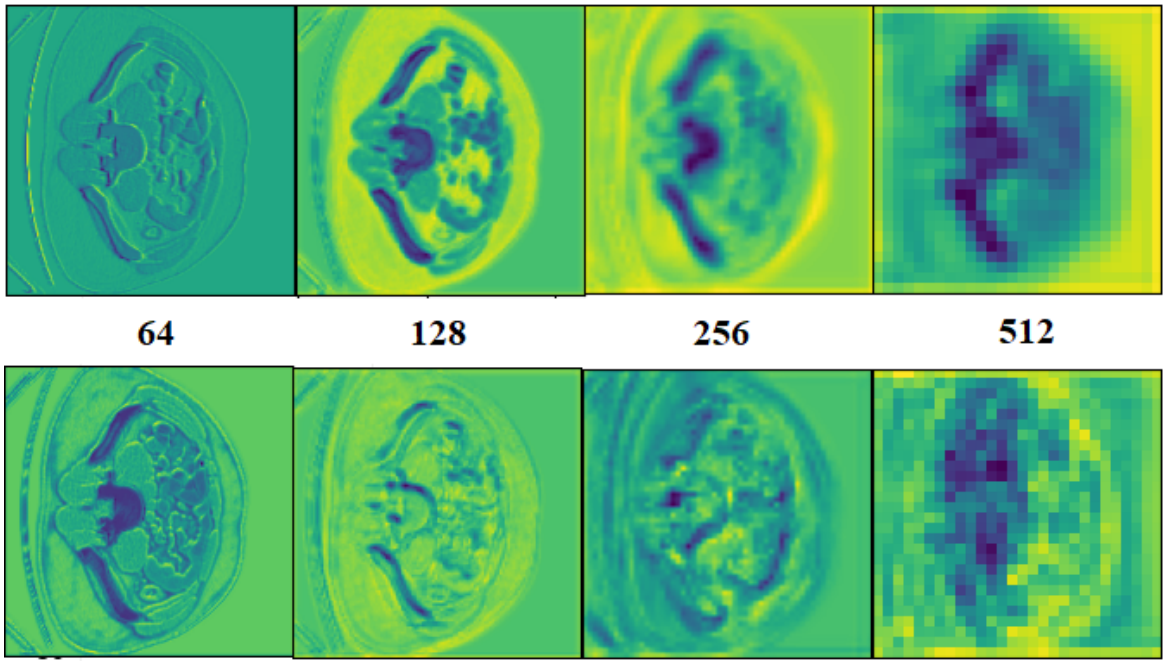

Рисунок 4.4. Усереднені карти ознак до входу у модуль DeSSCoB (зверху) та після нього (знизу). Зліва-направо – проміжні зв’язки від першого модуля дешифрувальника, до останнього (4-ого)

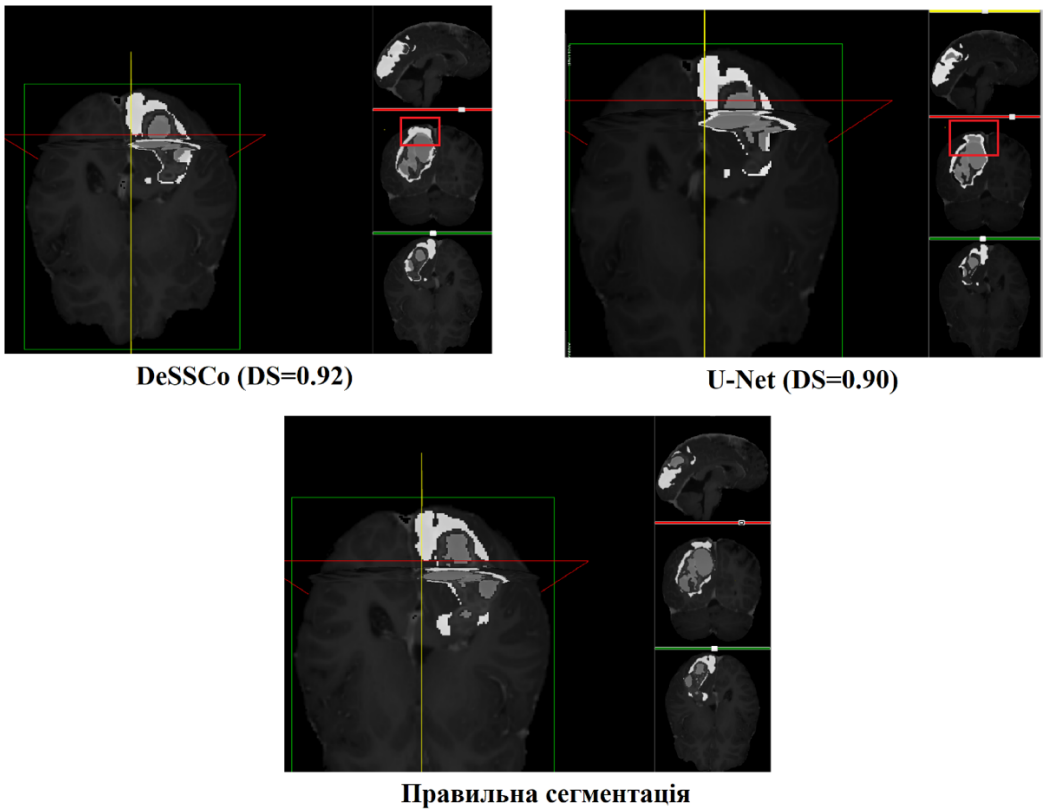

Рисунок 4.5. Порівняння результатів BraTS у трьох площинах. Червоний квадратик вказує область, де архітектура DeSSCo проявила себе краще, ніж базова архітектура.

4.2.4. Порівняння з mU-Net.

З метою порівняння ефективності використання глибинних роздільних проміжних зв'язків, було натреновано нейронну мережу, що використовує модифікацію mU-Net [55]. Саме у процесі аналізу даного варіанту архітектури U-Net виникло припущення у доцільності модифікації проміжних зв'язків.

Також спрощену версію mU-Net без операції зворотного згортування та віднімання (зображену на Рисунок 4.6.) було натреновано на наборі даних UWGIT. Ця архітектура містить 34 млн. параметрів. Мета даного експерименту – підтвердити саме якісну зміну у точності передбачень архітектури нейронної мережі за рахунок використання комбінації глибинних згорткових шарів і згорткових шарів з ядрами 1x1 у порівнянні з використанням традиційного згорткового шару 3x3 зі збереженням розмірності проміжних зв'язків.

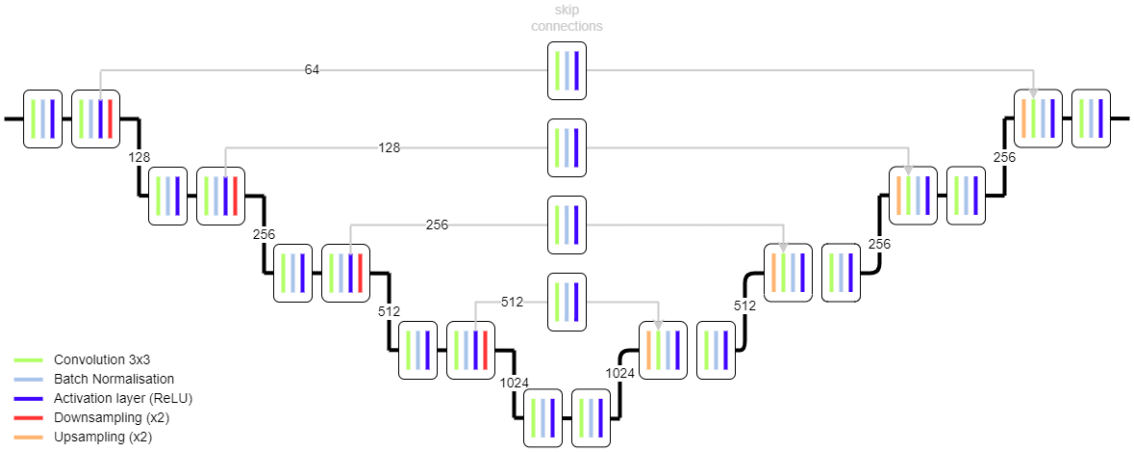

Рисунок 4.6. Спрощена архітектура mU-Net, на відміну від оригінальної mU-Net, тут відсутні операції віднімання від проміжних зв'язків та оберненого згортування

Таблиця 4.6. Порівняння результатів з mU-Net (коефіцієнт Дайса). *- базова архітектура

Архітектура	К-сть параметрів (млн.)	DS
U-Net*	31.4	0.718
DeSSCo(C)	31.7	0.726

DeSSCo(B)	32.8	0.726
mU-Net (спрощена)	34	0.722
mU-Net (повна)	37	0.717

Як видно з Таблиця 4.6, запропоновані в даній архітектурі підходи з використанням Глибинних Роздільних Проміжних Зв'язків перевершують модифікацію архітектури зі звичайним згортковим шаром з ядрами 3x3, і при цьому вимагають менше параметрів. Це свідчить, що наявність саме Глибинних Роздільних згорткових шарів спричиняє якісне покращення результатів.

4.3. Аналіз результатів гібридного способу глибинних роздільних проміжних зв'язків з варіацією коефіцієнта розширення.

Відповідно до отриманих результатів на наборі UWGIT (Таблиця 3.10), можливо спостерігати відчутне покращення точності сегментації у нейронних мережах, що використовують модуль DeSSCo(C) у порівнянні з архітектурами, у яких він не використовується. Що важливо, поставлена задача з отримання результатів, близьких до результатів базової архітектури з меншою кількістю параметрів, була досягнута, і більше того, відповідно до отриманого значення метрики перевірки (коефіцієнту Дайса *DS*), перевищена на 1.2%.

Також, цікавим спостереженням є досягнення малої мережі (R = 1.4 $\boldsymbol{\delta = 7}$) ідентичного результату з базовою архітектурою. В попередньому досліджені (Таблиця 3.2) не враховувалися результати нейронних мереж, натренованих на цілому тренувальному наборі даних, і хоча мережа R=1.4 показала при перехресній перевірці дещо гірші результати, результат тренування на повному наборі дає підстави вважати, що має місце перенавчання базової архітектури до тренувального набору, який у свою чергу не дає можливості покращити її результати. Комбінація запропонованих підходів (зменшення коефіцієнту розширення з метою поглиблення мережі, та додавання модулів DeSSCo) дозволила частково подолати дані обмеження. Використання більшої архітектури також дозволило покращити результати на 0.7% для мережі без модуля DeSSCo і

на 1.9% з даним модулем, однак з огляду на кількість параметрів це покращення дається доволі великою ціною, і, на думку автора, в не є доцільним.

Аналогічно, кращий результат було показано на наборі даних Synapse (Таблиця 3.12). Мережа *R = 1.4* $\boldsymbol{\delta = 7}$ змогла не лише досягти результату базової архітектури, а й перевершила його - на 1.3% більше від значення коефіцієнту Дайса базової мережі (*DS=0.7690* у порівнянні з *DS=0.7587*), натренованої на зображеннях розмірності 256x256, та на 0.47% більше від мережі, натренованій на 224x224 (*DS=0.7690* у порівнянні з *DS=0.7654*). Модуль DeSSCo допоміг довести даний результат до покращення коефіцієнту Дайса до 2.29% (*DS=0.7761* проти *DS=0.7587*) (до 1.38% порівняно з мережею, натренованою на даних розмірністю 224x224 - *DS=0.7761* проти *DS=0.7654*). Ще кращого результату вдалося досягнути з використанням DeSSCoB – *DS=0.7805*, що на 2.89% краще за базову мережу. Варто зазначити, що архітектура *R=1.4* $\boldsymbol{\delta = 7}$ **+** ***DeSSCoB*** також перевершила результати усіх архітектур, натренованих в ході дослідження, описаного у підпункті 3.4.3 та чиї результати наведені в Таблиця 3.7, при тому, що це найменша за кількістю параметрів архітектура з усіх наведених нейронних мереж.

Результати експерименту з набором даних Cityscapes (Таблиця 3.11) показали дещо інший результат. Нейронна мережа без модулів змогла досягнути результату базової архітектури (*mIoU=0.5441*) зі значно меншою кількістю параметрів (14 млн параметрів проти 31), однак додавання модулів DeSSCo погіршило результат. Можливою причиною цього явища також можна назвати перенавчання нейронних мереж DeSSCo, через додану комплексність (додаткові шари), і це частково підтверджувалося графіками функції втрат в процесі тренування. В цілому, в зв’язку зі складністю та специфікою задачі аналізу міського середовища (велика різноманітність об’єктів, а також відносна рідкість великої кількості категорій у вибірці даних), невеликими розмірами набору даних CityScapes, відсутністю модифікацій набору даних у порівнянні з експериментами на наборі Synapse, дозволяють припустити, що одним з факторів успішності застосування модулів DeSSCo є наявність більш різноманітного набору даних для тренування, оскільки вони мають властивість до перенавчання. Потенційним способом виправити

ситуацію, окрім збільшення варіативності навчального набору даних, може бути також додавання шарів відкидання (DropOut) до модуля DeSSCo, проте дане твердження потребує подальших експериментальних досліджень.

За підсумками проведених експериментів, додаткові модулі DeSSCo, покращили результат базової архітектури та архітектури, отриманих способом підбору коефіцієнту розширення, на двох з трьох обраних наборів даних. Не менш важливих результатів було досягнуто малими мережами R = 1.4 $\boldsymbol{\delta = 7}$ - усі вони змогли досягнути результатів, показаних базовими архітектурами, а у випадку набору Synapse – також покращити їх. Дані результати дозволяють стверджувати, що архітектура U-Net є доволі надлишковою, що часто призводить до її «перенавчання, і що її розміри можливо зменшити без погіршення метрик перевірки, а в деяких випадках можливо навіть покращити результат. Одним з додаткових векторів дослідження може бути виявлення мінімальних розмірів архітектури шляхом зменшення коефіцієнту розширення, при яких мережа усе ще буде досягати прийнятних результатів. Завдяки додаванню додаткових, «легковагових» модулів DeSSCo, відкривається можливість дещо компенсувати втрати точності нейронної мережі від зменшення її розмірів, що підкреслює корисність та практичність комбінації обох способів побудови мереж U-Net в рамках запропонованого методу.

4.4. Вимірювання швидкодії та використання пам’яті для різних архітектур нейронних мереж.

Ще одним із важливих факторів, що часто визначає вибір архітектури нейронної мережі, є такий фактор, як використання ресурсів, таких як використання пам’яті та обчислювальних ресурсів CPU. Не зважаючи на те, що для тренування нейронних мереж використовуються графічні процесори GPU, багато кінцевих приладів, які будуть запускати нейронні мережі, мають доволі обмежений обчислювальний ресурс – наприклад, мобільний телефон, чи контролери, що встановлюються на автономні прилади, такі як дрони, не мають окремого інтегрованого графічного процесора, тому розробники часто обмежені ресурсом

CPU, або спеціальних TPU приладів. Ще одним фактором є обмеження енергетичного ресурсу, оскільки графічні процесори часто мають велику потужність та витрати енергії, що впливає на їх використання у автономних пристроях, котрі живляться від батареї.

В екосистему TensorFlow входить технологія TensorFlow Lite [94], що дозволяє для скомпілювати нейронну мережу у форматі SavedModel спеціально під задану архітектуру виконання нейронних мереж на edge-пристроях [95]. Використання TensorFlow Lite для проведення вимірювання швидкодії та використання пам'яті нейронною мережею є доцільним, оскільки саме edge пристрої часто є кінцевими платформами застосування нейронних мереж у ході їх експлуатації. Також, мережі у форматі TensorFlow Lite будуть оптимізовані під платформу, що дає змогу відкинути всю надлишковість звичайного середовища розробки.

У набір інструментарію TensorFlow Lite також входять утиліти для проведення вимірювання швидкодії та використання пам'яті [96]. Даний інструмент дозволяє виміряти наступні метрики:

- Час ініціалізації ($\mathbf{T_{init}}$);
- Час передбачення (отримання результатів сегментації із зображення) на етапі розігріву ($\mathbf{T_w}$);
- Час передбачення у стабільному режимі ($\mathbf{T_{avg}}$);
- Використання пам'яті під час ініціалізації ($\mathbf{M_{init}}$);
- Загальне використання пам'яті ($\mathbf{M_{total}}$);

Програмний код, який проводить вимірювання швидкодії та використання пам'яті нейронної мережі, наведено нижче:

```
def benchmark_model(model_params):
    model = get_unet_model(**model_params)
    rand_name = random_str(5)

    saved_model_dir = os.path.join('content', rand_name)
    os.makedirs(saved_model_dir)
    model.save(saved_model_dir)
    model.summary()
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
    converter.target_spec.supported_ops = [
```

```
        tf.lite.OpsSet.TFLITE_BUILTINS,
        tf.lite.OpsSet.SELECT_TF_OPS
    ]
    tflite_model = converter.convert()
    model_file = f'model_{rand_name}.tflite'
    # Save the model.
    with open(model_file, 'wb') as f:
        f.write(tflite_model)
    logger.info("Running benchmark for parameters %s", model_params)
    try:
        res = subprocess.check_output(f'./linux_x86-
64_benchmark_model_plus_flex --graph={model_file} '
                                      '--input_layer=input --
input_layer_shape=1,224,224,1',
                                      shell=True)
    except Exception as e:
        logger.error('Exception %s', e)
        res = e.output
    logger.info('%s', res.decode('ascii'))
```

Вимірювання було проведено у середовищі Google Colab. В рамках вимірювання, у задану архітектури, скомпільовану в форматі .tflite на вхід подається 50 генерованих випадковим чином тензорів розмірами 1 x 224 x 224 x 1 (відбувається симуляція вибірки з 1 чорно-білого зображення розміром 224 x 224) для тестування модулів DeSSCo, і 1 x 256 x 256 x 1 для тестів архітектур модифікованих способом підбору коефіцієнту розширення. Для проведення вимірювань швидкодії використовувався лише ресурс CPU.

4.4.1. Вимірювання швидкодії та використання пам’яті способу підбору коефіцієнту розширення.

Результати вимірювань наведено у Таблиця 4.7. Дана таблиця розбита по групам з однаковою глибиною мережі $\boldsymbol{\delta}$, значення коефіцієнту розширення підібрані таким чином, щоб у кожній групі були нейронні мережі з однаковою кількістю параметрів, аналогічно до експериментів, описаних в Розділі 3. Також наведені результати базової архітектури, для проведення порівняльного аналізу. Виявилося, що підхід збільшення глибини мережі за рахунок зменшення кількості фільтрів згорткових шарів, окрім того, що дозволяє покращити точність

сегментації, також дозволяє і пришвидшити процес передбачення та більш ефективно використовувати пам’ять. Так, відповідно до значення середнього часу передбачення $\mathbf{T_{avg}}$, архітектури, що мають 31 млн параметрів (аналогічно до базової архітектури) роблять передбачення на 24.3% швидше у випадку архітектури $R = 1.7, \boldsymbol{\delta = 6}$, і на 47% швидше у випадку архітектури $R = 1.52, \boldsymbol{\delta = 7}$. Що стосується використання пам’яті, то для мережі $R = 1.7, \boldsymbol{\delta = 6}$ використовується на 4.7% менше пам’яті, тоді як для $R = 1.52, \boldsymbol{\delta = 7}$ цей показник краще на 5.6%. При цьому, мережа $R = 1.52, \boldsymbol{\delta = 7}$ (згідно Таблиця 3.2) також показала статистично незначне покращення результату базової мережі для коефіцієнту Дайса на К-кратній перевірки. Усі ці фактори вказують на більшу доцільність використання даного варіанту архітектури проти базової U-Net.

Таблиця 4.7. Результат вимірювань швидкодії та використання пам’яті для архітектур з різною глибиною $\boldsymbol{\delta}$ та коефіцієнтом розширення R. Для показників часу, (Т), результат наведено в мілісекундах. Для показників пам’яті (М) – у мегабайтах.

R	**К-сть параметрів (млн)**	$\mathbf{T_{init}}$	$\mathbf{T_w}$	$\mathbf{T_{avg}}$	$\mathbf{M_{init}}$	$\mathbf{M_{total}}$
$\boldsymbol{\delta = 5}$ (Базова)						
2	31.4	15.507	1026.02	1077.79	10.535	357.895
$\boldsymbol{\delta = 6}$						
1.52	13.41	20.845	684.346	715.866	10.3672	269.277
1.6	20	22.791	718.210	745.328	10.30	277.652
1.7	32.42	44.770	1026.86	866.504	10.523	341.777
1.75	41.121	22.185	833.843	857.591	10.863	378.883
1.8	51.792	21.287	906.742	942.583	10.726	428.105
$\boldsymbol{\delta = 7}$						
1.4	14.46	22.149	627.126	653.364	9.238	268.465

1.5	27.59	22.282	659.904	685.046	10.98	315.57
1.52	31.24	21.543	761.383	729.182	11.078	338.461
1.56	40.161	21.647	795.888	754.566	11.14	342.473
1.6	51.447	21.114	770.78	792.516	10.886	395.496

Також варто зазначити, що навіть мережі з більшою кількістю параметрів, мають кращі показники швидкодії, ніж базова архітектура – на 13% для архітектури з глибиною 6, та 27% для архітектури з глибиною 7. При цьому, використання пам'яті перевищує базову архітектуру на 10%, при тому, що кількість параметрів більша на 64%. Більш наглядно співвідношення між нейронними мережами з глибиною 6, 7, і базовою архітектурою зображено на Рисунок 4.8. Це сприяє використанню більшої мережі з метою отримання кращих результатів.

Поясненням зменшення тривалості передбачення може бути загальне зменшення кількості операцій над числами з плаваючою комою (Floating point operations, FLOP) за рахунок зменшення глибини карт ознак, що отримуються на виході з суміжних послідовних згорткових шарів. Оскільки більшість шарів представлених мереж є згортковими, для того, аби підрахувати кількість операцій, достатньо просумувати кількість операцій для усіх згорткових шарів. Кількість операцій для згорткового шару з кількістю фільтрів f, розмірністю $k_1 \times k_2$ та розмірами вхідної карти ознак H×W×C рахується приблизно як $\approx 2 \cdot k_1 \cdot k_2 \cdot f \cdot H \cdot W \cdot C$. Оскільки для мереж U-Net з коефіцієнтом розширення $f_i = R \cdot f_{i-1} = R^i \cdot C_0$, а розміри вхідної карти ознак $H_i \cdot W_i \cdot C_i = \frac{1}{2} H_{i-1} \cdot \frac{1}{2} W_{i-1} \cdot f_{i-1} = 2^{-2i} \cdot R \cdot W_0 \cdot H_0 \cdot C_0$, кількість операцій в залежності від глибини нейронної мережі та коефіцієнта розширення буде пропорційною наступному виразу:

$$FLOP \sim \left[(R+1) \sum_{i=1}^{\delta} 2^{-2i} R^{2i} \right] \tag{4.2}$$

З наведеної вище формули можна спостерігати, що у разі R=2, він спрощується до формули $FLOP \sim 3 \cdot \delta$, а також те, що на кожному з модулів базової архітектури, кількість операцій, що виконуються, приблизно однакова. У той же час, якщо R <

2, то зі збільшенням порядку модулів кількість операцій буде зменшуватися. Рисунок 4.7 візуалізує дане твердження.

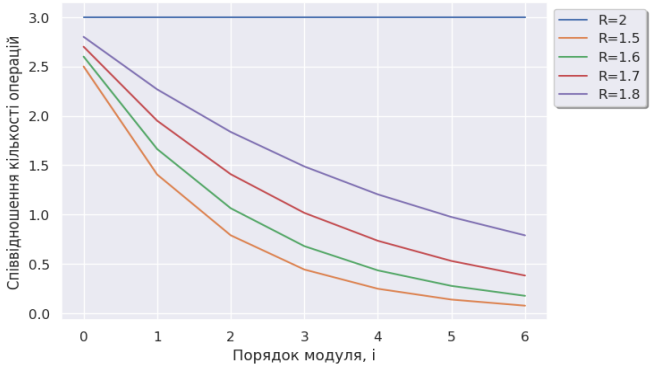

Рисунок 4.7. Співвідношення кількості операцій у модулі *i* в залежності від коефіцієнта розширення R. Значення вирахувані з формули 4.2.

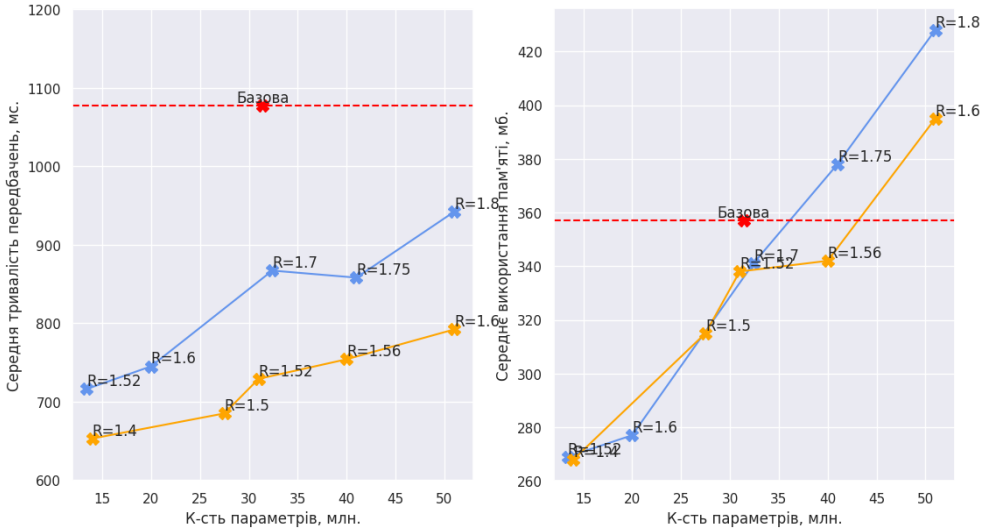

Рисунок 4.8. Середні значення по часу передбачення (зліва) та використанню пам’яті (справа) в залежності від кількості параметрів нейронної мережі. Синій колір $\boldsymbol{\delta = 6,}$ жовтий - $\boldsymbol{\delta = 7,}$ червоний – показники базової архітектури.

4.4.2. Вимірювання швидкодії та використання пам’яті для способу глибинних роздільних проміжних зв’язків.

Результати вимірювань наведені у Таблиця 4.8. Згідно таблиці, можливо зробити висновок, що складність додаткового модуля збільшує тривалість процесу передбачення – наприклад, передбачення з DeSSCo (відповідно до метрики середнього часу передбачення T_{avg}) займає на 37.2% більше часу, ніж базова

архітектура, DeSSCoB – на 44%, DeSSCoBR (DeSSCoB з залишковими зв'язків) – 48.4%, DeSSCo(2C) – на 61.9%. Аналогічний ефект спостерігається для Attention-UNet – 32.7% та 35.6% для DeSSCo та DeSSCoB.

Таблиця 4.8. Результати проведеного вимірювання швидкодії та використання пам'яті нейронної мережі в залежності від модифікації DeSSCo. Для показників часу, (T), результат наведено в мілісекундах. Для показників пам'яті (M) – у мегабайтах.

Архітектура	**К-сть параметрів (млн)**	**T_{init}**	**T_w**	**T_{avg}**	**M_{init}**	**M_{total}**
U-Net	31.4	15.507	569.499	541.435	10.41	314.141
Attn-UNet	33.14	18.510	832.242	685.518	12.1875	626.238
Attn-UNet + DeSSCoB	34.56	17.254	1158.151	929.058	13.125	642.691
Attn-UNet + DeSSCo	33.51	16.876	1022.151	909.060	12.5234	643.324
DeSSCoB	32.82	16.599	803.552	779.131	11.207	362.828
DeSSCoBR	32.82	12.545	792.077	803.148	11.0234	345.844
DeSSCo	31.76	13.99	781.618	742.827	10.6914	340.25
DeSSCo(2C)	35.25	13.839	924.163	876.489	10.5156	393.934

Що стосується використання пам'яті, то загальний об'єм пам'яті (M_{total}), необхідний для роботи нейронної мережі, також збільшується в залежності від її комплексності. Для DeSSCo – на 8.2% у порівнянні з базовою архітектурою; DeSSCoB – на 15.3%; DeSSCoBR – на 9.9%, DeSSCo(2C) - на 25.2%; для Attention-UNet, DeSSCo та DeSSCoB показали збільшення приблизно на однакову величину у 2.5%.

Варто зазначити, що за рахунок глибинного згорткового шару (Depthwise Convolution), швидкодія нейронної мережі зменшується доволі відчутно, особливо якщо врахувати незначну кількість додаткових параметрів, що додаються разом з даним шаром, у порівнянні зі звичайними згортковими шарами. Наприклад, для Attention-UNet, що складається виключно зі звичайних згорткових шарів, приріст в тривалості передбачень оцінюється у 0.08 мікросекунд на 1000 додаткових параметрів у порівнянні з класичною U-Net, тоді як для DeSSCo, даний приріст становить 1.79 мікросекунд на 1000 додаткових параметрів. Така різниця пояснюється відносно малим співвідношенням арифметичних операцій до операцій з пам’ятю [97], котрі є повільнішими, оскільки залежать від апаратної архітектури та реалізації. Разом з тим, глибинні згорткові шари дають менший приріст у використанні пам’яті, що зумовило його використання у архітектурі MobileNet, і робить дану модифікацію особливо актуальною для використання в edge пристроях, що характеризуються перш за все обмеженнями по ресурсам.

Збільшення часу передбачення варто враховувати при виборі модифікації, оскільки це доволі суттєво впливає на час тренування нейронної мережі та швидкість прийняття рішення у режимі передбачення, що варто враховувати у таких системах, які працюють в режимі реального часу, такі як автомобілі чи автоматизовані безпілотні системи.

4.4.3. Вимірювання швидкодії гібридного способу глибинних роздільних проміжних зв’язків та підбору коефіцієнта розширення.

Корисними також будуть вимірювання швидкодії для нейронних мереж, що комбінують у собі коефіцієнт розширення та модулі DeSSCo. У попередніх підпунктах комбінація даних модифікацій дозволила покращити результати у порівнянні з базовою архітектурою U-Net. Згідно Таблиця 4.7, використання способу підбору коефіцієнту розширення дозволяє суттєво пришвидшити передбачення нейронної мережі, тоді як глибинні роздільні проміжні зв’язки DeSSCo (згідно Таблиця 4.8) дещо збільшують цей час. Оскільки завдяки

додатковим модулям можливо досягнути кращих результатів, також важливо, аби отримана архітектура працювала швидше, ніж базова.

Результати вимірювань наведені у Таблиця 4.9. Експерименти проводилися для вхідних даних розмірів 1x256x256x1. Варто зазначити, що результати базової архітектури та мереж з коефіцієнтом розширення дещо розбігаються з Таблиця 4.7 – цю розбіжність можливо пояснити мінливістю середовища Google Colab, що виділяє ресурси для користувача відповідно до їх наявності, однак відносне співвідношення швидкодії між цими архітектурами зберігається.

Таблиця 4.9. Результати вимірювання швидкодії та використання пам’яті для гібридного способу. Значення часу подані у мілісекундах, значення пам’яті – у мегабайтах.

Архітектура	**К-сть параметрів (млн)**	$\mathbf{T_{init}}$	$\mathbf{T_w}$	$\mathbf{T_{avg}}$	$\mathbf{M_{init}}$	$\mathbf{M_{total}}$
U-Net	31.4	40.775	642.479	488.765	7.73	372.727
DeSSCo	31.76	39.247	724.216	681.622	7.48	389.5
R=1.6, $\delta = 7$	51.44	22.149	329.164	390.821	8.14	397.465
R=1.6, $\delta = 7$ DeSSCo	52.215	40.858	486.385	548.641	8.78	432.211
R=1.6, $\delta = 7$ DeSSCoB	54.43	40.314	596.792	581.861	9.02	505.133
R=1.4, $\delta = 7$	14.46	26.021	426.549	313.562	9.32	274.707
R=1.4, $\delta = 7$ DeSSCo	14.717	28.781	530.013	509.768	8.746	324.43
R=1.4, $\delta = 7$ DeSSCoB	15.43	36.1	501.348	495.716	9.05	335.484

З наведеної таблиці можна спостерігати, що архітектури, отримані гібридним способом модифікації, все таки працюють повільніше, аніж базова архітектура –

так, відповідно до метрики T_{avg}, R=1.6, $\delta = 7$ + DeSSCo працює на 12.3% довше; мережа R=1.4, $\delta = 7$ + DeSSCo – на 4.3%, проте дані нейронні мережі працюють значно швидше, ніж DeSSCo-UNet (на 24.3% та 33.8% відповідно).

Що стосується пам'яті (метрика M_{total}), то модифікована менша мережа R=1.4, $\delta = 7$ + DeSSCo використовує на 13% менше пам'яті, ніж базова архітектура (та на 18.2% більше, ніж мережа, що використовує лише коефіцієнт розширення - R=1.4, $\delta = 7$). Ці показники, разом з відносно незначним збільшення часу передбачення, вказують на корисність використання даної архітектури, особливо в тих випадках, коли вона також дозволяє покращити результати.

Висновки до розділу 4

В даному розділі було проаналізовано результати, отримані у ході експериментальної частини роботи. Основне обговорення стосується нейронних мереж, отриманих з використанням способу підбору коефіцієнта розширення та модуля глибинних роздільних проміжних зв'язків (Depthwise Separable Convolutions) архітектури U-Net. Також в даному розділі були висвітлені додаткові дослідження та експерименти (такі як використання коефіцієнту розширення на тривимірних даних, порівняння запропонованих архітектур з mU-Net).

В результаті експериментів, вдалося отримати архітектури, що змогли покращити метрики перевірки на усіх розглянутих наборах даних: модифікація Attention-DeSSCoB покращила результат на 3.2% для набору даних CityScapes, DeSSCoBR – на 1.61% для набору Synapse, DeSSCo – на 1.21% для набору UWGIT та 0.42% для набору BraTS. Варто зазначити, що використання одної з запропонованих модифікацій в рамках способу глибинних роздільних проміжних зв'язків (DeSSCo, DeSSCoB, DeSSCoBR, DeSSCoR) не завжди гарантує відчутне покращення, тобто ефективність того чи іншого модуля для поставленої задачі доведеться визначати емпіричним способом.

В рамках аналізу запропонованого методу, було проведено візуальний аналіз результатів передбачень запропонованих способів модифікації нейронної мережі

U-Net. Було встановлено, що модифікації DeSSCo сприяють кращому виділенню контурів об’єктів, що потенційно приводить до кращих результатів передбачень.

Серед іншого, було також проведено аналіз метрик швидкодії та використання пам’яті запропонованих модифікацій в рамках способу глибинних роздільних проміжних зв’язків, що дає краще уявлення про вплив даних модулів в залежності від кількості додаткових параметрів. Незважаючи на покращення точності передбачення, варто враховувати, що швидкодія нейронної мережі також зменшується. Аналогічні вимірювання було проведено на нейронних мереж, отриманих способом підбору коефіцієнту розширення. Було виявлено, що окрім покращення точності передбачення, також реально добитися пришвидшення роботи архітектури у режимі передбачення при більшій кількості параметрів, за рахунок зменшення кількості обчислювань для глибших карт ознак – так, архітектура $\boldsymbol{R=1.4,\delta = 7}$ працює у 1.64 рази швидше, ніж базова архітектура.

Перспективним розвитком запропонованих у даній роботі підходів буде подальше дослідження комбінації архітектур нейронних мереж, побудованих з використанням коефіцієнту розширення, та модулів DeSSCo. У проведених експериментах було підтверджено можливість досягнення результатів, що є близькими або ідентичними до результатів базової архітектури, з використанням значно менших нейронних мереж, які окрім цього також є більш ефективними з точки зору часу виконання та використання ресурсів, а модулі DeSSCo мають потенціал компенсувати втрати результуючих метрик від зменшення розмірів нейронних мереж, чи навіть покращити результат (наприклад, на 2.29% для набору Synapse, при розмірі архітектури у 14.7 млн параметрів, проти базових 31 млн параметрів).

ВИСНОВКИ

В дисертаційній роботі було проведено ряд експериментальних досліджень та запропоновано новий метод вирішення задач із семантичної сегментації зображень. Для досягнення даної мети, було виконано ряд наступних кроків:

- Наведено огляд контексту та наявних проблем у вирішені задачі сегментації зображень, сформульовано мету дослідження і основних способів її досягнення на основі методу модифікації існуючих архітектур глибоких нейронних мереж типу “шифрувальник-дешифрувальник” на прикладі нейронної мережі U-Net;
- Проведено ознайомлення та аналіз основних архітектур нейронних мереж, присвяченим задачам аналізу зображень з метою класифікації та семантичної сегментації, визначено їх принципові відмінності та нововведення. Описано основні принципи та компоненти, з яких побудовані згорткові нейронні мережі. Наведено приклади використання нейронних мереж для виконання різних задач в практичній площині, таких як аналіз медичних зображень та автономне керування авто. Особливу увагу було приділено такому фактору, як розміри нейронної мережі в залежності від того, які шари використовуються, виходячи з чого було запропоновано ряд кількісних та якісних модифікацій.
- Описано методологію проведення експериментальних досліджень, що включає в себе вибір тренувальних наборів даних, апаратно-програмних засобів для тренування нейронних мереж, а також архітектуру програмного забезпечення для проведення експериментів та візуалізації результатів з використанням бібліотеки TensorFlow. Запропоноване програмне рішення є таким, що дозволяє з відносною легкістю відтворити отримані результати, а також адаптувати для проведення додаткових експериментів з використанням інших архітектур мереж та наборів даних.
- Запропоновано новий спосіб підбору коефіцієнта розширення нейронної мережі для модифікації сімейства архітектур типу «шифрувальник-дешифрувальних» U-Net, що додає додаткової гнучкості до базової архітектури,

надаючи можливість регулювати збільшення розмірів мережі за рахунок зменшення глибини згорткових шарів. Це дозволило отримати нейронні мережі, що маючи малі розміри, спроможні досягнути близької до базової архітектури точності сегментації, а в деяких випадках – покращити її.

- Запропоновані новий спосіб модифікації нейронної мережі архітектури U-Net з використанням глибинних роздільних згорток, - спосіб глибинних роздільних проміжних зв'язків (англ. *Depthwise-Separable Skip Connections (DeSSCo, DSSC)*), а також його варіанти - глибинні роздільні проміжні зв'язки із Звуженням (англ. *Depthwise-Separable Skip Connection with Bottleneck (DeSSCoB, DSSCB)*), та залишкові глибинні роздільні проміжні звязки. Це дозволило збільшити точність сегментації у порівнянні з базовою архітектурою, при незначному збільшені кількості параметрів. Було запропоновано також модифікацію архітектури Attention-UNet, та гібридний спосіб з використанням способу підбору коефіцієнту розширення. Завдяки проведеним експериментам на різних наборах даних, вдалося показати життєздатність даних модифікацій для різноманітних задач. Так, вдалося покращити результати базової U-Net архітектури для набору CityScapes на 3.2% за допомогою використання модифікації Attn-DeSSCoB-UNet, що також покращило результат Attention-UNet на 2.6%; модифікація DeSSCo покращила результат на наборі UWGIT на 1.2%; DeSSCoBR для набору Synapse покращила результат базової архітектури на 1.6%;
- В рамках експериментальної частини дослідження було проведено велику кількість експериментів з використанням щонайменше 4-ьох загальнодоступних наборів даних, таких як UWGIT, Synapse, BraTS, Cityscapes, що представляють два різних та актуальних домени знань – аналіз медичних зображень, та аналіз міського середовища, а також дві різні розмірності даних – двовимірні зображення, та тривимірні об'єми. В багатьох експериментах використовувався різноманітний набір інструментів з модифікації тренувальних наборів даних.
- Окрім експериментального підтвердження доцільності модифікацій на основі способу глибинних роздільних згорткових шарів, та способу підбору

коефіцієнту розширення, було також проведено аналіз впливу даних модифікацій на час виконання передбачення, а також використання ресурсів пам’яті, що є актуальними метриками у випадку використання даних нейронних мереж у контексті edge пристроїв. Було встановлено, що мережі, отримані з використанням способу підбору коефіцієнта розширення, можуть не лише покращити точність сегментації зображень, а й значно пришвидшити процес передбачення, що актуально у контексті таких задач, як відстеження (трекінг) об’єктів.

- Наостанок, було проаналізовано ряд недоліків запропонованих способів: глибинні роздільні проміжні зв’язки збільшують час передбачення, а також можуть призводити до перенавчання нейронної мережі; спосіб підбору коефіцієнту розширення має обмежене використання в контексті тривимірної сегментації через ресурсні обмеження.

Покращення метрик у контексті аналізу міського середовища та медичних зображень навіть на долю відсотка є критично важливим для підвищення інформаційної обізнаності та безпеки дорожнього руху, чи діагностики хвороб. Саме тому отримані результати мають практичне значення, а різноманіття контексту проведених експериментів також підтверджує універсальність запропонованого методу.

ЛІТЕРАТУРА

[1] G. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. IEEE. 11. 33 - 35. 10.1109/N-SSC.2006.4785860.," *Solid-State Circuits Newsletter,* vol. 38, no. 8, pp. 33-35, 1965.

[2] N. J. Nilsson, . "Learning machines.", 1965.

[3] R. Duda and H. P., Pattern Recognition and Scene Analysis, Wiley Interscience, 1973.

[4] C. M. Bishop and N. M. Nasrabadi, Pattern recognition and machine learning, New York: springer, 2006.

[5] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[6] F. Rosenblatt, "The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain," *Psychological Review. ,* p. 386–408, 1958.

[7] C. Cortes and V. Vapnik, " Support-vector networks," *Mach Learn 20,* p. 273–297, 1995.

[8] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, " "Gradient-based learning applied to document recognition"," *Proceedings of the IEEE,* vol. 86, no. 11, p. 2278–2324, 1998.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, *"Mobilenets: Efficient convolutional neural networks for mobile vision applications.",* arXiv preprint arXiv:1704.04861, 2017.

[10] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Proceedings of the 32nd International Conference on Machine Learning,* pp. 448-456, 2015.

[11] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics,* vol. 20, no. 3, pp. 121-136, 1975.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research, JMLR. org, vol. 15, n. 1,* pp. 1929--1958, 2014.

[13] M. D. Zeiler, D. Krishnan, G. W. Taylor and R. Fergus, "Deconvolutional networks,," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, doi: 10.1109/CVPR.2010.5539957.*, San Francisco, CA, IEE, 2010, pp. 2528-2535.

[14] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty and P. Hammarlund, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 451--460.

[15] A. Krizhevsky, I. Sutskever and G. E. Hinton., ""Imagenet classification with deep convolutional neural networks."," in *Advances in neural information processing systems*, 2012.

[16] R. Girshick, J. Donahue, T. Darrell and J. & Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).*, 2014.

[17] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) pp. 1440-1448*, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2014.

[19] K. Simonyan and A. & Zisserman, *Very deep convolutional networks for large-scale image recognition.,* arXiv preprint arXiv:1409.1556., 2014.

[20] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult.," *IEEE transactions on neural networks 5.2,* pp. 157-166, 1994.

[21] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition.," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[22] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.

[23] O. Ronneberger, P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International conference on medical image computing and computer-assisted intervention*, 2015.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[25] L. M. Tan and V. Quoc, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946. [Accessed 30 9 2023].

[26] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.," *IEEE transactions on pattern analysis and machine intelligence,,* vol. 40, no. 4, pp. 834-848, 2016.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner and M. Dehghani, *An image is worth 16x16 words: Transformers for image recognition at scale.,* arXiv preprint arXiv:2010.11929, 2020.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, ""Attention is all you need."," *Advances in neural information processing systems ,* no. 30, 2017.

[29] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox and O. Ronneberger, "3D U-Net: learning dense volumetric segmentation from sparse annotation," pp. 424--432, 2016.

[30] F. Milletari, N. Navab and S.-A. Ahmadi, *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation,* arXiv:2105.05537, 2016.

[31] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano and G. Irving, "Fine-Tuning Language Models from Human Preferences," 2019. [Online]. Available: https://arxiv.org/abs/1909.08593.

[32] J. Deng, W. Dong, R. Socher, L. -J. Li, K. Li and L. Fei-Fei, ""ImageNet: A large-scale hierarchical image database,," in *2009 IEEE Conference on Computer Vision and Pattern Recognition,*, Miami, FL, USA, 2009.

[33] P. J. Werbos, The roots of backpropagation: from ordered derivatives to neural networks and political forecasting., John Wiley & Sons, 1994.

[34] K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, "Masked Autoencoders Are Scalable Vision Learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 16000-16009. 2022.*, 2022.

[35] L. S and G. CL., "Overfitting and neural networks: conjugate gradient and backpropagation," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 2000.

[36] "Google Scholar Data for "U-net: Convolutional networks for biomedical image segmentation"," 1 10 2023. [Online]. Available: https://scholar.google.com/citations?view_op=view_citation&hl=en&user=

7jrO1NwAAAAJ&citation_for_view=7jrO1NwAAAAJ:K3LRdlH-MEoC. [Accessed 1 10 2023].

[37] O. Oktay, J. Schlemper, L. L. M. Folgoc, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Hammerla, B. Kainz, B. Glocker and D. Rueckert, *Attention U-Net: Learning Where to Look for the Pancreas.,* arXiv:1804.03999, 2018.

[38] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh and J. Liang, *UNet++: A Nested U-Net Architecture for Medical Image Segmentation,* https://doi.org/10.48550/arXiv.1807.10165, 2018.

[39] Z. Z, L. Q and W. Y. I, "Road extraction by deep residual u-net.," *IEEE Geoscience and Remote Sensing Letters. 15(5):,* pp. 749-53., 8 Mar 2017.

[40] J. Chen, Y. Lu, Q. Yu, X. Luo, E. Adeli, Y. Wang, L. Lu, A. L. Yuille and Y. Zhou, *TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation,* https://arxiv.org/pdf/2102.04306.pdf, 2021.

[41] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian and M. Wang, *Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation,* https://doi.org/10.48550/arXiv.2105.05537, 2021.

[42] N. Beheshti and L. Johnsson, "Squeeze U-Net: A Memory and Energy Efficient Image Segmentation Network," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop*, Seattle, 2020.

[43] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,* arXiv:1602.07360, 2016.

[44] J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, *Squeeze-and-Excitation Networks,* arXiv:1709.01507, 2017.

[45] "Multi-Atlas Labeling Beyond the Cranial Vault - Workshop and Challenge," Synapse Storage, 2015. [Online]. Available:

https://www.synapse.org/#!Synapse:syn3193805/wiki/217789. [Accessed 13 10 2023].

[46] HuBMAP Consortium, "The human body at cellular resolution: the NIH Human Biomolecular Atlas Program".

[47] A. Howard, A. Sohier, C. Lindskog, E. Lundberg, K. Borner, L. Godwin, Shriya, S. Dane, T. Le and Y. Jain, "HuBMAP + HPA - Hacking the Human Body. Kaggle.," Kaggle, 2022. [Online]. Available: https://kaggle.com/competitions/hubmap-organ-segmentation. [Accessed 25 10 2023].

[48] R. Mehta, A. U. Filos, C. Baid and e. al, "QU-BraTS: MICCAI BraTS 2020 challenge on quantifying uncertainty in brain tumor segmentation-analysis of ranking scores and benchmarking results," *The journal of machine learning for biomedical imaging,* 2022.

[49] J. Yang, R. Shi and D. Wei, "MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification," *Scientific Data,* vol. 10, no. 1, p. 41, 2023.

[50] N. Altini, G. D. Cascarano, A. Brunetti, F. Marino, M. T. Rocchetti, S. Matino, U. Venere, M. Rossini, F. Pesce, L. Gesualdo and others, "Semantic segmentation framework for glomeruli detection and classification in kidney histological sections," *Electronics,* vol. 9, no. 3, p. 503, 2020.

[51] R. Statkevych, S. Stirenko and Y. Gordienko, "Human kidney tissue image segmentation by u-net models,," in *IEEE EUROCON 2021-19th international conference on smart technologies*, Lviv, 2021.

[52] R. Statkevych, Y. Gordienko and S. Stirenko, "Improving u-net kidney glomerulus segmentation with fine-tuning, dataset randomization and augmentations," in *International Conference on Computer Science, Engineering and Education Applications*, Kyiv, 2022.

[53] D. Govind, B. Ginley, B. Lutnick, J. E. Tomaszewski and P. Sarder, "Glomerular detection and segmentation from multimodal microscopy

images using a Butterworth band-pass filter," in *Medical Imaging 2018: Digital Pathology*, vol. 10581, International Society for Optics and Photonics, 2018, p. 1058114.

[54] R. Statkevych, Y. Gordienko and S. Stirenko, "Expansion Rate Parametrization and K-Fold Based Inference with U-Net Neural Networks for Multiclass Medical Image Segmentation," in *International Conference on Artificial Intelligence and Soft Computing*, Zakopane, 2023.

[55] H. Seo, C. Huang, M. Bassenne, R. Xiao and L. Xing, "Modified U-Net (mU-Net) with incorporation of object-dependent high level features for improved liver and liver-tumor segmentation in CT images," *IEEE transactions on medical imaging,* vol. 39, pp. 1316--1325, 2019.

[56] Y. Gordienko, P. Gang, J. Hui, W. Zeng, Y. Kochura, O. Alienin, O. Rokovyi and S. Stirenko, "Deep learning with lung segmentation and bone shadow exclusion techniques for chest X-ray analysis of lung cancer," in *International Conference on Computer Science, Engineering and Education Applications*, Springer, 2018, pp. 638--647.

[57] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya and others, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225,* 2017.

[58] P. Gang, W. Zhen, W. Zeng, Y. Gordienko, Y. Kochura, O. Alienin, O. Rokovyi and S. Stirenko, "Dimensionality reduction in deep learning for chest X-ray analysis of lung cancer," in *2018 tenth international conference on advanced computational intelligence (ICACI)*, IEEE, 2018, pp. 878--883.

[59] A. Adegun and S. Viriri, "Deep learning techniques for skin lesion analysis and melanoma cancer detection: a survey of state-of-the-art," *Artificial Intelligence Review,* vol. 54, no. 2, pp. 811--841, 2021.

[60] V. Doms, Y. Gordienko, Y. Kochura, O. Rokovyi, O. Alienin and S. Stirenko, "Deep learning for melanoma detection with testing time data augmentation,"

in *The international conference on artificial intelligence and logistics engineering*, 2021.

[61] M. Shulha, Y. Gordienko and S. Stirenko, "Deep Learning with Metadata Augmentation for Classification of Diabetic Retinop," in *Proceedings of Third International Conference on Sustainable Expert Systems . Lecture Notes in Networks and Systems,* , Singapore, 2023.

[62] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 3213-3223*, 2016.

[63] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, ""Vision meets robotics: The kitti dataset."," *The International Journal of Robotics Research ,* vol. 32, no. 11, pp. 1231-1237, 2013.

[64] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, *nuScenes: A multimodal dataset for autonomous driving,* https://doi.org/10.48550/arXiv.1903.11027, 2020.

[65] Q. Ha, K. Watanabe, T. Karasawa, Y. Ushiku and T. Harada, "MFNet: Towards real-time semantic segmentation for autonomous vehicles with multi-spectral scenes," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, 2017.

[66] H. Fu, M. Gong, C. Wang, K. Batmanghelich and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation," in *roceedings. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2018. 2002-2011. 10.1109/CVPR.2018.00214.* , 2018.

[67] J. Chang and Y. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[68] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell and K. Weinberger, "Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving," in *CVPR*, 2019.

[69] A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollar, "Panoptic Segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[70] A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton and G. Corrado, *GAIA-1: A Generative World Model for Autonomous Driving,* https://doi.org/10.48550/arXiv.2309.17080, 2023.

[71] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell and S. Xie, *A ConvNet for the 2020s,* arXiv:2201.03545v2, 2022.

[72] R. A. Berk, "An introduction to sample selection bias in sociological data," *American sociological review, ,* pp. 386--398, 1983.

[73] B. Zadrozny, ""Learning and evaluating classifiers under sample selection bias."," in *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[74] P. Refaeilzadeh, L. Tang and H. Liu, "“Cross-validation.”," in *Encyclopedia of database systems, vol. 5,*, 2009., p. pp. 532–538.

[75] G. Wang, W. Li, S. Ourselin and T. Vercauteren, "Automatic Brain Tumor Segmentation Using Convolutional Neural Networks with Test-Time Augmentation.," in *In book: Crimi A., Bakas S., Kuijf H., Keyvan F., Reyes M., van Walsum T. (eds) Brainlesion: Glioma Multiple Sclerosis, Stroke and Traumatic Brain Injuries. BrainLes 2018. Lecture Notes in Computer Science, vol 11384.*, Springer, 2019.

[76] I. Sobel and G. Feldman, ""A 3x3 Isotropic Gradient Operator for ImageProcessing"," in *Stanford Artificial Intelligence Project(SAIL)*, 1968.

[77] UW-Madison, "UW-Madison GI Tract Image Segmentation," 28 09 2022. [Online]. Available: https://www.kaggle.com/competitions/uw-madison-gi-tract-image-segmentation/overview.

[78] Statistics Online Computational Resource, "Brain Viewer Webapp," University of Michigan, 2006. [Online]. [Accessed 11 12 2023].

[79] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia and L, "TensorFlow: Large-scale machine learning on heterogeneous systems," Google Research, 2015. [Online]. Available: https://www.tensorflow.org. [Accessed 5 10 2023].

[80] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani and S. Chilamkurthy, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," 2019. [Online]. Available: https://pytorch.org/. [Accessed 5 10 2023].

[81] Keras Documentation, "Keras: Deep Learning for humans," [Online]. Available: https://keras.io/. [Accessed 23 11 2023].

[82] National Institutes of Health, "The NIFTI file format," 23 09 2012. [Online]. Available: https://brainder.org/2012/09/23/the-nifti-file-format/. [Accessed 19 11 2023].

[83] Tensorflow, "TFRecord and tf.train.Example," Tensorflow, [Online]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord. [Accessed 10 11 2023].

[84] TensorFlow Official Documentation, "Using the SavedModel format," [Online]. Available: https://www.tensorflow.org/guide/saved_model. [Accessed 19 11 2023].

[85] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization,* arXiv:1412.6980, 2014.

[86] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology, vol 26, n. 3,* pp. 297--302, 1945.

[87] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning, vol. 51, n. 2,* p. 181, 2003.

[88] S. Fu, Y. Lu, Y. Wang, Z. Y. W. Shen, E. Fishman and A. Yuille, ""Domain adaptive relational reasoning for 3d multi-organ segmentation."," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, October 4–8, 2020, Proceedings, Part I 23. Springer International Publishing*, Lima, Peru, 2020.

[89] Z. Jiang, C. Ding, M. Liu and D. Tao, "Two-stage cascaded u-net: 1st place solution to brats challenge 2019 segmentation task.," *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 5th International Workshop, BrainLes 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 17, 2019, Revised Selected Papers, Part I 5 ,* pp. (pp. 231-241), 2019.

[90] T. Henry, A. Carré, M. Lerousseau, T. Estienne, C. Robert, N. Paragios and E. Deutsch, "Brain tumor segmentation with self-ensembled, deeply-supervised 3D U-net neural networks: a BraTS 2020 challenge solution," *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 6th International Workshop, BrainLes 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 4, 2020, Revised Selected Papers, Part I 6,* pp. pp. 327-339, 2020.

[91] Y. Wu and K. He, "Group normalization.," *Proceedings of the European conference on computer vision (ECCV).,* pp. 3-19, 2018.

[92] D. P. Huttenlocher, G. A. Klanderman and W. J. Rucklidge, "Comparing images using the Hausdorff distance.," *IEEE Transactions on pattern analysis and machine intelligence,* vol. 15, no. 9, pp. 850-863., 1993.

[93] A. Der Kiureghian and O. Ditlevsen, "Aleatory or epistemic? Does it matter?.," *Structural safety ,* vol. 31, no. 2, pp. 105-112, 209.

[94] TensorFlow Official Documentation, "TensorFlow Lite | ML for Mobile and Edge Devices," [Online]. Available: https://www.tensorflow.org/lite. [Accessed 23 11 2023].

[95] TFLite Documentation, "Convert TensorFlow models," TensorFlow Official Documentation, [Online]. Available: https://www.tensorflow.org/lite/models/convert/convert_models . [Accessed 23 11 2023].

[96] TensorFlow Lite Documentation, "Performance measurement," TensorFlow Official Documentation, [Online]. Available: https://www.tensorflow.org/lite/performance/measurement. [Accessed 23 11 2023].

[97] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao and K. Keutzer, "SqueezeNext: Hardware-Aware Neural Network Design," *Proceedings of the IEEE conference on computer vision and pattern recognition workshops,* pp. 1638-1647, 2018.

[98] D. Anguita, A. Ghio, S. Ridella and D. Sterpi, "K-Fold Cross Validation for Error Rate Estimate in Support Vector Machines.," in *DMIN*, 2009, pp. 291--297.

[99] B. Baheti, S. Innani, S. Gajre and S. Talbar, "Eff-unet: A novel architecture for semantic segmentation in unstructured environment," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 358--359.

[100] Z. Ghahramani, ""Unsupervised learning."," in *Summer school on machine learning*, Berlin Heidelberg,, Springer , 2003, pp. 72-112.

[101] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research. ,* p. 237–285, 1996.

[102] S. Salehi, S. Mohseni, D. Erdogmus and A. Gholipour, "Tversky loss function for image segmentation using 3D fully convolutional deep networks," in *International workshop on machine learning in medical imaging*, Springer, 2017, pp. 379--387.

ДОДАТОК А.
ПРОГРАМНИЙ КОД

Лістинг А.1 – Використані бібліотеки

```
import os
import enum
import nibabel as nib
import random
import abc
import glob
import functools
import sys
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from IPython.display import Image

import numpy as np
import json
import string
import shutil
import functools
import zipfile
import shutil
import math
import tensorflow as tf; print(f"\t\t- TENSORFLOW VERSION: {tf.__version__}");
import skimage.transform as skTrans
import logging
import collections
import dataclasses
from concurrent.futures import ProcessPoolExecutor
import PIL
from collections import namedtuple
import glob
from keras.models import Model
from keras.layers import Conv3D, MaxPooling3D, UpSampling3D, Dropout
from keras.layers import concatenate, Conv3DTranspose, BatchNormalization
from keras import backend as K
from keras import layers
import keras
import random
import subprocess

from scipy import ndimage
from scipy.ndimage.interpolation import zoom
import pprint
import pandas as pd
from numpy import logical_and as l_and, logical_not as l_not
from scipy.spatial.distance import directed_hausdorff
import time

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logger.info("test")

policyConfig = 'mixed_float16'
policy = tf.keras.mixed_precision.Policy(policyConfig)
tf.keras.mixed_precision.set_global_policy(policy)



class Modes(enum.Enum):
    IMAGE = 1
    VOLUME = 2
```

Лістинг А.2 – Функції попередньої обробки даних

```
    def zscore_normalise(img):
        mask = img != 0
        indices = tf.where(mask)
        boolean_mask_vals = tf.boolean_mask(img, mask)
        img_2 = tf.tensor_scatter_nd_update(img, indices, boolean_mask_vals -
tf.reduce_mean(boolean_mask_vals, axis=-1)) / tf.math.reduce_std(boolean_mask_vals, axis=-1)
        return img_2


    def random_rot(image, label):
        #tf.print(image.shape, label.shape, 'bloody shapes')
        k = np.random.randint(0, 4)
        ax = random.sample(range(3), k=2)

        x = np.rot90(image.numpy(), k, axes=ax)
        y = np.rot90(label.numpy(), k, axes=ax)

        return x, y


    def random_nullify(image):
        image = image.numpy()
        k = np.random.randint(0, 4)
        image[..., k] = 0
        return image

    def normalize(image):
        min_ = np.min(image)
        max_ = np.max(image)
        scale = max_ - min_
        image = (image - min_) / scale
        return image

    def irm_min_max_preprocess(image, low_perc=1, high_perc=99):
        image = image.numpy()
        non_zeros = image > 0
        low, high = np.percentile(image[non_zeros], [low_perc, high_perc])
        image = np.clip(image, low, high)
        image = normalize(image)
        return image

    def aug_noise(image):
        image = image * random.uniform(0.9, 1.1)
        std = np.std(image, axis=(0,1,2))
        noise = tf.stack([np.random.normal(0, s * 0.1) for s in std])
        return image + noise

    def random_rot_flip(image, label):
        k = np.random.randint(0, 4)
        image = np.rot90(image, k)
        label = np.rot90(label, k)
        axis = np.random.randint(0, 2)
        image = np.flip(image, axis=axis).copy()
        label = np.flip(label, axis=axis).copy()
        return image, label

    def random_rotate(image, label):
        angle = np.random.randint(-20, 20)

        image = ndimage.rotate(image, angle, order=0, reshape=False)
        label = ndimage.rotate(label, angle, order=0, reshape=False)
        return image, label

    def rotate_image(image, label, output_size):
        image, label = image.numpy(), label.numpy()
        if random.random() > 0.5:
            image, label = random_rot_flip(image, label)
        elif random.random() > 0.5:
            image, label = random_rotate(image, label)
        x, y = image.shape

        return image, label
```

Лістинг А.3 - Базовий клас для представлення наборів даних

```
        def _bytes_feature(value):
            return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))


        def _float_feature(value):
            return tf.train.Feature(float_list=tf.train.FloatList(value=[value]))


        def _int_feature(value):
            return tf.train.Feature(int64_list=tf.train.Int64List(value=value))


        @dataclasses.dataclass
        class ZipSpec:
            original_path: str
            folder_name: str = ''

            def __post_init__(self):
                if not self.folder_name:
                    self.folder_name = os.path.basename(self.original_path)[:-4] # omit zip

        class BaseDataset(metaclass=abc.ABCMeta):
            max_classes = 4
            split_rate = 0.2
            shape_dims = (3, 3)
            id_len = 100
            name = None
            no_trunk = False
            source_zip = None
            source_tf_records_zip = None
            pixel_channels = 1
            buffer_size = 10
            _labels = None
            min_size = []

            @property
            def num_labels(self):
                return len(self.labels)

            @property
            def labels(self):
                if self.only_meaningful_labels:
                    return ([l for l in self._labels if not l.ignoreInEval or l.id == 0])
                return self._labels


            def __init__(self, work_folder='.', image_shape=None, regenerate=False, training_mode=True,
eval_mode=False,
                         pad_ratio = 2, only_meaningful_labels=False, shuffle=True):
                self.objects = {}
                self.image_shape = image_shape
                self.train_split = set()
                self.val_split = set()
                self.regenerate = regenerate
                self.work_folder = work_folder
                self.training_mode = training_mode
                self.shuffle = shuffle
                self.zip_download_dir = None
                self.pad_ratio = pad_ratio
                self.eval_mode = eval_mode
                self.only_meaningful_labels = only_meaningful_labels

            @property
            def is_ready(self):
                return os.path.exists(self._tf_record_folder)

            @property
            def tf_record_folder(self):
                if not os.path.exists(self._tf_record_folder):
                    os.makedirs(self._tf_record_folder)
```

```
            return self._tf_record_folder

        @property
        def _tf_record_folder(self):
            return None

        @abc.abstractmethod
        def read_metadata(self, download_path):
            return {}

        def _read_metadata(self):
            if not self.is_ready:
                logger.info('no metadata file found, generating new')
                objects = self.read_metadata(self.zip_download_dirs)
                logger.info("data extracted from %s, having %s unique keys", self.zip_download_dirs,
len(objects.keys()))
                meta_file_path = os.path.join(self.tf_record_folder, 'metadata.json')
                logger.info('metadata file location  %s', meta_file_path)

                with open(meta_file_path, 'w+') as fw:
                    json.dump(objects, fw)
                return objects
            meta_file_path = os.path.join(self._tf_record_folder, 'metadata.json')
            if os.path.exists(meta_file_path):
                with open(meta_file_path) as fw:
                    return json.load(fw)

        def proper_pad(self, img, is_eval=False):
            shape = tf.cast(tf.shape(img['y']), tf.float32)
            m_size = tf.convert_to_tensor(self.min_size, dtype=tf.float32)
            pad_ratio = self.pad_ratio
            pad = tf.math.maximum(pad_ratio * tf.math.ceil(shape / pad_ratio), m_size)
            diff_top = tf.math.floor((pad - shape ) / 2.)
            diff_bottom = tf.math.ceil((pad - shape) / 2.)

            mask_pads = tf.cast(tf.transpose(tf.stack([diff_top, diff_bottom])), dtype=tf.int32)
            if self.pixel_channels != 1 or tf.shape(img['x'])[-1] == 1: # channel in separate dimension
                diff_top = tf.concat((diff_top, [0]), 0)
                diff_bottom = tf.concat((diff_bottom, [0]), 0)

            img_pad = tf.cast(tf.transpose(tf.stack([diff_top, diff_bottom])), dtype=tf.int32)
            return {
                'x': tf.pad(img['x'], img_pad),
                'y': tf.pad(img['y'], mask_pads),
                'id': img['id']
            }

        def trunk(self, img):
            if not self.no_trunk:
                shape = tf.shape(img['x'])
                x = img['x']
                if shape[-1] == 1 or self.pixel_channels != 1:
                    x = tf.reduce_sum(img['x'], axis=-1)
                zero = tf.constant(0, dtype=tf.float32)
                where = tf.not_equal(x, zero)
                indices = tf.where(where)
                min_ = tf.math.reduce_min(indices, axis=0)
                max_ = tf.math.reduce_max(indices, axis=0)
                if shape[-1] == 1 or self.pixel_channels != 1:
                    min_img  = tf.concat([min_, [0]], 0)
                    max_img  = tf.concat([max_, [shape[-1]]], 0)
                else:
                    min_img = min_
                    max_img = max_
                return {
                    'x': tf.slice(img['x'], min_img, max_img - min_img),
                    'y': tf.slice(img['y'], min_, max_ - min_),
                    'id': img['id']
                }
            else:
                return img

        def check_and_download(self):
            if os.path.exists(self._tf_record_folder):
                logger.info("Folder with tf records exists, exiting, %s", self._tf_record_folder)
                return
```

```
            tf_records_bname = os.path.basename(self.source_tf_records_zip) if self.source_tf_records_zip else
None
            download_zip_dst = os.path.join(self.work_folder, 'zip', self.name)
            download_tf_dst = os.path.join(self.work_folder, 'tf', self.name)
            logger.info("download folders %s, %s", download_zip_dst, download_tf_dst)
            if self.source_tf_records_zip and os.path.exists(self.source_tf_records_zip):
                logger.info("tfrecords archive exist, downloading")
                if not os.path.exists(download_tf_dst):
                    os.makedirs(download_tf_dst)
                logger.info("Copying from %s to %s", self.source_tf_records_zip, download_tf_dst)
                shutil.copy(self.source_tf_records_zip, download_tf_dst)
                with zipfile.ZipFile(os.path.join(download_tf_dst, tf_records_bname), 'r') as zip_ref:
                    logger.info("Extracting files to %s", self.work_folder)
                    zip_ref.extractall(self.work_folder)
            else:
                if isinstance(self.source_zip, list):
                    source_zips = self.source_zip
                else:
                    source_zips = [self.source_zip]
                self.zip_download_dirs = []

                for zip_file in source_zips:
                    bname = os.path.basename(zip_file.original_path)

                    logger.info("downloading original archive %s to %s", zip_file.original_path,
download_zip_dst)

                    if not os.path.exists(download_zip_dst):
                        os.makedirs(download_zip_dst)
                    if not os.path.exists(os.path.join(download_zip_dst, bname)):
                        shutil.copy(zip_file.original_path, download_zip_dst)
                    else:
                        logger.info("archive %s is already on filesystem", bname)

                    dst = os.path.join(download_zip_dst, zip_file.folder_name)
                    if not os.path.exists(dst):
                        with zipfile.ZipFile(os.path.join(download_zip_dst, bname), 'r') as zip_ref:
                            logger.info("Extracting original files to %s", dst)
                            zip_ref.extractall(download_zip_dst)
                    logger.info("original files location %s", dst)
                    self.zip_download_dirs.append(dst)

        def prepare(self):
            self.check_and_download()
            self.objects = self._read_metadata()
            if not os.path.exists(self.tf_record_folder):
                os.makedirs(self.tf_record_folder)
            self.split_train_val()
            self.write_records()

        def split_train_val(self):
            train_split_file = os.path.join(self.tf_record_folder, 'train.txt')
            val_split_file = os.path.join(self.tf_record_folder, 'val.txt')
            if os.path.exists(val_split_file) and os.path.exists(train_split_file):
                logger.info("Reading existing files %s, %s", train_split_file, val_split_file)
                with open(val_split_file) as fr:
                    self.val_split = set(l.strip() for l in fr.readlines())
                with open(train_split_file) as fr:
                    self.train_split = set(l.strip() for l in fr.readlines())
                return
            logger.info("generating new split files %s, %s", val_split_file, train_split_file)
            self.train_split, self.val_split = self.generate_splits()
            with open(val_split_file, 'w+') as fw:
                fw.write('\n'.join(self.val_split))
            with open(train_split_file, 'w+') as fw:
                fw.write('\n'.join(self.train_split))

        def generate_splits(self):
            keys = self.objects.keys()
            val_amount = math.floor(self.split_rate * len(keys))
            val_set = set(random.choices(list(keys), k=val_amount))
            train_set = set(filter(lambda x: x not in val_set, keys))
            return train_set, val_set

        def _remap_values(self, y):
            labels = self.labels
            n_classes = len(labels)
```

```
            y = y.numpy()
            if self.only_meaningful_labels:
                remap_values = {label.id: i for i, label in enumerate(labels)}
                for label in self._labels:
                    if label.id not in remap_values:
                        y[y==label.id] = 0
                    else:
                        y[y==label.id] = remap_values[label.id]
            return y

        def _to_training_representation(self, x, is_eval=False):
            if not self.training_mode:
                return x
            x, y, id_ = x['x'], x['y'], x['id']
            y = tf.py_function(func=self._remap_values, inp=[y], Tout=tf.int32)
            return self.to_training_representation(x, y, id_, is_eval=is_eval)

        def to_training_representation(self, x, y, id, **kwargs):
            img = x / tf.reduce_max(x)
            return (
                tf.expand_dims(img, axis=-1),
                tf.one_hot(y, axis=-1, depth=self.max_classes)
            )

        def test_ds(self):
            logger.info('dataset training mode %s', self.training_mode)
            for img, y in self.get_train_ds().take(1):
                logger.info('train dataset shapes x=%s,y=%s', img.shape, y.shape)
            for img, y in self.get_val_ds().take(1):
                logger.info('val dataset shapes x=%s,y=%s', img.shape, y.shape)

        @abc.abstractmethod
        def serialize_example(self, img, label) -> tf.train.Example:
            pass

        def _serialize_examples(self, id_, img, label):
            img, label = self.serialize_example(img, label)
            feature = {
                'img_shape': _int_feature(np.array(img.shape, dtype=np.int64)),
                'mask_shape': _int_feature(np.array(label.shape, dtype=np.int64)),
                'id': _bytes_feature(bytes(id_, 'ascii')),
                'mask': _bytes_feature(tf.io.serialize_tensor(label.reshape(-1)).numpy()),
                'image': _bytes_feature(tf.io.serialize_tensor(img.reshape(-1)).numpy())
            }

            return [tf.train.Example(features=tf.train.Features(feature=feature))]

        def write_records(self) -> None:
            total = self.train_split | self.val_split
            for i, outname in enumerate(total):
                dst = os.path.join(self.tf_record_folder, f"{outname}.tfrecords")
                if os.path.exists(dst) and not self.regenerate:
                    continue
                logger.info('writing tfrecord %s', dst)
                obj = self.objects[outname]
                examples = self._serialize_examples(id_=outname, img=obj['image'], label=obj['mask'])

                for example in examples:
                    with tf.io.TFRecordWriter(
                        dst,
                        options= tf.io.TFRecordOptions(compression_type="GZIP")
                    ) as writer:
                        writer.write(example.SerializeToString())

        def parse_example(self, example) -> tf.Tensor:
            X = tf.io.parse_tensor(example['image'], out_type=tf.float32)
            Y = tf.io.parse_tensor(example['mask'], out_type=tf.float32)
            #logger.info('id=%s, img_shape=%s, mask_shape=%',
example['id'],example['img_shape'],example['mask_shape'])
            return {'x': tf.reshape(X, example['img_shape']),
                    'y': tf.dtypes.cast(tf.reshape(Y, example['mask_shape']), tf.int32),
                    'id': example['id']}

        def decode_example(self, record_bytes)-> dict:
            example = tf.io.parse_example(
                record_bytes,
                features = {
```

```
                    'img_shape': tf.io.FixedLenFeature([self.shape_dims[0]], dtype=tf.int64),
                    'mask_shape': tf.io.FixedLenFeature([self.shape_dims[1]], dtype=tf.int64),
                    'id': tf.io.FixedLenFeature([], dtype=tf.string),
                    "mask": tf.io.FixedLenFeature([], dtype=tf.string),
                    'image': tf.io.FixedLenFeature([], dtype=tf.string)
                }
            )
            logger.info('example decoded: %s', example)

            return example

        def get_batched_dataset(self, split, shuffle=True, is_eval=False):
            return self.get_tf_record_dataset(split, shuffle, is_eval)

        def get_tf_record_dataset(self, split, shuffle=True, is_eval=False) -> tf.data.Dataset:
            files = [os.path.join(self.tf_record_folder, f'{fname}.tfrecords')
                     for fname in split]

            dataset = tf.data.Dataset.from_tensor_slices(files)
            if shuffle:
                dataset = dataset.shuffle(self.buffer_size * 2, reshuffle_each_iteration=True)

            dataset = (dataset
                .flat_map(functools.partial(tf.data.TFRecordDataset, buffer_size=self.buffer_size,
compression_type='GZIP'))
                .map(self.decode_example, num_parallel_calls=tf.data.AUTOTUNE)
                .prefetch(tf.data.AUTOTUNE)
                .map(self.parse_example, num_parallel_calls=tf.data.AUTOTUNE)
                .map(self.trunk, num_parallel_calls=tf.data.AUTOTUNE)
                .map(functools.partial(self.proper_pad, is_eval=is_eval), num_parallel_calls=tf.data.AUTOTUNE)
                .map(functools.partial(self._to_training_representation, is_eval=is_eval),
num_parallel_calls=tf.data.AUTOTUNE)
            )
            return dataset

        def analyse_entire_dataset(self):
            files = [os.path.join(self.tf_record_folder, fname)
                     for fname in os.listdir(self.tf_record_folder)
                     if fname not in ['train.txt', 'val.txt', 'metadata.json']]
            class_scores = {}
            total = np.zeros([self.max_classes + 1]) # TODO: remove after brats ds fixed
            logger.info('total files %s', len(files))
            for fname, data in zip(files, tf.data.TFRecordDataset(files, compression_type='GZIP')):
                ex = self.decode_example(data)
                ex = self.parse_example(ex)
                unique = np.unique(ex['y'], return_counts=True)
                logger.info('file %s, having unique vals %s', ex['id'], unique)
                norm_arr = np.zeros([self.max_classes + 1]) # TODO: remove after brats ds fixed
                norm_arr[unique[0].astype(np.uint32)] = unique[1]
                class_scores[fname] =  norm_arr
                total += norm_arr
            print(total)
            plt.bar(np.arange(1, norm_arr.shape[0]), norm_arr[1:])
            plt.show()

        def get_train_ds(self):
            self.prepare()
            return self.get_batched_dataset(self.train_split, shuffle=self.shuffle)

        def get_val_ds(self):
            self.prepare()
            return self.get_batched_dataset(self.val_split, shuffle=self.shuffle, is_eval=True)
```

Лістинг А.4 - Базові класи для роботи з 2ох вимірними даними (ImageDataset), та тривимірними (NiBabelDataset), а також конвертер тривимірного зображення в набір двовимірних (в контексті synapse).

```
class NiBabelDataset(BaseDataset):
    mode = Modes.VOLUME

    def parse_label(self, fpath):
        return np.array(nib.load(fpath).get_fdata(), dtype=np.float32)

    def serialize_example(self, img, label) -> tf.train.Example:
        img = np.array(nib.load(img).get_fdata(), dtype=np.float32)
        label = self.parse_label(label)
        return img, label


class ImageDataset(BaseDataset):
    pixel_channels = 3
    mode = Modes.IMAGE
    expected_shape = (512, 1024)

    def proper_pad(self, img, **kwargs):
        return img

    def to_training_representation(self, x, y, id, **kwargs):
        img = x / 255.
        mask = y
        if self.image_shape:
            img = tf.image.resize_with_pad(img, *self.image_shape, method='nearest')
            mask = tf.image.resize_with_pad(mask, *self.image_shape, method='nearest')
            target_mask_shape = list(self.image_shape) + [self.max_classes]
        else:
            target_mask_shape = list(self.expected_shape) + [self.pixel_channels]

        mask = tf.one_hot(tf.squeeze(mask), axis=-1, depth=self.max_classes)

        return (
            img,
            tf.ensure_shape(mask, target_mask_shape)
        )

    def serialize_example(self, img, label) -> tf.train.Example:
        mask_file = tf.io.read_file(label)
        img_file = tf.io.read_file(img)
        mask_file = tf.io.decode_png(mask_file)
        img_file = tf.io.decode_png(img_file)
        if self.resize_in_tf_record:
            img_file = tf.image.resize(img_file, self.resize_in_tf_record,
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
            mask_file = tf.image.resize(mask_file, self.resize_in_tf_record,
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
        return img_file.numpy().astype(np.float32), mask_file.numpy().astype(np.float32)


class DatasetNibabelToImage(NiBabelDataset, ImageDataset):
    mode = Modes.IMAGE

    def _serialize_examples(self, id_, img, label):
        img, label = NiBabelDataset.serialize_example(self, img, label)
        ishape = tf.shape(img)
        mshape = tf.shape(label)
        imgs = tf.split(img, ishape[-1].numpy(), -1)
        labels = tf.split(label, mshape[-1].numpy(), -1)
        examples = []
        for i, (img, label) in enumerate(zip(imgs, labels)):
            img = tf.squeeze(img)
            label = tf.squeeze(label)
            logger.info('saving slice %s of image %s, img_shape=%s, mask_shape=%s',i, id_, img.shape,
label.shape)
            feature = {
```

```
                'img_shape': _int_feature(np.array(img.shape, dtype=np.int64)),
                'mask_shape': _int_feature(np.array(label.shape, dtype=np.int64)),
                'id': _bytes_feature(bytes(f'{id_}_{i}', 'ascii')),
                'mask': _bytes_feature(tf.io.serialize_tensor(label.numpy().reshape(-1)).numpy()),
                'image': _bytes_feature(tf.io.serialize_tensor(img.numpy().reshape(-1)).numpy())
            }

            examples.append(tf.train.Example(features=tf.train.Features(feature=feature)))
        return examples

    def write_records(self) -> None:
        total = self.train_split | self.val_split
        for i, outname in enumerate(total):
            dst = glob.glob(os.path.join(self.tf_record_folder, f"{outname}_*.tfrecords"))
            if len(list(dst)) and not self.regenerate:
                continue
            #logger.info('writing tfrecords for %s', outname)
            obj = self.objects[outname]
            examples = self._serialize_examples(id_=outname, img=obj['image'], label=obj['mask'])

            for i, example in enumerate(examples):
                with tf.io.TFRecordWriter(
                    os.path.join(self.tf_record_folder, f"{outname}_{i}.tfrecords"),
                    options= tf.io.TFRecordOptions(compression_type="GZIP")
                ) as writer:
                    writer.write(example.SerializeToString())

    def prepare_ds_splits(self, old_split):
        new_split = set()
        for img_id in old_split:
            slices = list(glob.glob(os.path.join(self.tf_record_folder, f'{img_id}_*.tfrecords')))
            for slice_ in slices:
                basename = os.path.splitext(slice_)[0]
                new_split.add(basename)
        logger.info('new split has %s images (was %s before)', len(new_split), len(old_split))
        return new_split

    def prepare(self):
        super().prepare()
        self.train_split = self.prepare_ds_splits(self.train_split)
        self.val_split = self.prepare_ds_splits(self.val_split)
```

Лістинг А.5 - Набір BraTS

```
BratsLabel = namedtuple(
    'BratsLabel', [
        'name',
        'id',
        'color',
        'ignoreInEval',
    ]
)


BRATS_LABELS = [
    BratsLabel('background',  0,  [0, 0, 0],   True),
    BratsLabel('l1',      1,  [128, 0, 0], False),# true
    BratsLabel('l2', 2, [0, 128, 0], False),#true
    BratsLabel('l3', 3,  [128, 128, 0], False),#true
]



class BraTSDataset(NiBabelDataset):
    max_classes = 4
    name = 'brats'
    source_zip = ZipSpec('/content/drive/MyDrive/datasets/BraTS2020_TrainingData.zip')
    source_tf_records_zip = '/content/drive/MyDrive/datasets/brats-tfrecords-full.zip'
    _labels = BRATS_LABELS
    no_background = True
    shape_dims = [4, 3,]
```

```
regenerate = False
pixel_channels = 4
crop = [128, 128, 128, 4]
min_size = [128, 128, 128]
def __init__(self, *args,norm='minmax',**kwargs):
    super().__init__(*args, **kwargs)
    self.norm = norm

def proper_pad(self, img, is_eval=False):
    img = super().proper_pad(img, is_eval=is_eval)
    x, y = img['x'], img['y']
    crop_t = tf.convert_to_tensor(self.crop, dtype=tf.float32)
    if not is_eval:
        shape = tf.cast(tf.shape(x), dtype=tf.float32)
        diff = shape - crop_t
        random_start_points = tf.random.uniform([4,], maxval=diff, dtype=tf.float32)
        random_start_points = tf.cast(tf.math.floor(random_start_points), dtype=tf.int32)
        x = tf.slice(x, random_start_points, tf.cast(crop_t, dtype=tf.int32))
        y = tf.slice(y, random_start_points[:3], tf.cast(crop_t[:3], dtype=tf.int32))

    return {
        'x': x,
        'y': y,
        'id': img['id']
    }


def parse_label(self, fpath):
    data = super().parse_label(fpath)
    data[data == 4.] = 3 # idk why we have label for but no label 3
    return data

@property
def _tf_record_folder(self):
    return '/content/brats-tfrecords-3'

def to_training_representation(self, x, y, id, is_eval=False, **kwargs):
    #img = x / tf.reduce_max(x, axis=[0,1,2])
    if self.norm == 'zscore':
        x = zscore_normalise(x)
    else:
        x = tf.py_function(irm_min_max_preprocess, [x], tf.float32)

    if is_eval is False:
        tta_flips = tf.random.uniform([3,], maxval=[1.,1.,1.]) > 0.5
        if tta_flips[0]:
            x = tf.reverse(x, [0])
            y = tf.reverse(y, [0])
        if tta_flips[1]:
            x = tf.reverse(x, [1])
            y = tf.reverse(y, [1])
        if tta_flips[2]:
            x = tf.reverse(x, [2])
            y = tf.reverse(y, [2])

        p = tf.random.uniform([2,], maxval=[1., 1., ])

        spec1 = tf.TensorSpec(shape=[None, None, None, 4], dtype=tf.float32)
        spec2 = tf.TensorSpec(shape=[None, None, None], dtype=tf.int32)

        x,y = tf.py_function(func=random_rot, inp=[x,y], Tout=[spec1, spec2])
        if p[0] < 0.8:
            x = tf.py_function(aug_noise, [x], tf.float32)
        if p[1] < 0.2:
            x = tf.py_function(random_nullify, [x], tf.float32)

        x = tf.ensure_shape(x, [128, 128, 128, 4])
        y = tf.ensure_shape(y, [128, 128, 128])

    et = y == 3
    tc = tf.math.logical_or(y == 3, y == 1)
    wt = tf.math.logical_or(tc, y == 2)
    y = tf.transpose(tf.stack([et, tc, wt]), [1,2,3,0])
    tf.print(f"loading image {id}")
    return (
        tf.cast(x, dtype=tf.float16),
        tf.cast(y, dtype=tf.float16)
```

```
        )

    def serialize_example(self, img, mask):
        label = self.parse_label(mask)

        arrs = np.zeros(shape=[label.shape[0], label.shape[1], label.shape[2], len(img)])
        for i, p in enumerate(img):
            arrs[..., i] = np.array(nib.load(p).get_fdata(), dtype=np.float16)

        return arrs.astype(np.float16), label.astype(np.float16)

    def read_metadata(self, path):
        path = path[0]
        objects = {}
        for folder in os.listdir(os.path.join(path, 'MICCAI_BraTS2020_TrainingData')):
            if folder == 'BraTS20_Training_355' or not folder.startswith('BraTS'): continue
            objects[folder] = {
                'image': [
                    os.path.join(path, 'MICCAI_BraTS2020_TrainingData', folder, f'{folder}_t1.nii'),
                    os.path.join(path, 'MICCAI_BraTS2020_TrainingData', folder, f'{folder}_t1ce.nii'),
                    os.path.join(path, 'MICCAI_BraTS2020_TrainingData', folder, f'{folder}_t2.nii'),
                    os.path.join(path, 'MICCAI_BraTS2020_TrainingData', folder, f'{folder}_flair.nii'),
                ],
                'mask': os.path.join(path, 'MICCAI_BraTS2020_TrainingData', folder, f'{folder}_seg.nii')
            }
        return objects
```

Лістинг А.6 - Набір synapse, та опис анотацій до нього

```
SynapseLabel = namedtuple(
    'SynapseLabel', [
        'name',
        'id',
        'color',
        'ignoreInEval',
    ]
)


SYNAPSE_LABELS = [
    SynapseLabel('background',  0,  [0, 0, 0],   True),
    SynapseLabel('spleen',      1,  [128, 0, 0], False),# true
    SynapseLabel('right kidney', 2, [0, 128, 0], False),#true
    SynapseLabel('left kidney', 3,  [128, 128, 0], False),#true
    SynapseLabel('gallbladder', 4,  [0, 0, 128], False),# true
    SynapseLabel('esophagus', 5,    [128, 0, 128], True),
    SynapseLabel('liver', 6,        [0, 128, 128], False),#true
    SynapseLabel('stomach', 7, (128, 64,128), False),# true
    SynapseLabel('aorta', 8, [64, 0, 0], False), # true
    SynapseLabel('inferior vena cava', 9, [192, 0, 0], True),
    SynapseLabel('portal vein and splenic vein', 10, [64, 128, 0], True),
    SynapseLabel('pancreas', 11, [192, 128, 0], False),#true
    SynapseLabel('right adrenal gland', 12, [64, 0, 128], True),
    SynapseLabel('left adrenal gland', 13, [192, 0, 128], True),
]



class SynapseDataset2D(DatasetNibabelToImage):
    max_classes = 14
    name = 'synapse_2d'
    pixel_channels = 1
    source_zip = ZipSpec('/content/drive/MyDrive/datasets/RawData.zip')
    source_tf_records_zip = None #'/content/drive/MyDrive/datasets/synapse-tfrecords.zip'
    shape_dims = (2,2)
    no_trunk = True
    source_tf_records_zip = '/content/drive/MyDrive/datasets/synapse2d-tfrecords.zip'
    _labels = SYNAPSE_LABELS

    def __init__(self, *args, g_to_rgb=False, eval_mode=False,  **kwargs):
        DatasetNibabelToImage.__init__(self, *args, **kwargs)
        self.to_rgb = g_to_rgb
```

```
            self.eval_mode = eval_mode

        @tf.function
        def to_training_representation(self, x, y, id, is_eval=False, **kwargs):
            img = tf.clip_by_value(
                x, -125, 275, name=None
            )

            mask = y
            min_, max_ = (tf.math.reduce_min(img), tf.math.reduce_max(img))
            img = (img - min_) / (-min_ + max_)

            mask = tf.expand_dims(mask, axis=-1)
            img = tf.expand_dims(img, axis=-1)

            if self.image_shape:
                img = tf.image.resize_with_pad(img, *self.image_shape, method='nearest')
                mask = tf.image.resize_with_pad(mask, *self.image_shape, method='nearest')
                target_mask_shape = list(self.image_shape) + [self.num_labels]
            else:
                target_mask_shape = list(self.expected_shape) + [self.pixel_channels]

            old_shape = tf.shape(img)

            spec1 = tf.TensorSpec(shape=[None, None], dtype=tf.float32)
            spec2 = tf.TensorSpec(shape=[None, None], dtype=tf.int32)
            if not is_eval:
                img, mask = tf.py_function(
                    func=rotate_image, inp=[tf.squeeze(img), tf.squeeze(mask), old_shape], Tout=[spec1,spec2],
name='rotate_image'
                )
                img = tf.expand_dims(img, -1)

            mask = tf.one_hot(tf.squeeze(mask), axis=-1, depth=self.num_labels)

            return (
                img if not self.to_rgb else tf.image.grayscale_to_rgb(img),
                tf.ensure_shape(mask, target_mask_shape) if self.eval_mode is False else {
                    'mask': tf.ensure_shape(mask, target_mask_shape),
                    'id': id
                }
            )

        def parse_example(self, ex):
            return ImageDataset.parse_example(self, ex)

        @property
        def _tf_record_folder(self):
            return '/content/synapse2d-tfrecords'

        def read_metadata(self, path):
            path = path[0]
            train_path = os.path.join(path, 'Training/img')
            label_path = os.path.join(path, 'Training/label')

            objects = {}
            for fname in os.listdir(train_path):
                id_ = fname[3:]
                objects[id_[:4]] = {
                    'image': os.path.join(train_path, fname),
                    'mask': os.path.join(label_path, f'label{id_}')
                }
            return objects
```

Лістинг А.7 - Набір CityScapes та його аннотації

```
Label = namedtuple( 'Label' , [
    'name'        ,
    'id'          ,
    'trainId'     ,
    'category'    ,
    'categoryId'  ,
```

```
            'hasInstances',
            'ignoreInEval',
            'color'        ,
        ])

        LABELS = [
            #       name                     id    trainId   category            catId     hasInstances
ignoreInEval   color
            Label(  'unlabeled'            ,  0 ,      255 , 'void'            , 0       , False        , True
, (  0,  0,  0) ),
            Label(  'ego vehicle'          ,  1 ,      255 , 'void'            , 0       , False        , True
, (  0,  0,  0) ),
            Label(  'rectification border' ,  2 ,      255 , 'void'            , 0       , False        , True
, (  0,  0,  0) ),
            Label(  'out of roi'           ,  3 ,      255 , 'void'            , 0       , False        , True
, (  0,  0,  0) ),
            Label(  'static'               ,  4 ,      255 , 'void'            , 0       , False        , True
, (  0,  0,  0) ),
            Label(  'dynamic'              ,  5 ,      255 , 'void'            , 0       , False        , True
, (111, 74,  0) ),
            Label(  'ground'               ,  6 ,      255 , 'void'            , 0       , False        , True
, ( 81,  0, 81) ),
            Label(  'road'                 ,  7 ,        0 , 'flat'            , 1       , False        , False
, (128, 64,128) ),
            Label(  'sidewalk'             ,  8 ,        1 , 'flat'            , 1       , False        , False
, (244, 35,232) ),
            Label(  'parking'              ,  9 ,      255 , 'flat'            , 1       , False        , True
, (250,170,160) ),
            Label(  'rail track'           , 10 ,      255 , 'flat'            , 1       , False        , True
, (230,150,140) ),
            Label(  'building'             , 11 ,        2 , 'construction'    , 2       , False        , False
, ( 70, 70, 70) ),
            Label(  'wall'                 , 12 ,        3 , 'construction'    , 2       , False        , False
, (102,102,156) ),
            Label(  'fence'                , 13 ,        4 , 'construction'    , 2       , False        , False
, (190,153,153) ),
            Label(  'guard rail'           , 14 ,      255 , 'construction'    , 2       , False        , True
, (180,165,180) ),
            Label(  'bridge'               , 15 ,      255 , 'construction'    , 2       , False        , True
, (150,100,100) ),
            Label(  'tunnel'               , 16 ,      255 , 'construction'    , 2       , False        , True
, (150,120, 90) ),
            Label(  'pole'                 , 17 ,        5 , 'object'          , 3       , False        , False
, (153,153,153) ),
            Label(  'polegroup'            , 18 ,      255 , 'object'          , 3       , False        , True
, (153,153,153) ),
            Label(  'traffic light'        , 19 ,        6 , 'object'          , 3       , False        , False
, (250,170, 30) ),
            Label(  'traffic sign'         , 20 ,        7 , 'object'          , 3       , False        , False
, (220,220,  0) ),
            Label(  'vegetation'           , 21 ,        8 , 'nature'          , 4       , False        , False
, (107,142, 35) ),
            Label(  'terrain'              , 22 ,        9 , 'nature'          , 4       , False        , False
, (152,251,152) ),
            Label(  'sky'                  , 23 ,       10 , 'sky'             , 5       , False        , False
, ( 70,130,180) ),
            Label(  'person'               , 24 ,       11 , 'human'           , 6       , True         , False
, (220, 20, 60) ),
            Label(  'rider'                , 25 ,       12 , 'human'           , 6       , True         , False
, (255,  0,  0) ),
            Label(  'car'                  , 26 ,       13 , 'vehicle'         , 7       , True         , False
, (  0,  0,142) ),
            Label(  'truck'                , 27 ,       14 , 'vehicle'         , 7       , True         , False
, (  0,  0, 70) ),
            Label(  'bus'                  , 28 ,       15 , 'vehicle'         , 7       , True         , False
, (  0, 60,100) ),
            Label(  'caravan'              , 29 ,      255 , 'vehicle'         , 7       , True         , True
, (  0,  0, 90) ),
            Label(  'trailer'              , 30 ,      255 , 'vehicle'         , 7       , True         , True
, (  0,  0,110) ),
            Label(  'train'                , 31 ,       16 , 'vehicle'         , 7       , True         , False
, (  0, 80,100) ),
            Label(  'motorcycle'           , 32 ,       17 , 'vehicle'         , 7       , True         , False
, (  0,  0,230) ),
            Label(  'bicycle'              , 33 ,       18 , 'vehicle'         , 7       , True         , False
, (119, 11, 32) ),
```

```
        Label(  'license plate'        , -1 ,       -1 , 'vehicle'         , 7       , False        , True
, (  0,  0,142) ),
    ]

    SMALL_CLASSES = [34, 33, 32, 24, 23, 19, 18, 17, 16, 15, 14, 13, 12]
    BIG_CLASSES = [i for i in range(35) if i not in SMALL_CLASSES]


    class CityScapes(ImageDataset):
        max_classes = 35
        resize_in_tf_record = (512, 1024)
        shape_dims = (3, 3)
        buffer_size = 100
        name = 'cityscapes'
        no_trunk = True
        source_zip = [ZipSpec('/content/drive/MyDrive/datasets/gtFine_trainvaltest.zip', 'gtFine'),
                      ZipSpec('/content/drive/MyDrive/datasets/leftImg8bit_trainvaltest.zip', 'leftImg8bit')]
        source_tf_records_zip = '/content/drive/MyDrive/datasets/cityscapes-tfrecords.zip'

        _labels = LABELS

        @property
        def _tf_record_folder(self):
            return '/content/cityscapes-tfrecords'

        def generate_splits(self):
            return self.ds_mapping['train'], self.ds_mapping['val']

        def read_metadata(self, download_paths):
            annotation_folder = download_paths[0]
            images_folder = download_paths[1]
            self.ds_mapping = collections.defaultdict(set)
            dataset_mapping = collections.defaultdict(dict)

            for fname in glob.glob(os.path.join(annotation_folder, '**', '*_labelIds.png'), recursive=True):
                basename = os.path.basename(fname)
                city = os.path.basename(os.path.dirname(fname))
                ds_type = os.path.basename(os.path.dirname(os.path.dirname(fname)))
                img_id = basename[:-len('_gtFine_labelIds.png')]
                dataset_mapping[img_id]['mask'] = fname
                self.ds_mapping[ds_type].add(img_id)

            for fname in glob.glob(os.path.join(images_folder, '**', '*.png'), recursive=True):
                basename = os.path.basename(fname)
                city = os.path.basename(os.path.dirname(fname))
                ds_type = os.path.basename(os.path.dirname(os.path.dirname(fname)))
                img_id = basename[:-len('_leftImg8bit.png')]
                dataset_mapping[img_id]['image'] = fname
            return dataset_mapping
```

Лістинг А.8 - Додатковий код для представлення моделей

```
    class NpEncoder(json.JSONEncoder):
        def default(self, obj):
            if isinstance(obj, np.integer):
                return int(obj)
            if isinstance(obj, np.floating):
                # 👇 alternatively use str()
                return float(obj)
            if isinstance(obj, np.ndarray):
                return obj.tolist()
            return json.JSONEncoder.default(self, obj)



    class CustomModel(keras.Model):
        n = 0

        def evaluate(self, *args, **kwargs):
            logs = keras.Model.evaluate(self, *args, **kwargs)
            self.n += 1
            data = {}
            if os.path.exists(f'runtime_stats_{self.name}.json'):
```

```
            with open(f'runtime_stats_{self.name}.json',) as fr:
                data = json.load(fr)
        data[self.n] = logs
        with open(f'runtime_stats_{self.name}.json', 'w') as fw:
            json.dump(data, fw, cls=NpEncoder)
        return logs
```

Лістинг А.9 - Код, що реалізує запропоновані архітектури (з коефіцієнтом розширення та DeSSCo)

```
from keras.src.backend import conv2d

class UnetModifications(enum.Enum):
    REGULAR = 1
    DESSCO = 2
    DESSCO2 = 3
    DESSCOB = 3
    MUNET = 4
    ATTENTION = 5
    RESIDUAL = 6
    DESSCO_2C = 7
    SIGMOID_RESULT = 8
    GROUP_NORM = 9
    NO_DESSCO_NORMS = 10
    DESSCO_INSTANCE_NORMS = 11


def attention_block(skips, gating, inter_shape, name, mode: Modes = Modes.IMAGE):
    conv1 = layers.Conv2D if mode == Modes.IMAGE \
        else layers.Conv3D
    conv2 = layers.Conv2D if mode == Modes.IMAGE \
        else layers.Conv3D
    conv3 = layers.Conv2D if mode == Modes.IMAGE \
        else layers.Conv3D

    gating = layers.UpSampling3D(name=f'ag-upsample-{inter_shape}')(gating)
    phi_g = conv1(inter_shape, 1, padding='same', name=f'ag-gate-{inter_shape}')(gating)

    theta_x = conv2(inter_shape, 1, padding='same', name=f'ag-query-{inter_shape}')(skips)  # 16
    concat_xg = layers.add([phi_g, theta_x], name=f'ag-concat-{inter_shape}')

    act_xg = layers.Activation('relu', name=f'ag-act1-{inter_shape}')(concat_xg)
    psi = conv3(1, 1, padding='same', name=f'ag-psi-{inter_shape}')(act_xg)
    sigmoid_xg = layers.Activation('sigmoid', name=f'ag-psi-act-{inter_shape}')(psi)

    y = layers.multiply([sigmoid_xg, skips], name=f'ag-mul-{inter_shape}')

    return y


def down(filters, activation, x, expansion_rate=2., mode: Modes = Modes.IMAGE, modifications = None):
    modifications = modifications if modifications is not None else []
    conv1 = layers.Conv2D if mode == Modes.IMAGE \
        else layers.Conv3D
    conv2 = layers.Conv2D if mode == Modes.IMAGE \
        else layers.Conv3D
    norm1 = layers.BatchNormalization( name=f'e-bn1-{filters}') \
        if UnetModifications.GROUP_NORM not in modifications else \
        tf.keras.layers.GroupNormalization(groups=16, name=f'e-gn1-{filters}')
    norm2 = layers.BatchNormalization( name=f'e-bn2-{filters}') \
        if UnetModifications.GROUP_NORM not in modifications else \
        tf.keras.layers.GroupNormalization(groups=16, name=f'e-gn2-{filters}')

    x = conv1(filters, 3, padding="same", name=f'e-conv1-{filters}')(x)
    x = norm1(x)
    x = activation( name=f'e-act1-{filters}')(x)

    x = conv2(filters, 3,  padding="same", name=f'e-conv2-{filters}')(x)
    x = norm2(x)
    x = activation( name=f'e-act2-{filters}')(x)
    return x


def up(filters, activation, skip, x, dropout=None, mode: Modes = Modes.IMAGE, modifications = None):
```

```
        modifications = modifications if modifications is not None else []

        conv1 = layers.Conv2D if mode == Modes.IMAGE \
            else layers.Conv3D
        conv2 = layers.Conv2D if mode == Modes.IMAGE \
            else layers.Conv3D
        norm1 = layers.BatchNormalization( name=f'd-bn1-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'd-gn1-{filters}')
        norm2 = layers.BatchNormalization( name=f'd-bn2-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'd-gn2-{filters}')

        up_lr = layers.UpSampling3D if mode == Modes.VOLUME else layers.UpSampling2D

        x = up_lr(name=f'upsample-{filters}')(x)
        x = layers.Concatenate(name=f'concat-{filters}')([skip, x])

        x = conv1(filters, 3, padding="same", name=f'd-conv1-{filters}')(x)
        x = norm1(x)
        x = activation(name=f'd-act1-{filters}')(x)

        x = conv2(filters, 3, padding="same", name=f'd-conv2-{filters}')(x)
        x = norm2(x)
        x = activation(name=f'd-act2-{filters}')(x)

        return x


    def munet(x, activation, filters, dropout=None,  mode: Modes = Modes.IMAGE):
        conv1 = layers.Conv2D if mode == Modes.IMAGE else layers.Conv3D
        x = conv1(filters, 3, padding='same')(x)
        x = layers.BatchNormalization()(x)
        x = activation()(x)
        if dropout:
            x = layers.Dropout(dropout)(x)
        return x


    def desscob_unet(x, activation, filters, dropout=None, mode: Modes = Modes.IMAGE, modifications:list =
None):
        modifications = modifications if modifications else []
        if mode == Modes.VOLUME:
            dconv = layers.Conv3D(filters, 3,
                                  groups=filters,
                                  padding='same', name=f'dessco2-dw-{filters}')
            conv1 = layers.Conv3D(2 * filters, 1, padding='same',
                                  name=f'dessco2-conv1-{filters}')
            conv2 = layers.Conv3D(filters, 1, padding='same',
                                  name=f'dessco2-conv2-{filters}')
        else:
            conv1 = layers.Conv2D(2 * filters, 1,
                                  padding='same', name=f'dessco2-conv1-{filters}')
            conv2 = layers.Conv2D(filters, 1, padding='same',
                                  name=f'dessco2-conv2-{filters}')

            dconv = layers.DepthwiseConv2D(3, padding='same', name=f'dessco-dw-{filters}')

        norm1 = layers.BatchNormalization( name=f'dessco2-bn1-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications or UnetModifications.DESSCO_INSTANCE_NORMS in
modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'dessco2-gn1-{filters}')
        norm2 = layers.BatchNormalization( name=f'dessco2-bn2-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications or UnetModifications.DESSCO_INSTANCE_NORMS in
modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'dessco2-gn2-{filters}')
        norm3 = layers.BatchNormalization( name=f'dessco2-bn3-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications or UnetModifications.DESSCO_INSTANCE_NORMS in
modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'dessco2-bn3-{filters}')

        x = dconv(x)
        if UnetModifications.NO_DESSCO_NORMS not in modifications:
            x = norm1(x)
        x = activation()(x)

        x = conv1(x)
        if UnetModifications.NO_DESSCO_NORMS not in modifications:
```

```
            x = norm2(x)
        x = activation()(x)

        x = conv2(x)
        if UnetModifications.NO_DESSCO_NORMS not in modifications:
            x = norm3(x)
        x = activation()(x)

        if dropout:
            x = layers.Dropout(dropout)(x)
        return x


    def dessco_unet(x, activation, filters, dropout=None, mode: Modes = Modes.IMAGE, modifications: list =
None):
        modifications = modifications if modifications else []
        if mode == Modes.VOLUME:
            groups = filters if UnetModifications.DESSCO_2C not in modifications else filters // 2
            dconv = layers.Conv3D(filters, 3,
                                  activation='relu',
                                  groups=groups,
                                  padding='same', name=f'dessco-dw-{filters}')
            conv = layers.Conv3D(filters, 1, activation='relu', padding='same',
                                 name=f'dessco-conv-{filters}')
        else:
            conv = layers.Conv2D(filters, 1, activation='relu',
                                 padding='same', name=f'dessco-conv-{filters}')
            dconv = layers.DepthwiseConv2D(3, activation='relu',
                                           padding='same', name=f'dessco-dw-{filters}')

        norm1 = layers.BatchNormalization( name=f'dessco-bn1-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications or UnetModifications.DESSCO_INSTANCE_NORMS in
modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'dessco-gn1-{filters}')
        norm2 = layers.BatchNormalization( name=f'dessco-bn2-{filters}') \
            if UnetModifications.GROUP_NORM not in modifications or UnetModifications.DESSCO_INSTANCE_NORMS in
modifications else \
            tf.keras.layers.GroupNormalization(groups=16, name=f'dessco-gn2-{filters}')

        x = dconv(x)
        if UnetModifications.NO_DESSCO_NORMS not in modifications:
            x = norm1(x)
        x = activation()(x)

        x = conv(x)
        if UnetModifications.NO_DESSCO_NORMS not in modifications:
            x = norm2(x)
        x = activation()(x)

        if dropout:
            x = layers.Dropout(dropout)(x)
        return x


    def get_unet_model(channels, num_classes, pooling='max',
                       dropout=None,
                       activation=layers.ReLU,
                       expansion_rate=2.,
                       start_filters=32.,
                       depth=4,
                       last_layer='softmax',
                       mode: Modes = Modes.IMAGE,
                       modifications: list = [UnetModifications.REGULAR],
                       name=''):
        input_size = (None, None, channels) if mode == Modes.IMAGE else (None, None, None, channels)
        inputs = x = keras.Input(shape=input_size, name='input')
        kernel_sz = 3
        stack = []
        filter_list = [int(start_filters * (expansion_rate ** i)) for i in range(depth)]
        for filters in filter_list[:-1]:
            down_op = layers.MaxPooling2D if mode == Modes.IMAGE else layers.MaxPooling3D
            x = down(filters, activation, x, expansion_rate=expansion_rate, mode=mode,
modifications=modifications)

            stack.append(x)
            x = down_op(2, strides=2, dtype='float32', name=f'e-pool-{filters}')(x)
```

```
            if dropout:
                x = layers.Dropout(dropout)(x)

        x = down(filter_list[-1], activation, x, expansion_rate=expansion_rate,mode=mode,
modifications=modifications)

        reversed_filters = filter_list[:-1][::-1]
        for filters in reversed_filters:
            skip = stack.pop()
            residual = skip
            logger.info('applying modification %s to filter level %s', modifications, filters)
            if UnetModifications.MUNET in modifications:
                skip = munet(skip, activation, filters,mode=mode)
            if UnetModifications.DESSCO in modifications:
                skip = dessco_unet(skip, activation, filters, mode=mode, modifications=modifications)
            if UnetModifications.DESSCO_2C in modifications:
                skip = dessco_unet(skip, activation, 2 * filters, mode=mode, modifications=modifications)

            if UnetModifications.DESSCOB in modifications:
                skip = desscob_unet(skip, activation, filters, mode=mode, modifications=modifications)
            if UnetModifications.RESIDUAL in modifications:
                skip = skip + residual

            if UnetModifications.ATTENTION in modifications:
               skip = attention_block(skips=skip, gating=x, inter_shape=filters,mode=mode, name=f'att-
{filters}')
            x = up(filters, activation, skip, x, dropout=dropout,mode=mode, modifications=modifications)

            if dropout:
                x = layers.Dropout(dropout)(x)
        final_layer = layers.Conv2D if mode == Modes.IMAGE else layers.Conv3D
        # Add a per-pixel classification layer
        if UnetModifications.SIGMOID_RESULT in modifications:
            outputs = final_layer(num_classes - 1, 1, padding="same", dtype='float32', activation='sigmoid',
name='score-vector')(x)
        else:
            outputs = final_layer(num_classes, 1, padding="same", dtype='float32', activation=last_layer,
name='score-vector')(x)
        model = CustomModel(inputs, outputs, name=name)
        return model
```

Лістинг А.10 - Допоміжні класи для різних метрик та функцій втрат

```
import gc


class GarbageCollectorCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        gc.collect()
        tf.keras.backend.clear_session()


class DiceCoef(tf.keras.metrics.Metric):

    def __init__(self, name='dice_coef_metric', ignore_bg=False,  argmax=False, **kwargs):
        super(DiceCoef, self).__init__(name=name, **kwargs)
        self.intersection = self.add_weight('intersection', initializer='zeros')
        self.union = self.add_weight('union', initializer='zeros')
        self.smooth = 1.
        self.argmax = argmax
        self.ignore_bg = ignore_bg
        #self.cache = []

    def reset_state(self):
        self.intersection.assign(0)
        self.union.assign(0)
        #self.cache = []

    def get_config(self):
        base_config = super().get_config()
        config = {
```

```
                "argmax": self.argmax,
            }
            return {**base_config, **config}

        @classmethod
        def from_config(cls, config):
            sublayer_config = config.pop("argmax", False)
            sublayer = keras.saving.deserialize_keras_object(sublayer_config)
            return cls(argmax=sublayer, **config)

        def update_state(self, y_true, y_pred, sample_weight=None):
            if self.argmax:
                y_true = tf.one_hot(tf.argmax(y_true, axis = -1), depth=4)
                y_pred = tf.one_hot(tf.argmax(y_pred, axis = -1), depth=4)
            if not self.ignore_bg:
                y_true, y_pred = y_true[...,1:], y_pred[...,1:]

            intersection = tf.math.reduce_sum( y_true * y_pred)
            union = tf.math.reduce_sum(y_true) + tf.math.reduce_sum(y_pred)

            self.intersection.assign_add(intersection)
            self.union.assign_add(union)

        def result(self):
            dice = tf.math.reduce_mean((2. * self.intersection + self.smooth)/(self.union + self.smooth))
            return dice



    class BinaryDice(tf.keras.metrics.Metric):
        def __init__(self, *args,class_id=None, **kwargs):
            super(BinaryDice, self).__init__(*args, **kwargs)
            self.class_id = class_id
            self.i = self.add_weight('i', initializer='zeros')
            self.u = self.add_weight('u', initializer='zeros')

        def update_state(self, y_true, y_pred):
            y_true = tf.reshape(y_true, -1)
            y_pred = tf.reshape(y_pred, -1)

            yt = tf.cast(y_true == self.class_id, tf.int16)
            yp = tf.cast(y_pred == self.class_id, tf.int16)

            intersection = np.sum( yt * yp)
            union = np.sum(yt) + np.sum(yp)
            #logger.info("class %s: DC = %s (i=%s, u=%s) - shapes now %s (before %s)",  self.class_id, (2 *
intersection + 1e-5) / (union + 1e-5), intersection, union, yt.shape, y_true.shape)

            self.i.assign_add(intersection)
            self.u.assign_add(union)

        def result(self):
            smooth = 1e-5
            return (2 * self.i + smooth) / (self.u + smooth)

        def reset_state(self):
            self.i.assign(0)
            self.u.assign(0)


    class DiceCoefClasswise(tf.keras.metrics.Metric):
        def __init__(self, *args, **kwargs):
            super(DiceCoefClasswise, self).__init__(*args, **kwargs)
            self.s = self.add_weight('s', initializer='zeros')
            self.t = self.add_weight('t', initializer='zeros')

        def update_state(self, y_true, y_pred):
            intersection = tf.math.reduce_sum( y_true * y_pred, axis=[1,2])
            smooth = 1e-5
            union = tf.math.reduce_sum(y_true, axis=[1,2]) + tf.math.reduce_sum(y_pred, axis=[1,2])
            self.s.assign_add(tf.reduce_mean((2. * intersection + smooth)/(union + smooth)))
            self.t.assign_add(1)

        def result(self):
            return self.s / self.t

        def reset_state(self):
            self.s.assign(0)
```

```
            self.t.assign(0)


    def combo_loss():
        loss_1 = tf.keras.losses.CategoricalCrossentropy(from_logits=False)
        loss_2 = DiceCoefClasswise(name='dc_cw')
        def _( y_true, y_pred):
            return 0.5 * loss_1(y_true, y_pred) + 0.5 * loss_2(y_true, y_pred)
        return _


    def dice_loss(y_true, y_pred, smooth=1.):
        i = tf.reduce_sum(y_true * y_pred)
        u = tf.reduce_sum(y_true) + tf.reduce_sum(y_pred)
        return 1 - 2 * (i + smooth) / (u + smooth)


    def combo_loss_2():
        loss_1 = tf.keras.losses.CategoricalCrossentropy(from_logits=False)
        loss_2 = tf.keras.losses.BinaryCrossentropy(from_logits=False)
        def _( y_true, y_pred):
            return loss_1(y_true, y_pred) + loss_2(y_true, y_pred)
        return _
```

Лістинг А.11 - Допоміжна функція для завантаження моделі з диску

```
    WEIGHTS_FOLDER = '/content/drive/MyDrive/datasets/weights'

    def load_model(model_name):
        cpkt_dst = os.path.join(WEIGHTS_FOLDER, f'{model_name}.cpkt')
        logger.info('weights supposed to be here %s', cpkt_dst)
        if os.path.exists(cpkt_dst):
            logger.info(f'weights exist, loading %s', f'{model_name}.cpkt')
            _model = tf.keras.models.load_model(cpkt_dst,
                                                compile=False,
                                                custom_objects={'DiceCoef':    DiceCoef,    'dice_coef_metric':
DiceCoef, 'dc_argmax': DiceCoef})
            _model.summary()
            return _model
        else:
            raise Exception(f"model {model_name} is not found")
```

Лістинг А.12 - Клас Trainer

```
    @dataclasses.dataclass
    class SchedulerOpts:
        min_lr: int = 1e-3
        max_lr: int = 1e-6
        epochs: int = 50

        def scheduler_func(self):
            def scheduler(epoch, lr):
                return self.min_lr - epoch * (self.min_lr - self.max_lr)/self.epochs
            return scheduler



    class Trainer:
        weights_folder = '/content/drive/MyDrive/datasets/weights'
        def __init__(self, model, dataset: BaseDataset, loss=None, no_model_save=False,
                     metrics=None, callbacks=None, epochs=50, batch_size=1, eager=False, retrain=False,
                     optimizer=None,  keep_lr=False,  monitor_metric='val_dice_coef_metric',validation_freq=1,
validation_steps=None,
                     cache=False, lr=0.0001, opt_mode = 'max', prefetch=False,
                     scheduler: SchedulerOpts = None):
            self.model = model
            self.batch_size = batch_size
            self.experiment_name = model.name
            self.dataset = dataset
            self.keep_lr = keep_lr
            self.eager = eager
            self.monitor_metric = monitor_metric
            self.opt_mode = opt_mode
```

```
            self.retrain = retrain
            self.optimizer = tf.keras.optimizers.Adam(0.0001) if not optimizer else optimizer
            self.loss = loss if loss else tf.keras.losses.BinaryCrossentropy(from_logits=False)
            self.no_model_save = no_model_save
            self.cache = cache
            self.prefetch = prefetch
            self.scheduler = scheduler
            # iou = tf.keras.metrics.MeanIoU(dataset.max_classes, ignore_class=0,
            #                                sparse_y_true=False, sparse_y_pred=False)
            self.metrics = metrics if metrics else [DiceCoef(), DiceCoef(argmax=True, name='dc-argmax'),
                                                    tf.keras.metrics.OneHotIoU(dataset.num_labels,
                                                                               name='iou',
                                                                               target_class_ids=range(1,
dataset.num_labels))]
            self.callbacks = callbacks if callbacks else []
            self.epochs = epochs
            self.validation_freq = validation_freq
            self.validation_steps = validation_steps

        def wrap_loss(self, loss):
            @tf.function
            def _(y_t, y_p):
                logger.info('yt=%s, yp=%s', y_t.shape, y_p.shape)
                return loss(y_t, y_p)
            return _
        def backup_previous_weights(self):
            cpkt_dst = os.path.join(self.weights_folder, f'{self.experiment_name}.cpkt')
            stat_file = os.path.join(self.weights_folder, f'{self.experiment_name}.json')
            run_stat_file = os.path.join(self.weights_folder, f'runtime_stats_{self.experiment_name}.json')

            backup =  os.path.join(self.weights_folder, 'backup')
            back_files = [cpkt_dst, run_stat_file, stat_file]
            for f in back_files:
                if os.path.exists(f):
                    logger.info('backing up file %s, is dir %s', f, os.path.isdir(f))
                    dst = os.path.basename(f)
                    if os.path.isdir(f):
                        shutil.copytree(f, os.path.join(backup, dst), dirs_exist_ok=True)
                    else:
                        shutil.copy(f, os.path.join(backup, dst))

        def train(self):
            train_ds = self.dataset.get_train_ds().batch(self.batch_size)
            val_ds = self.dataset.get_val_ds().batch(1)
            if self.cache:
                train_ds = train_ds.cache()
                val_ds = val_ds.cache()
            if self.prefetch:
                for i, imgs in enumerate(val_ds):
                    logger.info("prefetching validation image %s", i)
                for i, imgs in enumerate(train_ds):
                    logger.info("prefetching train image %s", i)
            cpkt_dst = os.path.join(self.weights_folder, f'{self.experiment_name}.cpkt')
            logger.info('weights supposed to be here %s', cpkt_dst)
            if os.path.exists(cpkt_dst) and not self.retrain:
                logger.info(f'weights exist, loading %s', f'{self.experiment_name}.cpkt')
                self.model = tf.keras.models.load_model(cpkt_dst,
                                                        compile=False,
                                                        custom_objects={'DiceCoef':                    DiceCoef,
'dice_coef_metric': DiceCoef, 'dc_argmax': DiceCoef})
                self.backup_previous_weights()
                #self.model.load_weights(f'/content/drive/MyDrive/datasets/{self.experiment_name}.cpkt')
                self.model.save(cpkt_dst)
            _gc_cb = GarbageCollectorCallback()

            callbacks = [_gc_cb]
            if self.scheduler is None:
                _es_cb = tf.keras.callbacks.EarlyStopping(monitor=self.monitor_metric,  patience=6, verbose=1,
mode=self.opt_mode, restore_best_weights=True)
                callbacks.append(_es_cb)
            if not self.no_model_save:
                _ckpt_cb = tf.keras.callbacks.ModelCheckpoint(cpkt_dst,
                                                              monitor=self.monitor_metric, mode=self.opt_mode,
                                                              save_weights_only=False,
                                                              save_best_only=True, options=None,
                                                              verbose=1)
                callbacks.append(_ckpt_cb)
```

```
        if self.keep_lr is False and self.scheduler is None:
            _lr_cb = tf.keras.callbacks.ReduceLROnPlateau(monitor=self.monitor_metric, factor=0.5,
                                                          patience=2, verbose=1, mode=self.opt_mode,
                                                                min_delta=0.00001)
            callbacks.append(_lr_cb)

        if self.scheduler:
            _scheduler_cp = tf.keras.callbacks.LearningRateScheduler(self.scheduler.scheduler_func())
            callbacks.append(_scheduler_cp)

        self.model.summary()
        gc.collect()
        callbacks.extend(self.callbacks)
        self.model.compile(optimizer=self.optimizer, loss=self.loss, metrics=self.metrics,
                           run_eagerly=self.eager)
        #self.model.evaluate(val_ds)
        if not self.no_model_save:
            self.model.save(cpkt_dst)

        history = self.model.fit(train_ds, validation_data=val_ds,
                                 validation_freq=self.validation_freq,
                                 validation_steps=self.validation_steps,
                                 epochs=self.epochs, callbacks=callbacks,)
        data_to_dump = {}
        data_to_dump.update(history.history)
        with open(f'/content/drive/MyDrive/datasets/{self.experiment_name}.json', 'w+') as fw:
            json.dump(data_to_dump, fw, cls=NpEncoder)

    def train_kfold(self):
        pass
```

Лістинг А.13 – класи, що реалізують процедуру отримання метрик перевірки.

```
from medpy import metric


class BaseEvaluator:
    weights_folder = '/content/drive/MyDrive/datasets/weights'

    def __init__(self, dataset: BaseDataset, model_name: str):
        self.dataset = dataset
        self.model_name = model_name
        self._model = None

    @property
    def model(self):
        if not self._model:
            self._model = load_model(self.model_name)
        return self._model

    @abc.abstractmethod
    def evaluate(self):
        pass


def calculate_metric_percase(pred, gt):
    if tf.reduce_sum(pred) > 0 and tf.reduce_sum(gt)>0:
        dice = metric.binary.dc(pred.numpy(), gt.numpy())
        hd95 = metric.binary.hd95(pred.numpy(), gt.numpy())
        return dice, hd95
    elif tf.reduce_sum(pred) > 0 and tf.reduce_sum(gt)==0:
        return 1, 0
    else:
        return 0, 0



class SynapseEvaluator(BaseEvaluator):

    def evaluate(self):
        count = 0
        self.dataset.prepare()
        val_split = self.dataset.val_split
        vol_to_img = collections.defaultdict(list)
        for key in val_split:
            img_id, slice_id = key.split('_')
```

```
                vol_to_img[img_id].append(key)

            meaningful_labels = [label for label in self.dataset.labels]
            total_metrics = np.zeros(shape=(len(meaningful_labels) - 1, 2))
            logger.info('processing keys %s', vol_to_img.keys())
            results = {}
            for volume_id, slices in vol_to_img.items():
                slices.sort(key=lambda x: int(x.split('_')[1]))
                logger.info('processing volume %s: %s', volume_id, slices)
                volume_pred = np.zeros(shape=[len(slices)] + list(self.dataset.image_shape))
                volume_mask = np.zeros(shape=[len(slices)] + list(self.dataset.image_shape))
                volume_img = np.zeros(shape=[len(slices)] + list(self.dataset.image_shape))

                slice_index = 0
                count += 1
                for img, mask in self.dataset.get_batched_dataset(slices , shuffle=False,
is_eval=True).batch(16):
                    preds = self.model.predict(img)
                    y_true = tf.argmax(mask['mask'], axis = -1)
                    y_pred = tf.argmax(preds, axis = -1)

                    for img_i, pred_i, mask_i in zip(img, y_pred, y_true):
                        volume_pred[slice_index] = np.squeeze(pred_i)
                        volume_mask[slice_index] = np.squeeze(mask_i)
                        volume_img[slice_index] = np.squeeze(img_i)
                        slice_index += 1

                metrics = np.zeros(shape=(len(meaningful_labels) - 1, 2))
                for i in range(1, len(meaningful_labels)):
                    metrics[i-1] = calculate_metric_percase(tf.cast(volume_pred == i, tf.float32),
                                                           tf.cast(volume_mask == i, tf.float32))
                logger.info('metrics for %s: %s', volume_id, metrics)
                logger.info('avg for %s: %s', volume_id, np.mean(metrics, axis=0))
                results[volume_id] = np.mean(metrics, axis=0)[0]
                total_metrics += metrics
            logger.info('per-class=%s (total %s)', total_metrics / count, count)
            logger.info('avg=%s', np.mean(total_metrics / count, axis=0))
            logger.info('result=%s', results)



    class BRaTSEvaluator(BaseEvaluator):

        def evaluate(self):
            count = 0
            self.dataset.prepare()

            slice_index = 0
            count += 1
            metrics_list = []
            ds = self.dataset.get_val_ds().batch(1).cache()
            for i, (img, mask) in enumerate(ds):
                print(f"loading img {i}")
            self.model.summary()
            for i, (img, mask) in enumerate(ds):
                start_time = time.time()
                preds = self.model(img, training=False)
                logger.info("evaluation time %s", time.time() - start_time)
                #display_image(img[...,  0],  255.  *  mask,  255.  *  preds,  step=5,  gif_only=True,
filename=f'animation-{i}.gif')
                start_time = time.time()
                mask = mask > 0.5
                preds = preds > 0.5
                metrics_list.append(calculate_metrics(tf.transpose(preds[0], [3, 0,1,2]), tf.transpose(mask[0],
[3,0,1,2]), i))
                logger.info("metric calculation time %s", time.time() - start_time)

            val_metrics = [item for sublist in metrics_list for item in sublist]
            df = pd.DataFrame(val_metrics)
            overlap = df.boxplot(METRICS[1:], by="label", return_type="axes")
            overlap_figure = overlap[0].get_figure()
            print(overlap_figure)
            #writer.add_figure("benchmark/overlap_measures", overlap_figure)
            haussdorf_figure = df.boxplot(METRICS[0], by="label").get_figure()
            print(haussdorf_figure)

            #writer.add_figure("benchmark/distance_measure", haussdorf_figure)
```

```
            grouped_df = df.groupby("label")[METRICS]
            summary = grouped_df.mean().to_dict()
            for metric, label_values in summary.items():
                for label, score in label_values.items():
                    logger.info("benchmark %s/%s: %s", metric, label, score)
            df.to_csv(('results.csv'), index=False)



    def calculate_metrics(preds, targets, patient, tta=False):
        pp = pprint.PrettyPrinter(indent=4)
        assert preds.shape == targets.shape, "Preds and targets do not have the same size"

        labels = ["ET", "TC", "WT"]

        metrics_list = []

        for i, label in enumerate(labels):
            metrics = dict(
                patient_id=patient,
                label=label,
                tta=tta,
            )

            if np.sum(targets[i]) == 0:
                print(f"{label} not present for {patient}")
                sens = np.nan
                dice = 1 if np.sum(preds[i]) == 0 else 0
                tn = np.sum(l_and(l_not(preds[i]), l_not(targets[i])))
                fp = np.sum(l_and(preds[i], l_not(targets[i])))
                spec = tn / (tn + fp)
                haussdorf_dist = np.nan

            else:
                preds_coords = np.argwhere(preds[i])
                targets_coords = np.argwhere(targets[i])
                haussdorf_dist = directed_hausdorff(preds_coords, targets_coords)[0]

                tp = np.sum(l_and(preds[i], targets[i]))
                tn = np.sum(l_and(l_not(preds[i]), l_not(targets[i])))
                fp = np.sum(l_and(preds[i], l_not(targets[i])))
                fn = np.sum(l_and(l_not(preds[i]), targets[i]))

                sens = tp / (tp + fn)
                spec = tn / (tn + fp)

                dice = 2 * tp / (2 * tp + fp + fn)

            metrics[HAUSSDORF] = haussdorf_dist
            metrics[DICE] = dice
            metrics[SENS] = sens
            metrics[SPEC] = spec
            pp.pprint(metrics)
            metrics_list.append(metrics)

        return metrics_list



    class CityScapesEvaluator(BaseEvaluator):
        class Prober:
            def __init__(self, no_show=False,labels=LABELS):
                n_classes = len(labels)
                self.ious = {i.id: tf.keras.metrics.IoU(n_classes, [i.id], name=f'iou-{i.name}') for i in
labels}
                self.dices = {i.id: BinaryDice(class_id=i.id, name=f'dice-{i.name}') for i in labels}
                self.labels_present = {}
                self.n_classes = n_classes
                self.confusion_matrix = np.zeros([n_classes, n_classes])
                self.labels = labels
                self.no_show = no_show

            def update(self, y, y_pred):
                yc = np.unique(np.reshape(y, -1))
                for class_id in yc:
                    if class_id in self.ious:
                        self.ious[class_id].update_state(y, y_pred)
                        self.dices[class_id].update_state(y, y_pred)
```

```
                        self.labels_present[class_id] = True
                for y_i, y_pi in zip(y, y_pred):
                    conf = tf.math.confusion_matrix(
                        y_i.reshape(-1),
                        y_pi.reshape(-1),
                        num_classes=self.n_classes
                    )
                    self.confusion_matrix += conf.numpy()

            def show(self):
                result_dict = {'iou': {}, 'dice': {},}
                plots, axis = plt.subplots(1,2,  figsize=(20, 5),)
                labels = [label for label in self.labels if not label.ignoreInEval  and
self.labels_present.get(label.id)]
                bar = axis[1].bar([label.name for label in labels],
                                  [self.ious[label.id].result().numpy() for label in labels],
                                  color=[[l.color[0]/255, l.color[1]/255,l.color[2]/255] for l in labels])
                axis[1].set_xticks(axis[1].get_xticks())
                axis[1].set_xticklabels(axis[1].get_xticklabels(), rotation=90)
                axis[1].bar_label(bar,  fmt='%.2g')
                label_names = [l.name for l in labels]

                for l in self.labels:
                    if not l.ignoreInEval and self.labels_present.get(l.id):
                        result_dict['iou'][l.name] =  self.ious[l.id].result().numpy()
                        result_dict['dice'][l.name] =  self.dices[l.id].result().numpy()
                        if not self.no_show:
                            logger.info('label: %s, IoU: %s', l.name, self.ious[l.id].result().numpy())
                if not self.no_show:
                    logger.info('mIoU: %s', np.mean([self.ious[l.id].result().numpy() for l in labels]))
                result_dict['mIoU'] = np.mean([self.ious[l.id].result().numpy() for l in labels])
                if not self.no_show:
                    axis[0].imshow(self.confusion_matrix, vmin=0, extent=[0, len(label_names), 0,
len(label_names)])
                    axis[0].set_xticks(range(len(label_names)), label_names, rotation=90)
                    axis[0].set_yticks(range(len(label_names)), label_names)
                    plt.show()
                return result_dict

        def evaluate(self):
            count = 0
            self.dataset.prepare()
            meaningful_labels = [label for label in self.dataset.labels]
            prober = self.Prober(labels=meaningful_labels)
            for img, mask in self.dataset.get_val_ds().batch(16):
                preds = self.model.predict(img)
                prober.update(tf.argmax(mask, axis=-1).numpy(), tf.argmax(preds, axis=-1).numpy())
            data = prober.show()
            data['model_name'] = self.model_name
            return data
```

Допоміжний клас, що дозволяє отримати проміжні карти ознак

```
    class ModelLayerExplorer:
        def __init__(self, model_name, layer_names=None):
            self.layer_names = layer_names if layer_names else None
            self.model_name = model_name

        def patch_model(self):
            model = load_model(self.model_name)
            outputs = []
            for layer_name in self.layer_names:
                layer = model.get_layer(layer_name)
                outputs.append(layer.output)
            new_model = CustomModel(inputs=model.inputs,
                                    outputs={
                                        'intermediate': outputs,
                                        'mask': model.output
                                    })
            return new_model
```

Лістинг А.14 - Код для проведення тестів швидкодії та використання пам’яті

```
def random_str(N):
    return ''.join(random.choices(string.ascii_uppercase + string.digits, k=N))


def benchmark_model(model_params):
    model = get_unet_model(**model_params)
    rand_name = random_str(5)

    saved_model_dir = os.path.join('content', rand_name)
    os.makedirs(saved_model_dir)
    model.save(saved_model_dir)
    model.summary()
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
    converter.target_spec.supported_ops = [
        tf.lite.OpsSet.TFLITE_BUILTINS,
        tf.lite.OpsSet.SELECT_TF_OPS
    ]
    tflite_model = converter.convert()
    model_file = f'model_{rand_name}.tflite'
    # Save the model.
    with open(model_file, 'wb') as f:
        f.write(tflite_model)
    logger.info("Running benchmark for parameters %s", model_params)
    cmd = f'./linux_x86-64_benchmark_model_plus_flex --graph={model_file} '\
                                        '--input_layer=input --input_layer_shape=1,256,256,1 --
warmup_min_secs=5'
    logger.info("Benchmark command: %s", cmd)

    try:
        res = subprocess.check_output(cmd, shell=True)
    except Exception as e:
        logger.error('Exception %s', e)
        res = e.output
    logger.info('%s', res.decode('ascii'))


modifications_list = [
    [],
    [UnetModifications.ATTENTION],
    [UnetModifications.ATTENTION, UnetModifications.DESSCO2],
    [UnetModifications.ATTENTION, UnetModifications.DESSCO],

    [UnetModifications.DESSCO2],
    [UnetModifications.DESSCO2, UnetModifications.RESIDUAL],
    [UnetModifications.DESSCO],
    [UnetModifications.DESSCO_2C],
]


for mods in modifications_list:
    benchmark_model(dict(channels=1, num_classes=5, **mods))
```

ДОДАТОК Б.
СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові праці в яких опубліковано основні наукові результати дисертації:

1. **Statkevych, R.,** Gordienko, Y., Stirenko, S. (2022). Improving U-Net Kidney Glomerulus Segmentation with Fine-Tuning, Dataset Randomization and Augmentations. In: Advances in Computer Science for Engineering and Education. Lecture Notes on Data Engineering and Communications Technologies, vol 134. *ISSN 2367-4520 (electronic)* | Springer, Cham. https://doi.org/10.1007/978-3-031-04812-8_42, - Scopus, Q3
2. **Statkevych, R.,** Gordienko, Y., Stirenko, S. (2023). Expansion Rate Parametrization and K-Fold Based Inference with U-Net Neural Networks for Multiclass Medical Image Segmentation. In: Artificial Intelligence and Soft Computing. Lecture Notes in Computer Science(), vol 14125. *ISSN 0302-9743*, Springer, Cham. https://doi.org/10.1007/978-3-031-42505-9_22, - Scopus, Q3
3. **Statkevych R**, Stirenko S, Gordienko Y. Human kidney tissue image segmentation by U-Net models. *ISSN 2473-2001 (Online)* | *IEEE Xplore digital library*. https://doi.org/10.1109/EUROCON52738.2021.9535599, - Scopus
4. **Statkevych R**, Gordienko Y, Stirenko S. Improving Pedestrian Detection Methods by Architecture and Hyperparameter Modification of Deep Neural Networks. In Advances in Artificial Systems for Logistics Engineering 2021 (pp. 44-53). *ISSN 2367-4512*, Springer International Publishing. https://doi.org/10.1007/978-3-030-80475-6_5, - Scopus, Q3