МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

Магістерська робота

на тему:

Програмно-алгоритмічне забезпечення для оброблення медичних зображень з використанням фрактальних фізико-інформованих нейронних мереж

Виконав: студент групи ПМІМ-22с
спеціальності
122 — комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Манохін Д.А.
(прізвище та ініціали)

Керівник (підпис) Соколовський Я.І.
(прізвище та ініціали)

Рецензент (підпис) Кособуцький П.С.
(прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*
Кафедра *інформаційних систем*
Спеціальність *122 — комп'ютерні науки*
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри *проф. Шинкаренко Г.А*
"*5*" *вересня* **20***22* **року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Манохіну Денису Андрійовичу
(прізвище, ім'я, по батькові)

1 .Тема роботи *Програмно-алгоритмічне забезпечення для оброблення медичних зображень з використанням фрактальних фізико-інформованих нейронних мереж*

керівник роботи *Соколовський Ярослав Іванович, доктор технічних наук, професор,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від "*13*" *вересня* 20*22* року №*15*

2. Строк подання студентом роботи *12.12.2022 р.*

3. Вихідні дані до роботи *Література та інтернет-ресурси за тематикою роботи, датасет КТ-сканів головного мозку з позначеними крововиливами*

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)
1. Аналіз сучасного стану проблеми
2. Інформаційне та алгоритмічне забезпечення
3. Програмна реалізація
4. Аналіз отриманих результатів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схеми, діаграми, презентація дипломної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *05.09.2022 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	*Аналіз сучасного стану проблеми*	*Вересень*	*виконано*
2.	*Розробка алгоритмів*	*Жовтень*	*виконано*
3.	*Програмна реалізація*	*Листопад*	*виконано*
4.	*Аналіз результатів*	*Листопад*	*виконано*
5.	*Оформлення роботи і подання до захисту*	*Грудень*	*виконано*

Студент _____ *Манохін Д.А.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *проф. Соколовський Я.І.*
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Глибинне навчання	підгалузь машинного навчання, що спеціалізується на глибоких штучних нейронних мережах.
ЗНМ	згорткова нейронна мережа.
Машинне навчання	галузь штучного інтелекту, яка часто застосовує статистичні прийоми для надання комп’ютерам здатності “навчатися” (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоб бути програмованими явно.
Карта ознак	від англійського feature map, результат застосування одного окремого фільтра згорткової нейронної мережі.
КТ	комп’ютерна томографія.
КТ-скан	зображення отримане в результаті комп’ютерної томографії.
Розмір порції	від batch size, кількість даних що подаються на вхід нейронної мережі за один крок тренування чи валідації.
ШНМ	штучна нейронна мережа.
CPU	central processing unit (центральний процесор).
FCD	оператор для покращення текстури зображення на основі дробової похідної Рісса.
GPU	graphics processing unit (графічний процесор).
YuiFeiPU	оператори для покращення текстури зображення на основі дробових похідних Рімана-Ліувілля Грюнвальда-Лєтнікова.
SNR	signal to noise ratio (співвідношення сигналу до шуму).

ПОСТАНОВКА ЗАДАЧІ

1. Проаналізувати наявні розробки в сфері застосування штучних нейронних мереж (ШНМ) до автоматизації аналізу медичних зображень, зокрема сегментації внутрішньочерепних кровотеч за даними комп’ютерної томографії (КТ).

2. Підібрати набір КТ-сканів пацієнтів з внутрішньочерепним крововиливом для тренування ШНМ. Основною вимогою є те, щоб окрім самих сканів датасет містив також області кровотеч правильно виділені професіоналами.

3. Серед архітектур ШНМ, розглянутих в першому пункті, обрати таку, яка найбільше підходить для вирішення задачі. Розробити розпаралелений алгоритм навчання ШНМ з використанням технологій CUDA. Підібрати оптимальні параметри, щоб максимізувати точність результатів. Проаналізувати результати.

4. Дослідити алгоритми покращення якості зображення на основі методів фрактального аналізу. Реалізувати та застосувати до обраного датасету такий алгоритм. Перевірити як вплине застосування цього алгоритму на точність сегментації внутрішньочерепних крововиливів.

5. Модифікувати обраний алгоритм таким чином, щоб для апроксимації диференціальних операторів дробового порядку використовувалась фрактальна фізико-інформована нейронна мережа.

ВСТУП

Актуальність. Крововилив внаслідок гострої черепно-мозкової травми (ЧМТ) є третьою за поширеністю причиною смерті. А серед пацієнтів молодого віку, навіть випереджає захворювання серцево-судинної системи та онкологію, посідаючи перше місце. Окрім цього існує високий ризик втрати дієздатності особи, що перенесла внутрішньочерепний крововилив [1].

Для успішного лікування надзвичайно важливим є вчасне та точне виявлення крововиливу. Для цього найчастіше використовується КТ. Аналізом КТ-сканів займаються лікарі-радіологи. Оскільки аналіз проводиться вручну, він може займати багато дорогоцінного для збереження життя пацієнта часу, а правильність діагнозу повністю залежить від кваліфікації лікаря. Автоматизувати цей процес з достатнім рівнем точності, щоб повністю довірити йому життя пацієнта вдасться не скоро. Тому наразі метою є розробка алгоритмів, достатньо надійних для того, щоб допомогти радіологу у прийнятті рішення, вказавши підозрілі ділянки на зображенні.

У попередній роботі [2] розглядалася задача сегментації внутрішньочерепних кровотеч за даними комп’ютерної томографії (КТ) за допомогою штучної нейронної мережі (ШНМ). Розроблена модель показала непогані результати у вирішенні поставленої задачі. Однак все ще є можливість для її вдосконалення.

Медичні зображення зазвичай мають малу низьку роздільну здатність, високий рівень шуму та багато текстур. При цьому, вони несуть величезні об’єми інформації. Наявність шуму значно ускладнює інтерпретацію таких зображень. Тому покращення текстури та усунення шумів є ключовим етапом аналізу медичних зображень.

Метою роботи є розроблення програмно-алгоритмічного забезпечення для покращення оброблення текстури медичних зображень з використанням методів фрактального аналізу.

Для досягнення поставленої мети у роботі виконано такі завдання:

1. Проаналізувати наявні розробки в сфері застосування штучних нейронних мереж (ШНМ) до автоматизації аналізу медичних зображень, зокрема сегментації внутрішньочерепних кровотеч за даними комп'ютерної томографії (КТ).

2. Підібрати набір КТ-сканів пацієнтів з внутрішньочерепним крововиливом для тренування ШНМ. Основною вимогою є те, щоб окрім самих сканів датасет містив також області кровотеч правильно виділені професіоналами.

3. Серед архітектур ШНМ, розглянутих в першому пункті, обрати таку, яка найбільше підходить для вирішення задачі. Розробити розпаралелений алгоритм навчання ШНМ з використанням технологій CUDA. Підібрати оптимальні параметри, щоб максимізувати точність результатів. Проаналізувати результати.

4. Дослідити алгоритми покращення якості зображення на основі методів фрактального аналізу. Реалізувати та застосувати до обраного датасету такий алгоритм. Перевірити як вплине застосування цього алгоритму на точність сегментації внутрішньочерепних крововиливів.

5. Модифікувати обраний алгоритм таким чином, щоб для апроксимації диференціальних операторів дробового порядку використовувалась фрактальна фізико-інформована нейронна мережа.

Об'єкт дослідження — оброблення медичних зображень для сегментації внутрішньочерепних крововиливів.

Предмет дослідження — програмне та алгоритмічне забезпечення для покращення якості зображень за допомогою фрактальних фізико-інформованих нейронних мереж та навчання ШНМ для сегментації медичних зображень.

Наукова новизна полягає в застосуванні алгоритму покращення якості зображень на основі дробових похідних для покращення точності сегментації внутрішньочерепних крововиливів та адаптації цього алгоритму до обчислення дробових операторів за допомогою фрактальних фізико-інформованих нейронних мереж. А також в дослідженні розпаралелення навчання ШНМ для сегментації медичних зображень з використанням технологій CUDA.

Практичне значення роботи полягає в тому, що розроблений програмно-алгоритмічний комплекс може бути застосований для покращення точності сегментації внутрішньочерепних кровотеч за даними комп'ютерної томографії, що допоможе лікарям ідентифікувати проблемні ділянки.

Апробація.

Манохін Д. Програмно алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж з використанням технологій CUDA / Д. Манохін // Міжнародна студентська наукова конференція з питань прикладної математики та комп'ютерних наук (МСНКПМК-2021), 6-7 травня 2021 р. – Львів:2021. – С. 21-25. Режим доступу: https://ami.lnu.edu.ua/wp-content/uploads/2021/05/Ministerstvo-osvity-i-nauky-Ukrainy.docx

Sokolovskyy Y., Development of software and algorithms of parallel learning of artificial neural networks using CUDA technologies / Sokolovskyy Y., Manokhin D., Kaplunsky Y., Mokrytska O. // *Technology Audit and Production Reserves*, *5*(2(61). – 2021. – P.21-25. Available from: https://doi.org/10.15587/2706-5448.2021.239784

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПРОБЛЕМИ

1.1 Огляд предметної області

Хоча, як було сказано раніше, застосування ШНМ не вимагає досконалих знань області застосування, проте, оскільки правильна підготовка даних є критичною для успішного навчання моделі, базове розуміння природи даних стане в нагоді. Особливо це важливо саме в роботі з зображеннями медичного характеру, оскільки вони кардинально відрізняються від звичних нам фотографій. Тому розглянемо детальніше роботу комп'ютерного томографа.

Комп'ютерна томографія була винайдена сером Годфрі Гаунсфілдом, за що в 1979 році його було нагороджено Нобелівською премією. Разом з ним було нагороджено іншого вченого, Алана Кормака, який незалежно від Гаунсфілда математично описав теоретичний прототип КТ сканера [3].

КТ дозволяє зробити знімок потрібної ділянки тіла у розрізі. Комбінуючи багато таких зображень, можна отримати тривимірне зображення. І хоч КТ є історично першим методом об'ємного сканування людського організму, він залишається одним з найпоширеніших і сьогодні. Оскільки порівняно з сучаснішими методами, такими як магнітно-резонансна томографія (МРТ), КТ є дешевшою та займає значно менше часу. Тому при підозрі на внутрішньочерепний крововилив пацієнту переважно призначають саме КТ.

Основними компонентами комп'ютерного томографа є джерело та детектор рентгенівського випромінювання розташовані на протилежних сторонах рамки, що обертається. Для того щоб отримати зображення тіла у розрізі, пацієнт поміщається між випромінювачем та детектором, і рамка обертається, вимірюючи коефіцієнти поглинання рентгенівських променів при проходженні тіла пацієнта під різними кутами (рис. 1.1). Коефіцієнт поглинання позначають μ, він показує наскільки послабиться пучок променів після проходження через певне середовище (в нашому випадку тканини тіла пацієнта) [4].

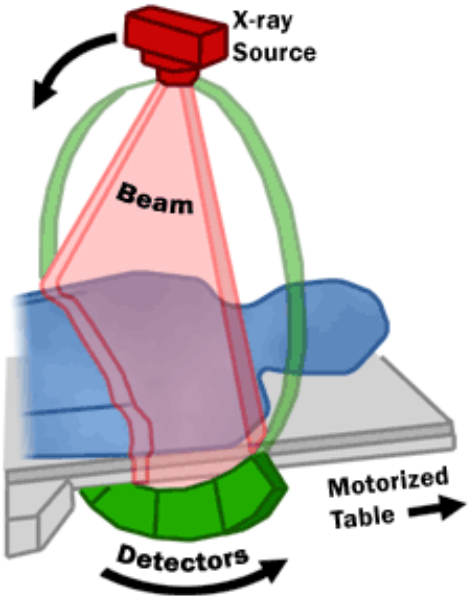


Рисунок 1.1. Схема роботи комп'ютерного томографа [5]

Маючи виміри для різних кутів надходження рентгенівських променів, комп’ютерна програма може відновити значення коефіцієнтів поглинання в кожній точці відсканованого перерізу та представити результат у вигляді зображення в рівнях сірого, де більшим коефіцієнтам поглинання відповідають світліші відтінки, а меншим — темніші. Реконструкція зображень з так званих “сирих” даних з томографу є окремою галуззю в якій існує багато алгоритмів зі своїми перевагами та недоліками для різних задач, однак вони виходять за межі теми даної роботи.

Перед тим як реконструйоване зображення буде готове до аналізу радіологом, значення кожного пікселя переводять до одиниць Гаунсфілда. Одиниця Гаунсфілда (позначають HU від Hounsfield Unit) є лінійним перетворенням коефіцієнту поглинання μ

$$HU = 1000 \times \frac{\mu - \mu_B}{\mu_B - \mu_П} \tag{1.1}$$

де μ_B — коефіцієнт поглинання води, $\mu_П$ — коефіцієнт поглинання повітря.

Шкала Гаунсфілда встановлює відповідність між цими величинами та матеріалами. Так мінімальне значення -1000 HU має повітря за нормальних умов (температура 0°C та тиск 10^5 Па), дистильованій воді відповідає 0 HU, для деяких кісток значення може сягати 2000 HU, а металам відповідають значення більші 3000 HU [6].

Зазвичай, для збереження КТ-сканів використовуються 12-бітні зображення, що можуть зберігати значення від -1024 до 3071, оскільки тканини людського тіла не виходять за межі цього діапазону. Однак, пікселі стандартних рідкокристалічних дисплеїв можуть відображати лише 256 рівнів сірого. При стисненні діапазону дуже погіршується контрастність, критично необхідна для коректного аналізу зображення. І навіть якщо використовувати спеціальні дисплеї, що можуть відобразити 4096 рівнів сірого, людське око не зможе досконало розрізнити таку кількість відтінків одного кольору. Тому, для аналізу КТ-сканів, з зображення виділяють вікно вужчого діапазону, який покриває лише частину шкали Гаунсфілда, яка нас цікавить. Вікно характеризується положенням центру на шкалі Гаунсфілда (WL від window level) та шириною (WW від window width). Мінімальне та максимальне значення (I_{min} та I_{max} відповідно), що потрапляє в задане вікно можна знайти за наступними формулами:

$$I_{min} = WL - (WW \div 2) \tag{1.2}$$

$$I_{max} = WL + (WW \div 2) \tag{1.3}$$

На отриманому зображенні всі значення менші за I_{min} будуть чорними, а більші за I_{max} — білими (рис. 1.2).

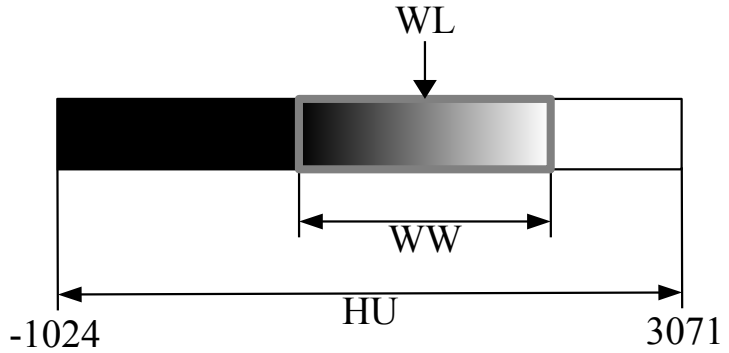


Рисунок 1.2. Схематичне зображення вікна на шкалі Гаунсфілда

Змінюючи положення центра вікна ми можемо концентруватись на дослідженні тканин, що нас цікавлять. При цьому чим ширше вікно, тим більше типів тканин буде охоплено, однак ціною цьому буде нижчий контраст, і навпаки, вужче вікно дозволить отримати контрастне зображення, але відкине досить багато інформації, що не потрапила в його межі. Тому важливо обрати оптимальні

параметри вікна. Різницю між зображеннями, отриманими з різними параметрами вікна можна бачити на рис. 1.3.

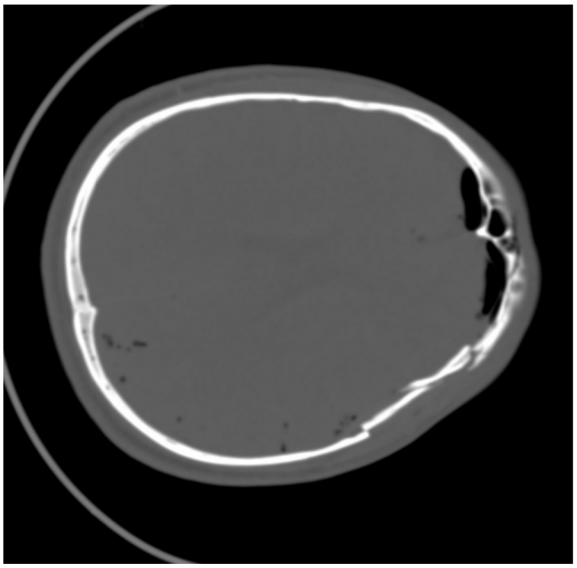
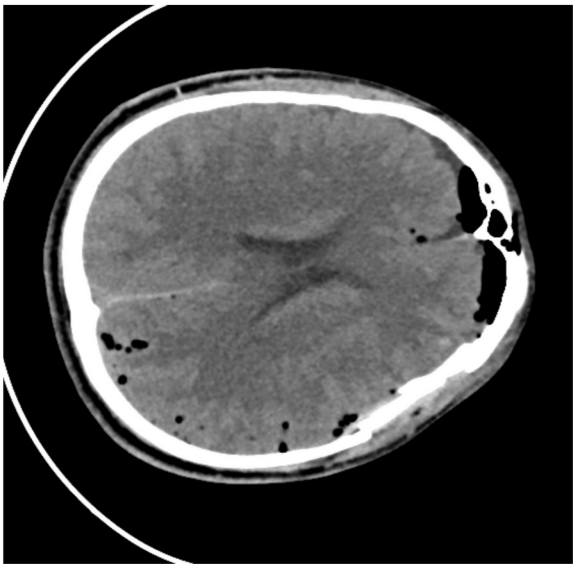
а)  б) 

Рисунок 1.3. КТ-скан головного мозку а) вікно (WL: 300, WW: 2000) добре виділяє кістки та дозволяє чітко побачити перелом справа, але сам мозок майже повністю представлений одним відтінком б) вікно (WL: 40, WW:120) навпаки дозволяє детальніше розглянути частини мозку, проте перелому тепер не видно, так як тканини біля кісток виходять за межі обраного вікна і тому зображаються так само як і самі кістки — білим кольором

1.2 Сегментація зображень за допомогою ШНМ

Сегментацією зображення називають процес локалізації об’єктів на зображенні. Сегментація зображень є однією з ключових задач комп’ютерного зору. Існує безліч алгоритмів сегментації різного рівня складності. Більшість з них зводяться до того, що кожен піксель відносять до відповідного класу на основі певних спільних для цього класу характеристик, як от яскравість, колір чи текстура. Найпростішим алгоритмом є порогування, метод при якому обирається певне значення (поріг) і всі пікселі з інтенсивностями більшими за поріг відносять до одного класу, а з меншими до іншого.

Однак часто, об’єкти які нас цікавлять на зображенні не так просто виділити, оскільки вони можуть бути дуже неоднорідними. Наприклад знайти людину на зображенні не вдасться просто за спільним кольором чи текстурою. Більше того, серед класичних алгоритмів комп’ютерного зору не існує загальних методів сегментації, які б однаково добре працювали у всіх випадках. Кожна

задача вимагає індивідуального підходу. Особливо гостро ця проблема постає при сегментації медичних зображень, оскільки від розробника вимагаються не тільки навички з області комп’ютерного бачення, але і доволі глибокі знання з медицини.

Допомогти вирішити ці проблеми можуть ШНМ. По-перше ті самі архітектури ШНМ можуть використовуватись для вирішення дуже широкого кола задач. А по-друге, не вимагають від розробника досконалого знання предметної області, а лише правильно підготованих даних. Тому в області сегментації медичних зображень, зараз поширеною практикою є наступний спосіб роботи. Професійні лікарі для великого набору даних вручну виділяють, області які потрібно навчитись сегментувати автоматично, а задачею розробника вже є тренування ШНМ за допомогою цих даних.

Історично ШНМ були застосовані до задачі класифікації зображень раніше, ніж до сегментації. Особливо добре себе показали згорткові нейронні мережі (ЗНМ). Одну з таких мереж, застосовану до задачі багатокласової класифікації було розглянуто в попередній роботі [7].

Враховуючи ефективність застосування ЗНМ до задач класифікації, для сегментації також було використано цей тип мереж. Зрозуміло, що конкретні архітектури відрізняються, як мінімум, через те, що для класифікації мережа повинна на виході повертати лише одне значення класу для всього зображення, а для сегментації — класи кожного пікселя вхідного зображення. Але принцип роботи також базується на операції згортки.

Одним з ранніх підходів, що дозволив перевершити по точності класичні методи сегментації була мережа, запропонована у 2012 році Чирешаном та ін. [8] Їхній підхід полягав в тому, що для кожного пікселя вхідного зображення береться певний окіл і класифікується класичною ЗНМ. Зрозуміло, що такий спосіб вимагав значних обчислювальних потужностей та був не надто ефективним за часом, проте продемонстрував чудові результати у вирішенні задачі та приніс перемогу своїм розробникам на конкурсі ISBI 2012 значно перевершивши інші алгоритми.

Ще кращі результати і при цьому за менший час були отримані завдяки іншому підходу — повністю згортковим мережам. Основною ідеєю цього підходу є використання лише згорткових шарів, без повністю зв’язаних, які зазвичай розташовувались перед виводом класичних ЗНМ.

Повністю згортковою мережею, розробленою спеціально для задач сегментації медичних зображень, є мережа U-Net запропонована Олафом Ронненбергером та ін. 2015 року [9]. Метою її створення, була необхідність в мережі, що може ефективно навчатись на невеликих наборах даних, оскільки в області медицини часто є проблеми з наявністю великих датасетів, через вимоги до конфіденційності даних такого характеру. І, можна сказати, з цією задачею автори справились, оскільки U-Net продемонструвала настільки хороші результати, що стала по суті класичною архітектурою для сегментації медичних зображень. Тому будемо використовувати її в цій роботі для сегментації внутрішньочерепних крововиливів. Детальніше архітектуру цієї мережі розглянуто в пункті 2.3.

1.3 Класифікація та сегментація внутрішньочерепних крововиливів за КТ-сканами з використанням ШНМ

Автоматична сегментація внутрішньочерепних крововиливів за даними КТ-сканів на сьогодні є недостатньо досліджена. Проблемою є те, що як і для багатьох задач обробки медичних зображень, до недавнього часу не існувало датасетів з промаркованими областями крововиливів у відкритому доступі. Причиною цьому є як складність підготовки такої інформації в правовому полі, оскільки зображення можуть містити персональні дані пацієнтів, так і те, що для задачі сегментації, професіонали повинні приділити багато часу ручному маркуванню проблемних областей на сканах.

Оскільки промаркувати лише саму наявність та тип крововиливу на зображенні легше ніж позначати його локацію, більш поширеними є набори даних для задачі класифікації КТ-сканів. Так популярним датасетом для валідації класифікатора є CQ500, набір, що містить 491 КТ-сканів голови для кожного з

яких позначено відповідний тип крововиливу чи його відсутність. Для нього було отримано дуже хороші результати з близько 95% точністю розпізнавання для кожного класу [10]. Також нещодавно на платформі Kaggle проводилось змагання від Північноамериканської спільноти радіологів (RSNA) для якого було оприлюднено датасет з понад 25000 зображень [11]. В ході нього учасниками було запропоновано багато високоточних моделей класифікації.

Як бачимо наявність великої кількості даних у відкритому доступі дозволила значно просунутись у вирішені задачі класифікації КТ-сканів головного мозку на предмет наявності того чи іншого типу крововиливу. А через відсутність стандартних датасетів для сегментації виникають проблеми у дослідженні цієї задачі. Як було показано Гссаєні та колегами в огляді сучасних методів сегментації внутрішньочерепних крововиливів [12], для вирішення цієї задачі досить успішно застосовуються як методи, що базуються на класичних алгоритмах комп'ютерного зору, так і методи на основі глибинного навчання. Однак, наявні методи тестувались на різних датасетах з обмеженим доступом, тому ми не можемо коректно порівняти ці моделі на одному наборі даних. Тому цими ж авторами було створено відкритий набір даних для тренування та валідації моделей сегментації внутрішньочерепних крововиливів [13]. Його було опубліковано на платформі PhysioNet [14].

Останній датасет і буде розглядатись в цій роботі. Детальніше його структура описана в розділі про інформаційне забезпечення, пункт 2.1.

1.4 Останні результати для обраного датасету

Окрім створення датасету у згаданій роботі [12] автори також описали процес тренування мережі U-Net на цьому датасеті та навели оцінки точності виведення натренованої моделі у різних метриках.

Для тренування використовувались окремі шари виділені з об'ємних сканів з вікном (WL: 40, WW: 120), при чому використовувались як зразки пацієнтів з крововиливом так і без. Оскільки зразків без крововиливу набагато більше, використовувалась лише певна вибрана серед них випадковим чином частина.

Також оскільки даних не є дужа багато, а через широкий діапазон віку, розміри черепу є досить різними, під час тренування до даних застосовувались випадкові афінні перетворення для покращення генералізації моделі.

Було проведено два експерименти, як з повними зображеннями, так і з зображеннями розбитими на блоки 160x160. Більше уваги приділялось саме другому методу.

Тренування проводилось протягом 150 епох, використовуючи GPU Nvidia GeForce RTX2080 з 11 Гб пам’яті. На це було витрачено близько 5 годин.

Для навчання моделі використовувався оптимізатор Adam з коефіцієнтом навчання 10^{-5}. В якості функції втрат була використана перехресна ентропія. Тренування відбувалось з розміром порції 32.

Результати тестування навченої мережі можна бачити в таблиці 1.1. Коефіцієнти Дайса та Жаккара мають невеликі значення, але співмірні з іншими експериментами, де мережа тренувалась на доволі невеликому наборі даних. Великим плюсом є велике значення чутливості, оскільки в області медицини, краще, щоб модель видавала “хибну тривогу” ніж, щоб пропустила дійсно проблемну ділянку. Тому можна сказати, що в цій роботі автори показали, можливість створення ШНМ для асистування радіологам та надали базову модель, на яку можуть опиратись інші дослідники при розробці покращених алгоритмів.

Таблиця 1.1. Результати тестування натренованої ШНМ [12]

	Коефіцієнт Жаккара	**Коефіцієнт Дайса**	**Чутливість (%)**	**Специфічність (%)**
Min	0.00	0.00	50	0
Max	0.528	0.677	100	100
STD	0.163	0.211	9.9	29.9
Average	0.218	0.315	97.28	50.4

1.5 Покращення якості зображень на основі похідних дробового порядку

Фрактальний аналіз має багато застосувань в математиці, фізиці та інженерії. Цей розділ математичного аналізу займається дослідженням

диференціальних операторів дробового порядку, що є узагальненням диференціювання цілочисельного порядку. Похідні дробового порядку є чудовим інструментом для опису властивостей багатьох матеріалів та процесів.

Існує декілька способів визначення похідної дробового порядку. Доволі поширеними є узагальнені похідні Рімана-Ліувілля, Грюнвальда-Лєтнікова та Рісса.

Похідна Рімана-Ліувілля дробового порядку v ($0 \le v < n$) задається формулою:

$$D_{R-L}^{v} = \frac{d^{v}}{[d(x-a)]^{v}} s(x) = \frac{d^{n}}{dx^{n}} \frac{d^{v-n}}{[d(x-a)]^{v-n}} s(x) = \sum_{k=0}^{n-1} \frac{(x-a)^{k-v} s^{(k)}(a)}{\Gamma(k-v+1)} + \frac{1}{\Gamma(n-v)} \int_{a}^{x} \frac{s^{(n)}(\xi)}{(x-\xi)^{v-n+1}} d\xi \tag{1.4}$$

де n найменше ціле число більше ніж v.

Похідна Грюнвальда-Лєтнікова порядку v:

$$D_{G-L}^{v} s(x) = \frac{d^{v}}{[d(x-a)]^{v}} s(x) = \lim_{N \to \infty} \left\{ \left(\frac{x-a}{N} \right)^{-v} \sum_{k=0}^{N-1} \frac{\Gamma(k-v)}{\Gamma(k+1)} \times s\left(x - k\left(\frac{x-a}{N} \right) \right) \right\} \tag{1.5}$$

Дробова похідна Рісса порядку v для нескінченного інтервалу $-\infty < x < +\infty$ задається формулою:

$$D_{R}^{v} s(x) = \frac{\partial^{v} s(x)}{\partial |x|^{v}} = -\frac{1}{2\cos\left(\frac{\pi v}{2}\right)} \left(\frac{\partial^{v}}{\partial x^{v}} + \frac{\partial^{v}}{\partial (-x)^{v}} \right) s(x) \tag{1.6}$$

де $v \neq 1$, $p - 1 < v \le p \le 2$ для $p \in \mathbb{N}$ та

$$\frac{\partial^{v} s(x)}{\partial x^{v}} = \frac{1}{\Gamma(p-v)} \frac{\partial^{p}}{\partial x^{p}} \int_{-\infty}^{x} \frac{s(\xi) d\xi}{(x-\xi)^{v+1-p}} \tag{1.7}$$

$$\frac{\partial^{v} s(x)}{\partial (-x)^{v}} = \frac{(-1)^{p}}{\Gamma(v-p)} \frac{\partial^{p}}{\partial x^{p}} \int_{x}^{+\infty} \frac{s(\xi) d\xi}{(\xi-x)^{v+1-p}} \tag{1.8}$$

Комп’ютерний зір — це одна зі сфер, де останнім часом активно починають застосовуватись ці оператори. А саме досить хороші результати продемонстрували методи покращення текстури зображення на основі дробових похідних.

Більшість популярних методів покращення зображень базуються на похідних цілочисельних порядків. Вони обчислюються для зображення за допомогою масок (також відомих як фільтри, оператори або шаблони) Собела, Робертса або Превіта. Однак такі оператори мають деякі недоліки, найбільший з яких висока чутливість до шуму [15].

Оператори цілочисельних порядків в області комп'ютерного зору також часто називають високочастотними фільтрами, оскільки вони посилюють інтенсивність високочастотних компонент зображення, при цьому послаблюючи низькі частоти.

Частоти зображення визначають зміну рівнів сірого відносно зміни відстані. До прикладу краї та шум є високочастотними компонентами, оскільки вони характеризуються значною зміною інтенсивності пікселів протягом короткої відстані. І навпаки текстури та фон є прикладами низькочастотних компонент, оскільки вони характеризуються незначними змінами рівнів сірого [16].

Оскільки шум характеризується високою частотою, застосування високочастотного фільтру до зображення, окрім підсилення елементів що нас цікавлять, значно підсилить і рівень шуму. Обійти цей недолік дозволяють запропоновані 2010 року Юі-Феєм Пу та ін. [17] маски на основі похідних дробового порядку (також відомі як оператори YuiFeiPU). В статті автори пропонують кілька варіантів побудови таких масок на основі похідних Рімана-Ліувілля (1.4) та Грюнвальда-Лєтнікова (1.5).

Така маска зберігає низькі частоти в гладких областях зображення, а також зберігає високі частоти там де є значні зміни інтенсивності, при цьому підсилюючи деталі текстури в областях де зміни інтенсивності менш помітні.

Маска запропонована Юі-Феєм складається з 8 частин. Для лівосторонніх похідних вздовж осі x, осі y, та двох діагоналей і відповідних правосторонніх похідних. Це забезпечує незалежність отриманого оператора від повороту.

Однак, як було показано в роботі [15], використання похідної Рісса (1.7) в запропонованих масках дозволяє ще більше підвищити стійкість до шуму.

Модифікований оператор автори назвали FCD. В таблиці 1.2 наведено співвідношення сигналу до шуму (Signal to Noise Ratio — SNR) отримані для різних операторів.

Таблиця 1.2. SNR для різних операторів [15]

Оператор	**SNR**
Sobel	3.5
Laplacian	3.3
YiFeiPU	8.3
FCD	10.2

Окрім цього, дослідження стійкості до шуму показали, що найкраще значення SNR отримано для оператора FCD з мінімальним розміром маски (таблиця 1.3).

Таблиця 1.3. SNR для різних розмірів маски оператора FCD [15]

Розмір маски	**SNR**
3x3	13.3
5x5	10.2
7x7	8.8
9x9	8.0
11x11	7.5

Враховуючи ці результати, в даній роботі буде застосовано саме оператор FCD, тому в пункті 2.4 наведено детальний опис цього алгоритму.

1.6 Висновки

- В цьому розділі було розглянуто специфіку роботи з КТ-сканами та підібрано датасет з КТ-сканами для вирішення прикладної задачі сегментації внутрішньочерепних крововиливів.
- Також було проаналізовано стан справ в області сегментації зображень за допомогою ШНМ і як результат обрано мережу U-Net для застосування до вирішення нашої задачі.

• Окрім цього в результаті аналізу існуючих методів покращення якості зображень за допомогою похідних дробового порядку було обрано метод на основі похідної Рісса для подальших досліджень.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Датасет для сегментації внутрішньочерепних крововиливів

Дані для створення датасету були отримані між лютим та серпнем 2018 року в госпіталі Аль Гілла, в Іраку. Для сканування використовувався комп'ютерний томограф Siemens / SOMATOM Definition AS. Всього було досліджено 82 пацієнтів, серед них 46 чоловіків та 36 жінок, з середнім віком 27,8±19,5 років. Загальний діапазон віку від 1 дня до 72 років. Пацієнтів молодших 18 років — 27, а старших — 55. Внутрішньочерепна кровотеча виявлена у 36 осіб, а перелом черепа у 22 [12]. Якщо розглядати окремі шари, отримаємо 318 зображень з наявною кровотечею та 2491 без.

Зображення представлені у форматі NIfTI, з розширенням 512x512 та змінною кількістю шарів для кожного пацієнта (зазвичай близько 30). Окрім цього, кожному зображенню відповідає маска, де в разі наявності крововиливу, відповідні пікселі виділені білим кольором. Також до даних надається csv файл в якому для кожного шару кожного зображення вказано наявний тип крововиливу чи його відсутність.

2.2 Алгоритмічне забезпечення навчання ШНМ

Алгоритмічне забезпечення навчання ШНМ розглянемо на прикладі однієї з найпростіших моделей ШНМ — багатошарового перцептрону (рис. 2.1).

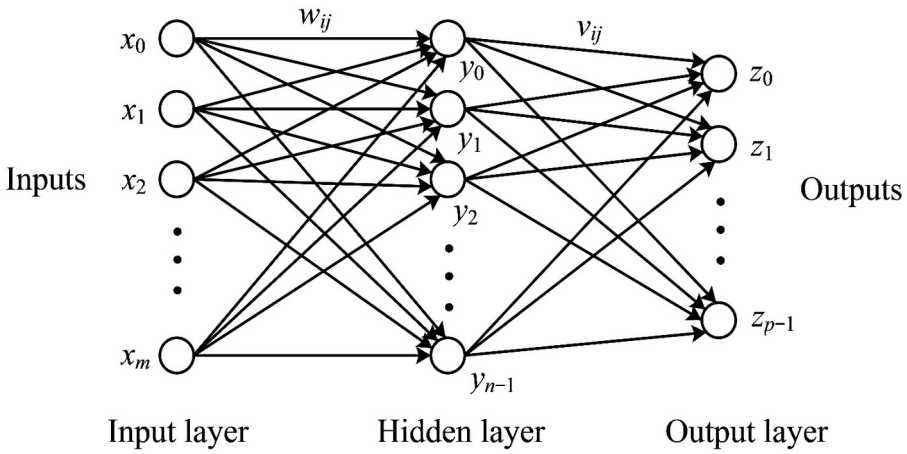


Рисунок 2.1. Багатошаровий перцептрон з одним прихованим шаром [18]

Кожен штучний нейрон виглядає наступним чином (рис. 2.2):

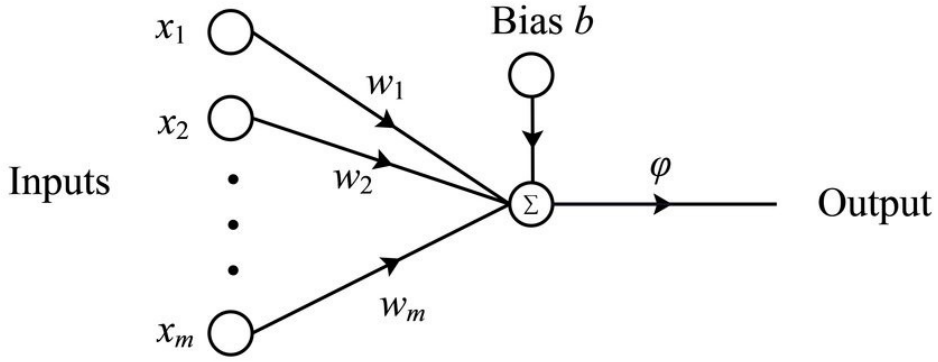


Рисунок 2.2. Штучний нейрон [18]

Математично його можна описати такою функцією:

$$\sigma(\bar{x}) = \varphi(\bar{x} \cdot \bar{w} + b) \tag{2.1}$$

де $\bar{x} = (x_1, x_2, \dots x_m)^T$ $x_i \in \mathbb{R}$ — вектор з вхідними даними довжини m, $\bar{w} = (w_1, w_2, \dots w_m)^T$ $w_i \in \mathbb{R}$ — ваги цього нейрону, ($i = \overline{1,m}$), $b \in \mathbb{R}$ — деяке число, відоме також як зсув.

Так само можна записати шар ШНМ:

$$f(\bar{x}) = \varphi(W^T \cdot \bar{x} + \bar{b}) \tag{2.2}$$

де W — матриця, стовпці якої — це вектори ваг кожного нейрону, а $\bar{b}$ — вектор зсувів кожного нейрону.

Тоді саму ШНМ можна представити як композицію функцій кожного нейрону. Наприклад двошаровий перцептрон з рисунку 2.1 може бути записаний як $f(\bar{x}) = f^{(2)}(f^{(1)}(\bar{x}))$, де $f^{(1)}$ та $f^{(2)}$ — функції першого та другого шарів відповідно.

Активаційну функцію обирають залежно від задачі. В перших ШНМ це була порогова функція, що при перевищенні певного значення рівна 1, а інакше — 0. Пізніше почали використовувати нелінійні функції такі як сигмоїда чи тангенс гіперболічний, оскільки вони дозволили вирішувати складніші лінійно нероздільні задачі. Також дуже добре себе показала кусково лінійна функція ReLU. Приклади деяких активаційних функцій можна побачити на наступній сторінці (рис. 2.3).

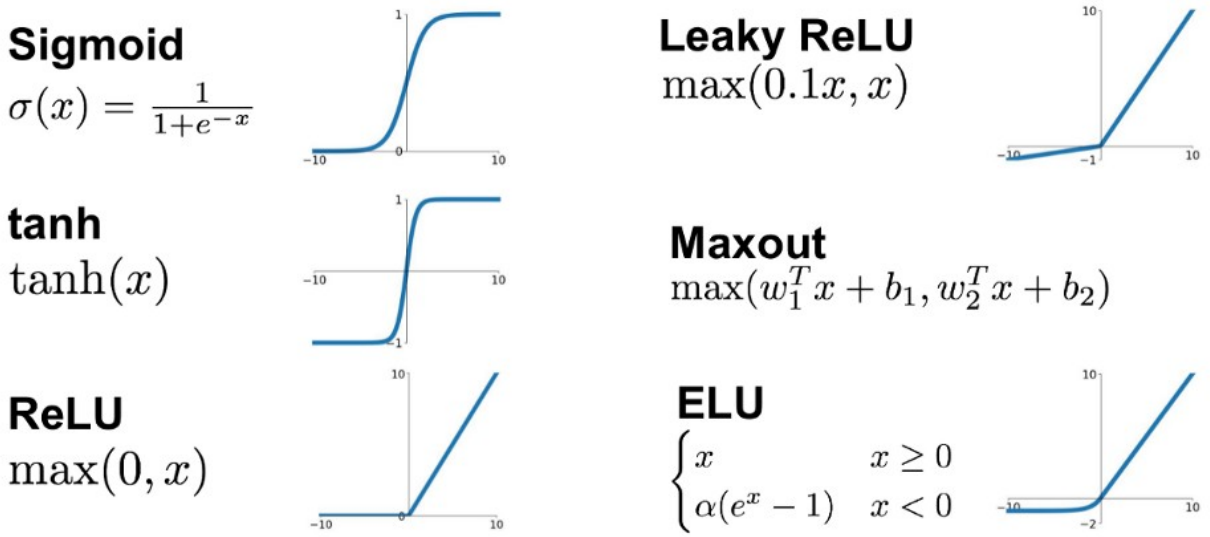


Рисунок 2.3. Активаційні функції [19]

Ще одним важливим компонентом ШНМ є функція втрат. Вона дає змогу оцінити, наскільки результат, що дає і нейронна мережа відрізняється від очікуваного. Наприклад це може бути середнє квадратичне відхилення:

$$J(\Theta)=\frac{1}{n}\sum_{i=0}^{n}\left(y_i-y_i^{(\Theta)}\right) \tag{2.3}$$

де Θ — параметри ШНМ (ваги та зміщення), n — обсяг вибірки, y_i — очікуваний результат i-го елемента вибірки, $y_i^{(\Theta)}$ — результат, що дає ШНМ для i-го елемента вибірки.

Тренування мережі відбувається за допомогою мінімізації функції втрат. Для цього, методом градієнтного спуску, оновлюють параметри Θ:

$$\Theta_{t+1}=\Theta_t-\alpha\,\nabla_\Theta J(\Theta) \tag{2.4}$$

де α — коефіцієнт навчання, ∇_Θ — градієнт функції втрат.

Наведений вище опис базується на книзі Яна Ґудфелоу [20].

Також варто згадати про теорему Цибенка, також відому як теорема про універсальний апроксиматор функцій, відповідно до якої багатошаровий перцептрон з одним прихованим шаром є універсальним апроксиматором функцій. Тобто така ШНМ може вивчити довільну функцію з будь-яким заданим рівнем точності, у випадку якщо немає обмеження на кількість нейронів в прихованих шарах [21].

2.3 Архітектура мережі U-Net

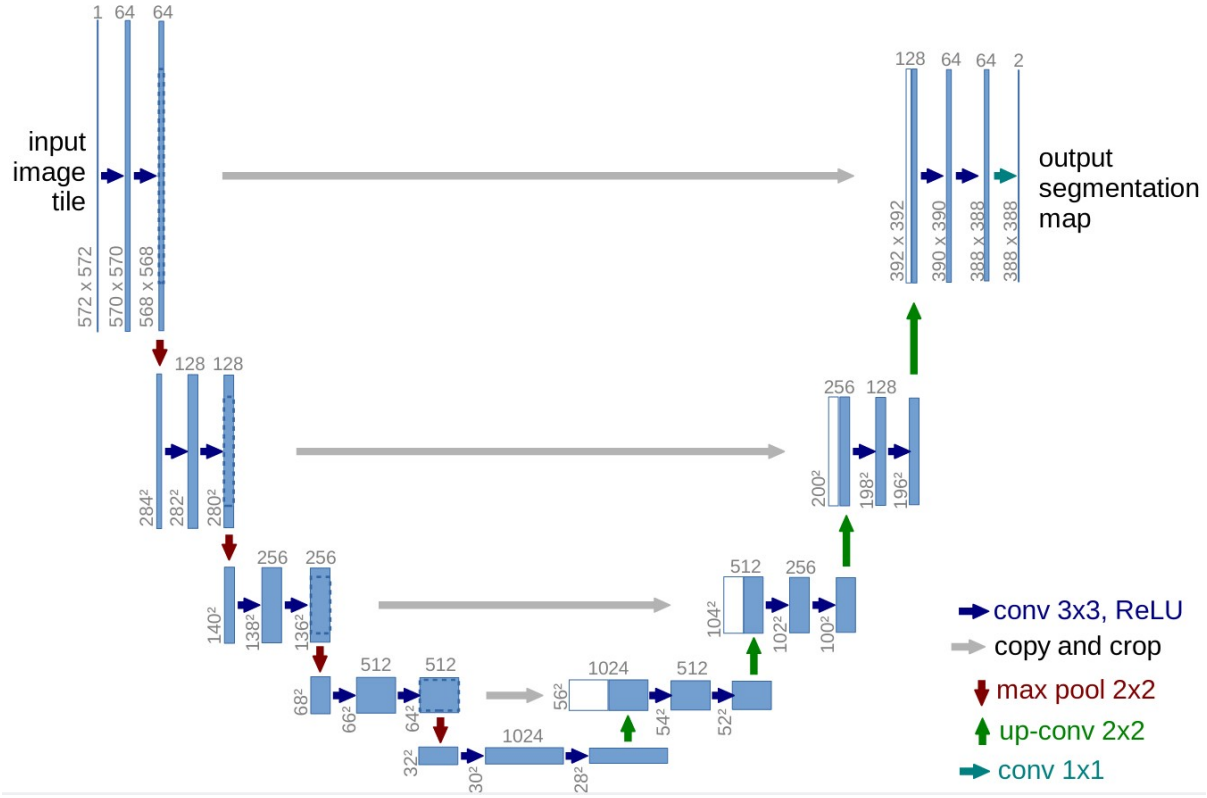


Рисунок 2.4. Архітектура U-Net [9]

Своїй назві ця мережа завдячує архітектурі, подібній на літеру U (рис. 2.4). Тобто вона складається з двох симетричних частин, які ще часто називають енкодер та декодер. Енкодер працює як класична ЗНМ, де кожен шар зменшує роздільну здатність вхідного зображення та збільшує кількість карт ознак на виході. Завдяки ньому ми з заданого зображення отримуємо багато карт ознак невеликої роздільної здатності. Декодер ж навпаки, відновлює повнорозмірне зображення з отриманих на вході карт ознак. І в енкодері, і в декодері використовуються блоки з двох послідовних згорток з розміром фільтра 3x3 та функції активації ReLU. Однак в енкодері, між блоками використовується максимізаційне агрегування (max pooling) 2x2 (рис. 2.5) для зменшення роздільної здатності, а в декодері навпаки — обернена згортка 2x2 (рис. 2.6).

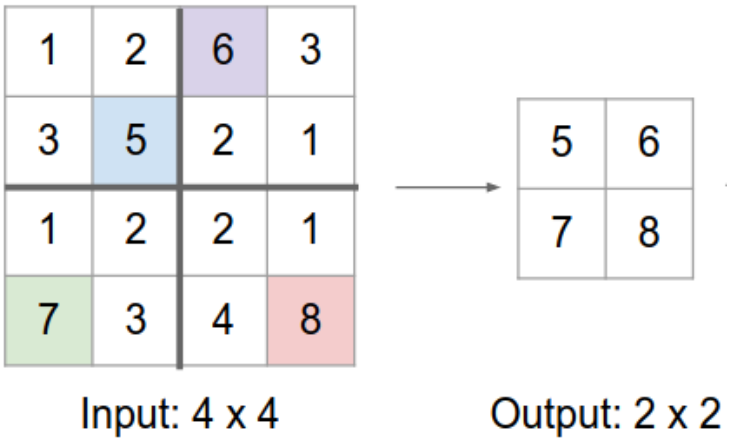


Рисунок 2.5. Приклад максимізаційного агрегування. Вхідне зображення розділяється на блоки та з кожного знаходиться максимум [22]

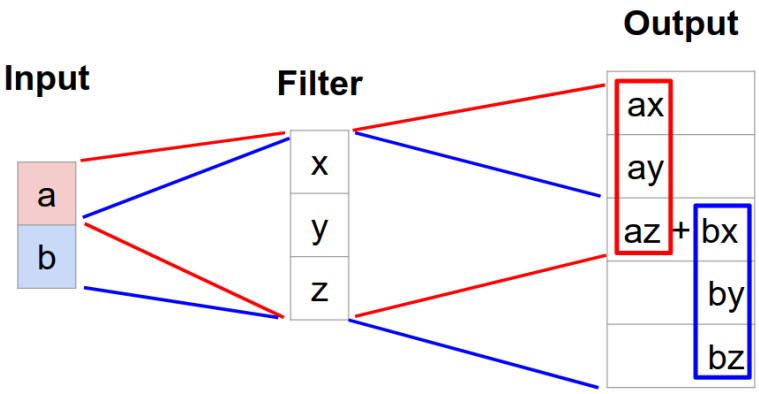


Рисунок 2.6. Приклад оберненої згортки (для спрощення наведений в одновимірному варіанті) [22]

Окрім цього до вхідних карт ознак для кожного блоку декодера конкатенуються відповідні карти ознак з енкодера, це дозволяє передати додаткову інформацію про низькорівневі ознаки та покращити якість результату. Оскільки відповідні зображення в декодері є меншого розміру, для об'єднання використовується обрізане зображення [9].

Сучасні модифікації U-Net зазвичай перед згорткою додають до вхідного зображення рамку, що дозволяє отримати на виході зображення такої самої роздільної здатності і через це обрізання зображень більше не потрібне. Також часто замість оберненої згортки також використовують один з класичних методів збільшення роздільної здатності, як от метод найближчого сусіда чи білінійна

інтерполяція. До більш глобальних модифікацій належить заміна енкодера на одну з популярних архітектур ЗНМ, наприклад ResNet.

Для оцінки точності сегментації популярними метриками є коефіцієнти Джаккара та Дайса.

Коефіцієнт Джаккара між справжнім зображенням та результатом на виході ШНМ визначається як відношення між кількістю пікселів, що належать перетину виділених масок сегментів та загальною кількістю пікселів цих сегментів.

$$J(y,\hat{y})=\frac{|y\cap\hat{y}|}{|y\cup\hat{y}|} \tag{2.5}$$

Коефіцієнт Дайса визначається як подвоєна кількість пікселів перетину сегментів поділена на суму кількостей пікселів кожного сегмента.

$$D(y,\hat{y})=\frac{2|y\cap\hat{y}|}{|y|+|\hat{y}|} \tag{2.6}$$

2.4 Алгоритм покращення якості зображень на основі похідної Рісса

В цьому розділі наведено основні етапи алгоритму застосування оператора FCD так як це описано авторами в роботі [15].

Оператор FCD базується на похідній Рісса (1.6), (1.7), (1.8). Для обчислення похідної Рісса порядку ν ($0 < \nu < 1$) використовується дискретизація з кроком h=1

$$\begin{aligned}\frac{\partial^{\nu}s(x)}{\partial|x|^{\nu}}&=-\frac{1}{2\cos\left(\frac{\pi\nu}{2}\right)}\left(\frac{\partial^{\nu}}{\partial x^{\nu}}+\frac{\partial^{\nu}}{\partial(-x)^{\nu}}\right)\\ &\approx-\frac{1}{2\cos\left(\frac{\pi\nu}{2}\right)h^{\nu}}\left[\sum_{k=0}^{+\infty}\omega_k s(x-kh)+\sum_{k=-\infty}^{0}\omega_k s(x-kh)\right]\end{aligned} \tag{2.7}$$

де

$$\begin{aligned}\omega_0&=-\frac{\Gamma(1-\nu/2)}{\nu\Gamma(1+\nu/2)\Gamma(-\nu)}\\ \omega_k&=\frac{(-1)^{k+1}\Gamma(\nu/2)\Gamma(1-\nu/2)}{\Gamma(\nu/2-k+1)\Gamma(\nu/2+k+1)\Gamma(-\nu)}\quad \text{для k } \pm1,\pm2,\ldots\end{aligned} \tag{2.8}$$

Сам алгоритм застосування оператора FCD на основі похідної Рісса до зображення вимагає трьох вхідних параметрів:

- s — матриця зображення до якого потрібно застосувати оператор
- v — порядок похідної Рісса ($0 < v < 1$)
- m — натуральне число, що визначає розмір вікна (оскільки нам потрібно, щоб маска завжди мала центр, розмір вікна буде $(2m + 1)$ x $(2m + 1)$)

Першим кроком алгоритму є обчислення коефіцієнтів C_s за формулою, що випливає з дискретизації (2.7), (2.8):

$$C_{s_k} = -\frac{1}{2\cos\left(\frac{\pi v}{2}\right)h^v}\omega_k \quad \text{для } k = 0, 1, 2, \ldots, 2m \tag{2.9}$$

Маючи ці коефіцієнти можна побудувати 8 масок W_l, так як зображено на рисунках 2.7 і 2.8.

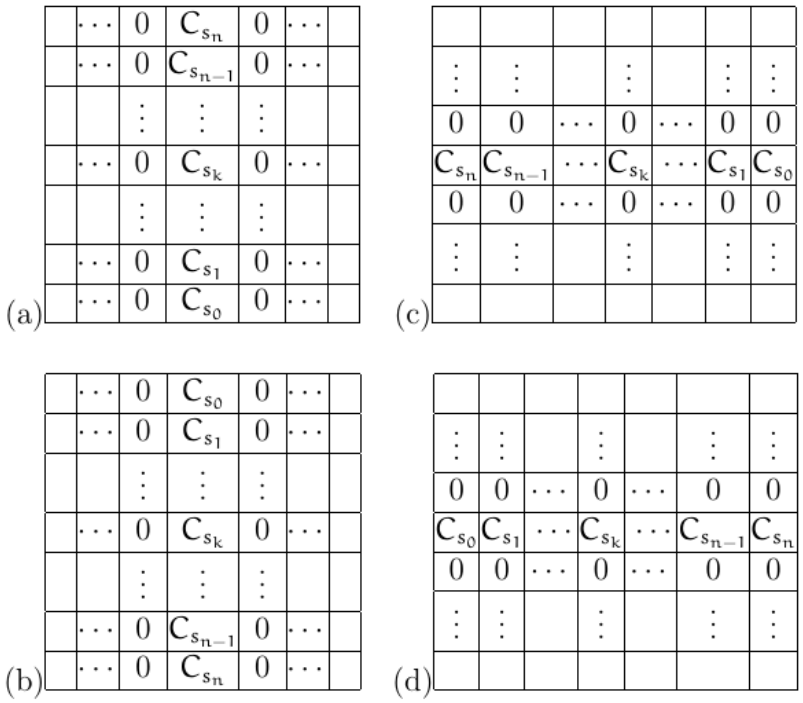


Рисунок 2.7. Маски для чотирьох з восьми напрямків a) W_1 — лівостороння похідна по x; b) W_2 — правостороння похідна по x; c) W_3 — лівостороння похідна по y; d) W_4 — правостороння похідна по y [15]

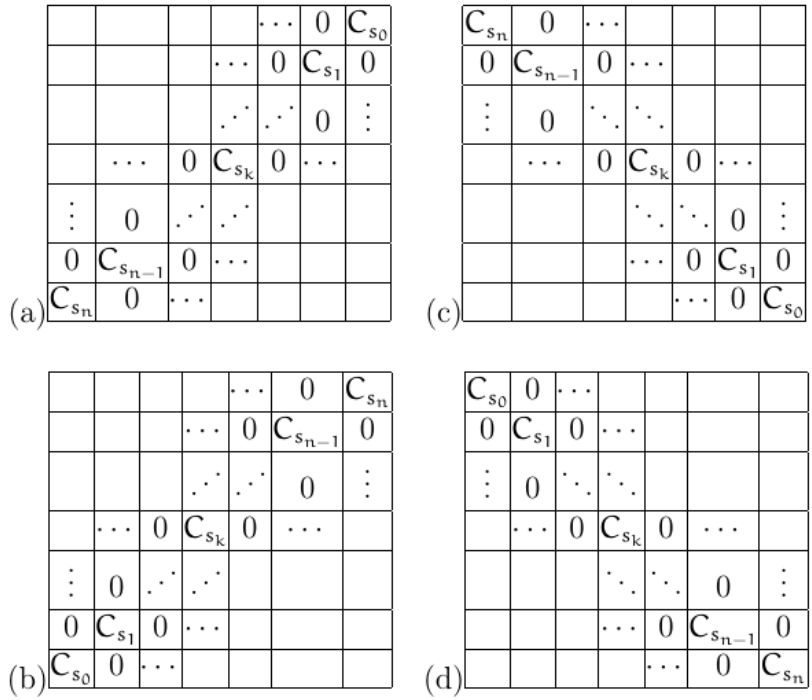


Рисунок 2.8. Маски для інших чотирьох з восьми напрямків a) W_5 — діагональ вліво вниз; b) W_6 — діагональ вправо вгору; c) W_7 — діагональ вліво вгору; d) W_8 — діагональ вправо вгору [15]

Ці маски застосовуються до зображення за допомогою операції згортки, з центром в точці C_{s_0}, тобто коефіцієнт C_{s_0} відповідає поточному пікселю. Для цього для кожного пікселя $s(x, y)$, де x — рядок в матриці s, а y — стовпець, обчислюємо:

$$s_l(x,y)=\sum_{i=M_l}^{N_l}\sum_{j=P_l}^{Q_l} W_l(i,j)s(x+i,y+j) \quad \text{для } l=1,2,3,4 \tag{2.10}$$

$$\begin{aligned} s_l(x,y)&=\sum_{i=M_l}^{N_l}\sum_{j=P_l}^{Q_l} 2^{-\nu/2}W_l(i,j)s(x+i,y+j) \\ &+(1-2^{-\nu/2})W_l(0,0)s(x,y) \quad \text{для } l=5,6,7,8 \end{aligned} \tag{2.11}$$

де

$$\begin{array}{llll} M_1=-2m & N_1=0 & P_1=-m & Q_1=m \\ M_2=0 & N_2=2m & P_3=-2m & Q_2=m \\ M_3=-m & N_3=m & P_3=-2m & Q_3=0 \\ M_4=-m & N_4=m & P_4=0 & Q_4=2m \\ M_5=0 & N_5=2m & P_5=-2 & Q_5=0 \\ M_6=-2m & N_6=0 & P_6=0 & Q_6=2m \\ M_7=-2m & N_7=0 & P_7=-2m & Q_7=0 \\ M_8=0 & N_8=2m & P_8=0 & Q_8=2m \end{array}$$

Після цього сумуємо результати застосування всіх масок, щоб отримати остаточне значення:

$$s(x,y)=\frac{\sum_{l=1}^{8} s_l(x,y)}{4\left[\sum_{k=0}^{n} C_{s_k}+\sum_{k=1}^{n} 2^{-v/2} C_{s_k}+C_{s_0}\right]} \tag{2.12}$$

В результаті отримуємо зображення з покращеною текстурою.

2.5 Фрактальна фізико-інформована нейронна мережа (fPINN)

Фізико-інформована нейронна мережа (Physics-informed neural network - PINN) — це модифікація ШНМ розроблена спеціально для розв’язування диференційних рівнянь. Головною перевагою таких мереж є те, що окрім тренувальних даних, такі мережі дозволяють врахувати інформацію про фізичні закони, що визначають систему, яку ми намагаємось апроксимувати нейронною мережею.

Фізико-інформовані нейронні мережі базуються на двох фундаментальних властивостях ШНМ:

- Теоремі про універсальний апроксиматор функцій
- Легкості обрахунку похідних результату на виході ШНМ завдяки алгоритму автоматичного диференціювання, що активно застосовується при навчанні ШНМ

По суті PINN є звичайним багатошаровим перцептроном, функція втрат якого побудована на основі заданого диференційного рівняння. Оскільки відповідно до теореми Цибенка багатошаровий перцептрон може апроксимувати будь-яку функцію, PINN використовується для наближення невідомої функції u, яка є шуканим розв’язком рівняння. Тоді в якості функції втрат береться середньоквадратична різниця між лівою та правою частинами рівняння в які замість функції u підставлено вивід нейронної мережі u^* для всіх тренувальних даних. При цьому алгоритм автоматичного диференціювання дозволяє ефективно обчислювати похідні від виходу ШНМ, тобто від наближеного розв’язку u^*, по всіх змінних. Цей самий алгоритм використовується і для знаходження похідних

по вагах нейронної мережі, що потрібно для оптимізації функції втрат методом градієнтного спуску.

Однак похідні дробового порядку не можна обчислити за допомогою алгоритму автоматичного диференціювання, тому таку модель не вдається ефективно застосовувати для розв'язання диференційних рівнянь дробового порядку. Для вирішення цієї проблеми було запропоновано фрактальну фізико-інформовану нейронну мережу (Fractional physics-informed neural network — fPINN) [23].

Ця модель побудована на основі PINN, але використовує гібридний підхід, відповідно до якого похідні цілочисельного порядку обчислюються за допомогою алгоритму автоматичного диференціювання, а для операторів дробового порядку використовуються чисельні дискретизації (рис. 2.9).

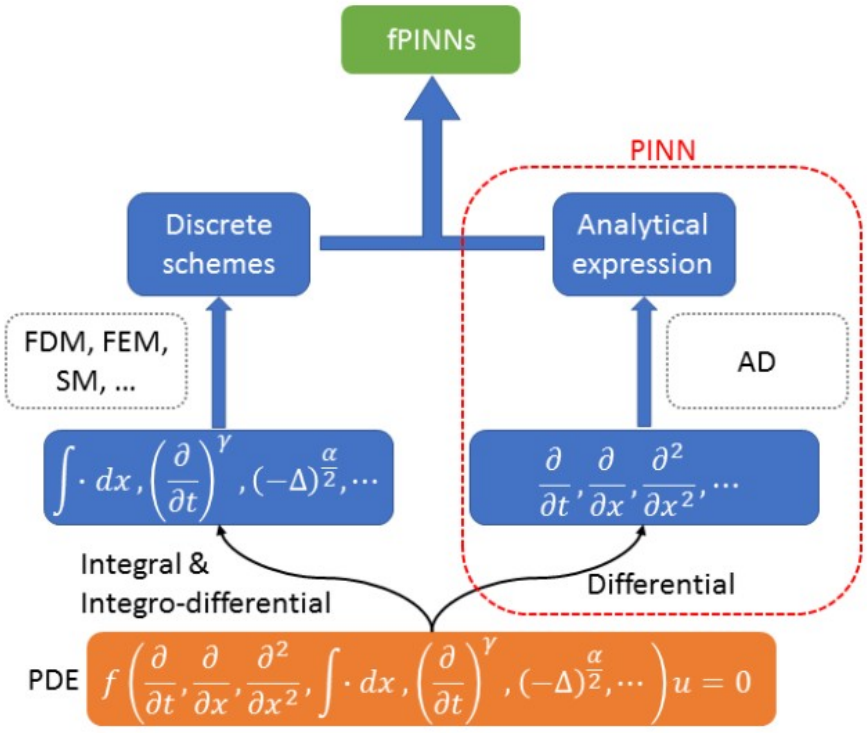


Рисунок 2.9. fPINN може працювати з операторами як цілочисельного, так і дробового порядку [23]

2.6 ШНМ для апроксимації довільних операторів DeepONet

Як згадувалось раніше ШНМ є універсальним апроксиматором функцій. Однак менш відомою властивістю є те, що за допомогою ШНМ так само можна

апроксимувати довільний нелінійний оператор. Про це говорить теорема, яку довели Чен та ін. [24].

Теорема (Універсальний апроксиматор операторів). Нехай σ — неперервна не поліноміальна функція, X — простір Банаха, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ — дві компактні множини в X та $\mathbb{R}^d$ відповідно, V — компактна множина в $C(K_1)$, G — нелінійний неперервний оператор, що відображає множину V в $C(K_2)$. Тоді для будь-якого $\epsilon > 0$, існують додатні цілі числа n, p, m, константи $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}, w_k \in \mathbb{R}^d$, $i = 1,\ldots,n, k = 1,\ldots,p, j = 1,\ldots,m$ такі, що

$$\left| G(u)(y) - \underbrace{\sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon \tag{2.13}$$

виконується для всіх $u \in V$ та $y \in K_2$.

Базуючись на цій теоремі, Лу та ін. 2020 року запропонували мережу DeepONet [25]. Ця мережа складається з двох частин branch та trunk, кожна з яких є звичайним багатошаровим перцептроном, хоча можуть використовуватись і складніші архітектури.

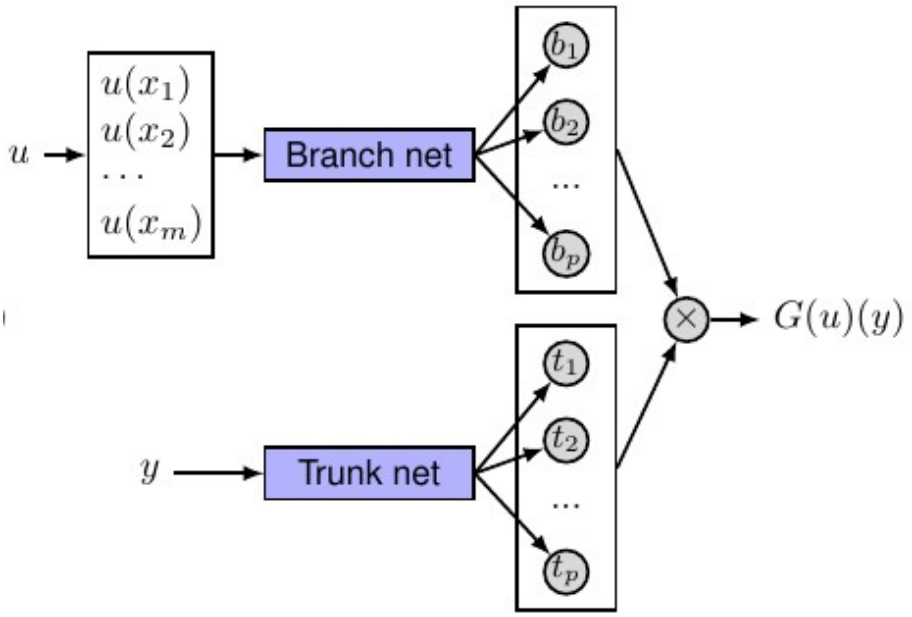


Рисунок 2.10. Архітектура мережі DeepONet [25]

Branch-мережа на вході очікує функцію $u(x)$ задану у вигляді вектора, що містить значення функції $u(x)$ в m точках (які автори називаються сенсорами) $\{x_1, x_2, ..., x_m\}$, де x_i — вектор розмірності d. При цьому єдиним обмеженням на

вхідні дані є те, що для всіх функцій в датасеті сенсори повинні бути розміщені в однакових локаціях.

Trunk-мережа при тренуванні очікує локації сенсорів використаних для вхідних функцій, що задаються у вигляді матриці розміру mxd. А натренована мережа вже дозволяє знайти значення вихідної функції $G(u)(y)$ у будь-якій точці y, навіть якщо вона не відповідає жодному сенсору, що є значною перевагою цієї моделі.

2.7 Висновки

В цьому розділі наведено структуру датасету та детальний огляд певних алгоритмів, які в результаті аналізу літератури, було вирішено застосувати в даній роботі. А також для деяких алгоритмів наведено теоретичні основи на яких вони базуються.

Загалом планується розробити алгоритм в якому до вхідного зображення КТ для покращення якості текстури буде застосовуватись похідна Рісса, обчислена за допомогою fPINN або DeepONet, після чого це зображення буде передаватись в мережу U-Net для сегментації крововиливу.

Деталі програмної реалізації розглянутих алгоритмів наведено у наступному розділі.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Підготовка даних

Як і в оригінальній статті, зображення NIfTI формату були поділені на окремі шари, до яких було застосоване вікно (WL: 40, WW: 120), щоб виділити тканини мозку (рис. 3.1). Отримані зображення було збережено у 8-бітному форматі PNG у дві директорії: скани — в image, а відповідні маски з такою самою назвою файлу — в label. Поділ на тренувальну та тестову вибірку буде здійснений безпосередньо перед тренуванням.

Оскільки, окрім розробки моделі здатної сегментувати внутрішньочерепні крововиливи, ціллю цієї роботи є також дослідження масок на основі дробових похідних та фрактальних фізико-інформованих нейронних мереж, було вирішено спростити першу задачу. Тому перед мережею ставиться лише вимога сегментація крововиливу будь-якого типу на зображенні. Тобто звичайна задача бінарної сегментації, де кожному пікселю потрібно поставити у відповідність мітку одного чи іншого класу. Більше того, будуть розглядатись лише зображення, де крововилив точно присутній. Таких як раніше згадувалось у датасеті є лише 318. Це обмеження не є надто штучним, оскільки, враховуючи точність, з якою сьогодні вдається класифікувати цілі КТ-скани, видається цілком ймовірним створення в майбутньому ансамблю з двох моделей, де одна мережа класифікує зображення, і, вже якщо воно відноситься до класу з наявним крововиливом, воно передається на вхід іншої мережі, для точного визначення розташування цього пошкодження.

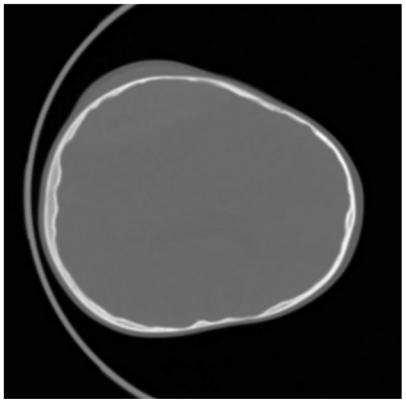
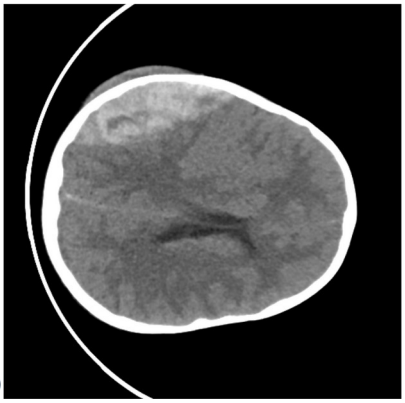
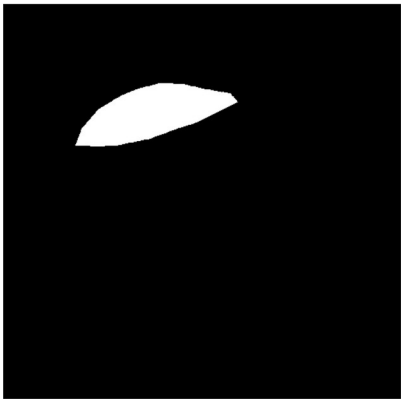
а)  б)  в) 

Рисунок 3.1. Приклад зображення з датасету а) до обробки б) після застосування вікна (WL: 40, WW:120) в) маска, що виділяє область з крововиливом

3.2 Використані технології

Технологія CUDA дозволяє використовувати потужності графічного процесора для виконання математичних обчислень. Як було розглянуто в попередній роботі, це дозволяє значно прискорити процес тренування. Однак, використання цієї технології для навчання ШНМ напряму з мов C/C++, було б дуже складним, оскільки вимагало б приділяти дуже багато часу низькорівневим деталям реалізації. Тому було розроблено чимало фреймворків для роботи з ШНМ більш високорівневими мовами, як от Python. В цій роботі було використано один з таких фреймворків — PyTorch.

PyTorch, як і Chainer, що використовувався в [7], базується на моделі Define-by-Run (рис. 3.2). Основною ідеєю якого є те, що граф обчислень будується динамічно прямо під час тренування. Під час проходу вперед, кожна функція, окрім обчислення значення, зберігає історію обчислень разом з посиланням на попередній вузол у графі. Оскільки структура графу залежить від шляху виконання програми, стало можливе використання синтаксичних конструкцій мови програмування, таких як умовні оператори та цикли. Окрім цього тепер з'явилась можливість використання зневаджувачів для детальнішого дослідження процесу тренування [26].

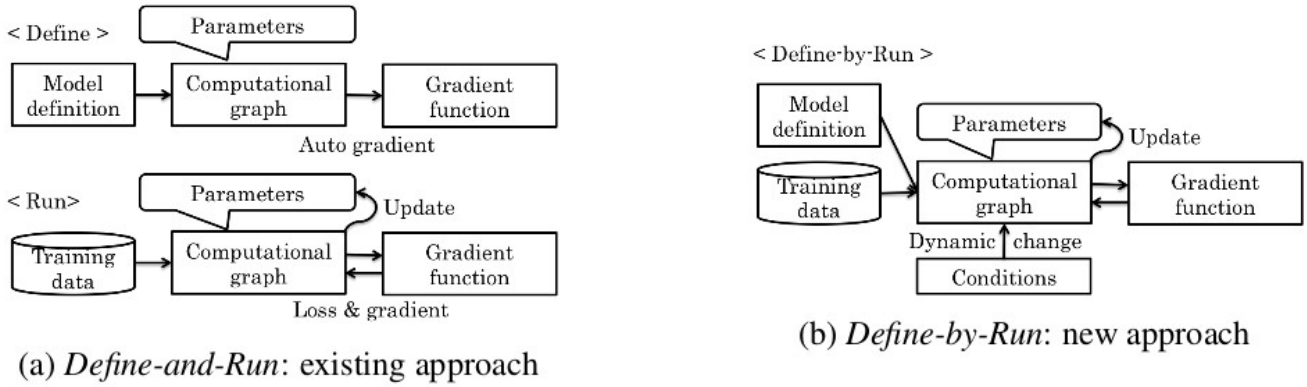


Рисунок 3.2. Схема процесу тренування для підходу
a)Define-and-Run b)Define-by-Run [26]

Окрім цього, перевагою PyTorch є активне ком'юніті. Через це існує багато додаткових пакетів, для спрощення роботи з цим фреймворком. Одним з таких є MedicalTorch — невелика бібліотека, що містить реалізації популярних архітектур

ШНМ для роботи з медичними даними та деякі допоміжні функції для їх тренування [27]. Цю бібліотеку також було використано в цій роботі.

Для роботи з фізико-інформованими нейронними мережами застосовувалась бібліотека DeepXDE. Це спеціальна бібліотека для наукового машинного навчання, що містить реалізації багатьох ШНМ, що досліджуються в цій сфері, серед них і fPINN та DeepONet, які нас цікавлять. При чому в цій бібліотеці наявні реалізації для різних популярних фреймворків (серед яких і PyTorch) та надано спільний інтерфейс, що приховує деталі реалізації та дозволяє використовувати той самий код, не залежно від того який фреймворк використовується.

Як середовище програмування було обрано Jupyter Notebook, оскільки інтерактивний режим виконання програми, що він пропонує є зручним для досліджень. Окрім цього він дозволяє зручно візуалізовувати дані, а ipynb-файли які створюються в ньому (також їх називають просто ноутбуками) підтримуються для виконання багатьма хмарними сервісами.

Для дослідження активно використовувався хмарний сервіс Google Colab, оскільки він є безкоштовним та зручним у використанні. А також надає доступ до потужних GPU, що необхідні для тренування ШНМ.

3.3 Навчання ШНМ U-Net за допомогою фреймворку PyTorch

Для тренування ШНМ потрібні такі компоненти [28]:

- Model — сама ШНМ, представлена у вигляді об’єкта класу, що наслідує `torch.nn.Module` та реалізовує метод `forward`, який описує прямий прохід по нейронній мережі
- Dataset — об’єкт класу, що наслідує `torch.utils.data.Dataset` та реалізує методи `__len__` та `__getitem__` потрібні для отримання кількості елементів датасету та отримання довільного елементу за індексом відповідно. Оскільки набір даних зазвичай дуже великий, щоб повністю тримати його в пам’яті, Dataset використовується, щоб при доступі до елементу набору даних зчитати його і та за потреби застосувати певні перетворення

- DataLoader — об’єкт, що використовується для ефективного завантаження об’єктів датасету, при створенні приймає об’єкт датасету, та набір налаштувань, щодо того як завантажувати дані: розмір порції (`batch_size`), кількість потоків використаних для завантаження (`num_workers`) та чи потрібно перемішувати дані (`shuffle`)
- Optimizer — об’єкт, що при створені приймає набір параметрів моделі та додаткові параметри конкретного методу оптимізації, він використовується для самого навчання, тобто оновлення параметрів мережі для мінімізації функції втрат

Спочатку було створено клас ICHDataset (рис. 3.3). Конструктор цього класу приймає шлях до директорії, що містить дві піддиректорії image і label та опційний параметр `transform`. Кожен елемент датасету є словником, де ключу `"input"` відповідає саме зображення, а ключу `"gt"` — правильно сегментована маска. Окрім цього є ще два пусті словники для запису метаданих. Такого формату даних вимагають перетворення зображень з бібліотеки MedicalTorch.

Після цього було створено відповідний об’єкт, поділено датасет на тренувальну та тестову вибірки, таким чином, що 80% випадково вибраних зображень було виділено для тренування, а решта для тестування. До тренувальної частини було додано трансформацію у вигляді випадкового афінного перетворення для збільшення різноманітності даних. Також було створено відповідні завантажувачі даних (рис. 3.4).

```python
class ICHDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.scans_dir = os.path.join(root_dir, "image")
        self.masks_dir = os.path.join(root_dir, "label")
        self.fnames = os.listdir(self.scans_dir)
        self.transform = transform

    def __len__(self):
        return len(self.fnames)

    def __getitem__(self, idx):
        scan_path = os.path.join(self.scans_dir, self.fnames[idx])
        mask_path = os.path.join(self.masks_dir, self.fnames[idx])

        scan = torchvision.io.read_image(scan_path, torchvision.io.image.ImageReadMode.GRAY).float() / 255
        mask = torchvision.io.read_image(mask_path, torchvision.io.image.ImageReadMode.GRAY).float() / 255

        data_dict = {
            'input': scan,
            'gt': mask,
            'input_metadata': {},
            'gt_metadata': {},
        }

        if self.transform is not None:
            scan = self.transform(data_dict)

        return data_dict

    def set_transform(self, transform):
        self.transform = transform
```

Рисунок 3.3. Клас ICHDataset

```python
dataset = ICHDataset(DATA_ROOT)
train_len = int(0.8 * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(
    dataset,
    [train_len, len(dataset) - train_len],
    generator=torch.Generator().manual_seed(42))

transforms = torchvision.transforms.Compose(
    [mt_transforms.ToPIL(),
     mt_transforms.RandomAffine(degrees=20, translate=(0.1, 0.1), scale=(0.8, 1.2)),
     mt_transforms.ToTensor()]
)
train_dataset.dataset.set_transform(transforms)


train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=True, num_workers=NUM_WORKERS)
```

Рисунок 3.4. Створення датасету та завантажувачів даних

В якості нейронної мережі для сегментації було використано стандартну архітектуру U-Net з бібліотеки MedicalTorch. Варто зауважити, що це не зовсім описаний раніше класичний варіант цієї мережі, а новіша її модифікація. В ній перед згортками до вхідних даних додаються рамки, для того, щоб на виході отримати зображення такого самого розміру, як і на вході. Також замість оберненої

згортки виконується білінійна інтерполяція. Окрім цього в блоках з подвійною згорткою використовується нормалізація за порцією та дропаут.

Для створення моделі достатньо вказати параметри нормалізації та дропауту, або залишити їх стандартними. Після чого використовуючи метод `to(device)` де `device` повинен мати значення `"cuda"` для тренування за допомогою GPU, або відповідно `"cpu"` для CPU. Також потрібно пам'ятати, що дані, які передаємо мережі мають бути розташовані на цьому ж приладі, що і мережа. В більшості випадків це все що потрібно знати для використання графічного прискорювача за допомогою фреймворку PyTorch. Такий спосіб є дуже зручним, так як дозволяє розробнику фокусуватись над його основною роботою, а все розпаралелення відбувається автоматично. Якщо потрібно ефективно розпаралелити певну мало поширену операцію, реалізації якої у фреймворку нема, є можливість написати власне CUDA ядро [28]. Проте, в цьому дослідженні такої потреби не виникло.

Окрім моделі, також створюється оптимізатор Adam, а в якості функції втрат використовується коефіцієнт Дайса з від'ємним знаком. Через це його значення будуть лежить в межах від 0 до -1, а при навчанні, значення функції спадати.

```
model = mt_models.Unet().float().to(DEVICE)
loss_fn = mt_losses.dice_loss
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-8)
writer = SummaryWriter(os.path.join(LOG_DIR, datetime.datetime.now().strftime("%Y%m%d-%H%M%S")))
```

Рисунок 3.5. Створення мережі та підготовка до тренування

Однією з головних частин є також тренувальний цикл. Так називають функцію яка саме описує алгоритм тренування, PyTorch вимагає ручного написання цієї функції. З однієї сторони це часто змушує писати багато однакового коду, але з іншої надає гнучкості, адже для різних задач, може знадобитись різний алгоритм навчання.

```python
def train(model, dataloader, test_dataloader, loss_fn, optimizer, start_epoch, epochs_count, device, writer):
    for epoch in range(start_epoch, start_epoch+epochs_count):
        print(f"Epoch: {epoch}")
        # show network output for random test image
        generate_images(model, next(iter(test_dataloader)))

        running_loss = 0.0
        loop = tqdm.notebook.tqdm(dataloader)

        for i, batch in enumerate(loop, 0):
            # send inputs to selected device
            inputs = batch["input"].to(device)
            labels = batch["gt"].to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()

            # log
            writer.add_scalar("Loss/train", loss.item(), epoch)
            loop.set_postfix(loss=loss.item())
            running_loss += loss.item()

        print(f"Loss: {running_loss / len(dataloader)}")
        # save checkpoint
        if (epoch + 1) % 10 == 0:
            torch.save({
                'epoch': epoch,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
            }, f"./model-{epoch}.tar")
```

Рисунок 3.6. Тренувальний цикл

На рисунку 3.6 можна бачити використаний тренувальний цикл. Більшість коду тут відповідає за логування, візуалізацію проміжних результатів та збереження параметрів моделі. Основна частина з прямим проходом, зворотнім поширенням помилки та оптимізацією параметрів мережі займає доволі небагато рядків.

3.4 Особливості застосування сервісу хмарних обчислень Google Colab

Окрім безкоштовності, ще однією перевагою Colab є простота у користуванні. Дійсно для того, щоб розпочати роботу достатньо перейти за посиланням: https://colab.research.google.com/. В результаті буде відкрито середовище подібне до Jupyter Notebook, з яким одразу ж можна починати роботу. При цьому більшість популярних бібліотек вже буду встановлені на віртуальній машині, а більш вузькоспеціалізовані можна завантажити командою `!pip install`, в будь-якій вільній комірці. У випадку цієї роботи так потрібно було встановити MedicalTorch. Також можна завантажити наявний ноутбук у хмарне

сховище Google Drive, після чого просто відкрити його як звичайний файл, за замовчування його буде завантажено у Colab.

Google Drive можна використовувати і для зберігання даних для тренування ШНМ. Щоб використати їх в Colab, потрібно підключитись до свого сховища командою з рисунку 3.7. Після чого можна буде доступитись до даних диску за шляхом `/content/drive/MyDrive`.

```
from google.colab import drive
drive.mount('/content/drive')
```

Рисунок 3.7. Підключення Google Drive в Colab

За потреби в графічному процесорі для обчислень, потрібно перейти у Runtime -> Change runtime type та змінити поточне середовище виконання з CPU на GPU. На жаль, не можна обрати який саме графічний процесор буде наданий, оскільки його автоматично обирає балансувальник навантаження. Загалом наявні такі моделі GPU: Nvidia Tesla K80, Nvidia Tesla T4, Nvidia Tesla P4 та Nvidia Tesla P100 [29]. На момент досліджень, була використана Nvidia Tesla T4 з 16 Гб відеопам’яті.

3.5 Реалізація масок на основі похідних Рісса дробового порядку

За допомогою мови програмування Python та бібліотеки алгоритмів комп’ютерного зору OpenCV відповідно до алгоритму викладеного у [15] було реалізовано функцію `fractional_differential_mask(img, nu, m)`, яка для зображення `img` обчислює похідну Рісса порядку `nu`, з розміром вікна, що вказується параметром `m`. Відповідно до алгоритму розмір маски $(2m+1, 2m+1)$.

Спочатку знаходимо коефіцієнти C_s за формулою (2.9):

```
k = np.arange(0, n+1)
w = np.empty(n+1)
w[0] = - gamma(1 - nu / 2) / (nu * gamma(1 + nu / 2) * gamma(-nu))
w[1:] = ((-1)**(k[1:] + 1) * gamma(nu / 2) * gamma(1 - nu / 2)) \
      / (gamma(nu / 2 - k[1:] + 1) * gamma(nu / 2 + k[1:] + 1) * gamma(-nu))

Cs = -w / (2 * np.cos(np.pi * nu / 2) * h**nu)
```

Рисунок 3.8. Обчислення коефіцієнтів C_s

Далі з використанням знайдених коефіцієнтів побудуємо 8 масок, для кожного напрямку:

```
W = []

center = mask_size[0] // 2
Wl = np.zeros(mask_size)
Wl[:, center] = Cs[::-1]
W.append(Wl)

Wl = np.zeros(mask_size)
Wl[:, center] = Cs
W.append(Wl)

Wl = np.zeros(mask_size)
Wl[center] = Cs[::-1]
W.append(Wl)

Wl = np.zeros(mask_size)
Wl[center] = Cs
W.append(Wl)

Wl = np.zeros(mask_size)
np.fill_diagonal(np.fliplr(Wl), Cs)
W.append(Wl)

Wl = np.zeros(mask_size)
np.fill_diagonal(np.flipud(Wl), Cs)
W.append(Wl)

Wl = np.zeros(mask_size)
np.fill_diagonal(Wl, Cs[::-1])
W.append(Wl)

Wl = np.zeros(mask_size)
np.fill_diagonal(Wl, Cs)
W.append(Wl)
```

Рисунок 3.9. Побудова масок для кожного напрямку

Для застосування масок реалізовано допоміжну функцію `compute_sl(l,s)`, яка для вхідного зображення `s` застосовує маску з індексом `l`. При цьому для пришвидшення виконання згортка (2.10)-(2.11) виконується за допомогою функції `filter2d` з бібліотеки OpenCV (рис. 3.10).

```python
def compute_sl(l, s):
    Wl = W[l]
    center = [(2*m, m), (0, m), (m, 2*m), (m, 0),
              (0, 2*m), (2*m, 0), (2*m, 2*m), (0, 0)]

    cx, cy = center[l]
    anchor = (cy, cx)

    if 0 <= l < 4:
        sl = cv2.filter2D(s, cv2.CV_64F, Wl, anchor=anchor)
    elif 4 <= l < 8:
        sl = 2**(-nu/2) * cv2.filter2D(s, cv2.CV_64F, Wl, anchor=anchor)
        sl += (1 - 2**(-nu/2)) * Wl[0 + cx, 0 + cy] * s
    else:
        raise ValueError("Wrong value of parameter 'l', must be 0 <= l < 8")

    return sl
```

Рисунок 3.10. Допоміжна функція для застосування маски з індексом `l`

Обчислюємо всі 8 масок, та сумуємо їх за формулою (2.12), щоб отримати остаточний результат

```python
s = []
for l in range(8):
    sl = compute_sl(l, W[l], img)
    s.append(sl)

S = np.sum(s, axis=0) / (4 * (sum(Cs) + sum(2**(-nu/2) * c for c in Cs[1:]) + Cs[0]))
```

Рисунок 3.11. Обчислення результату, що враховує всі 8 масок

3.6 Застосування fPINN та DeepONet за допомогою бібліотеки DeepXDE

Бібліотека DeepXDE [30] містить реалізації багатьох фізико-інформованих моделей та підтримує кілька популярних фреймворків для глибинного навчання, серед яких і PyTorch. При цьому ця бібліотека надає зручний високорівневий інтерфейс, завдяки якому можна використовувати той самий код не залежно від того який фреймворк використовується. Ще одною важливою перевагою DeepXDE є те, що у разі наявності GPU, автоматично застосовується розпаралелений алгоритм навчання за допомогою технологій CUDA.

Основним класом цієї бібліотеки є `dde.Model`. Він об’єднує дані та архітектуру ШНМ. Тому для створення об’єкту цього класу нам потрібні два об’єкти що наслідують класи `dde.data.Data` та `dde.nn.NN`, для

представлення даних та архітектури ШНМ відповідно. Після цього за допомогою методу `Model.compile` потрібно підготувати мережу до тренування та розпочати безпосередньо саме тренування за допомогою методу `Model.train`. Отримати результат від натренованої мережі можна за допомогою методу `Model.predict`. На рисунках 3.12 та 3.13 наведено схему роботи з PINN та DeepONet за допомогою бібліотеки DeepXDE.

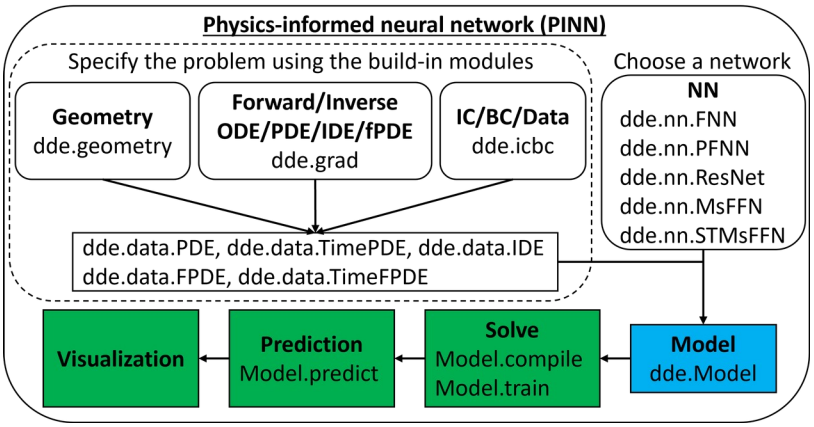


Рисунок 3.12. Схема роботи з PINN за допомогою бібліотеки DeepXDE [30]

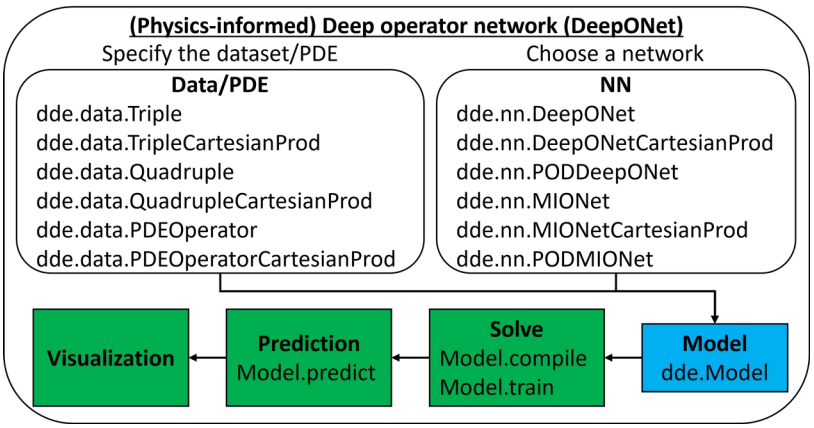


Рисунок 3.13. Схема роботи з DeepONet за допомогою бібліотеки DeepXDE [30]

3.6.1 fPINN

Для fPINN використовується мережа `dde.nn.FNN` (Fully-connected Nueral Network), що є звичайним багатошаровим перцептроном. Але в якості даних передається об'єкт `dde.data.FPDE` в якому якраз і задається диференціальне рівняння з похідними дробового порядку та тренувальні дані.

В документації [30] розглядається застосування fPINN для розв’язання рівняння Пуассона з прикладу в роботі [23]:

$$(-\Delta)^{\nu/2}u(x)=f(x) \quad x\in(0,1) \tag{3.1}$$

з крайовою умовою Діріхле $u(0)=u(1)=0$, де

$$f(x)=\frac{1}{2\cos(\pi\nu/2)}\left[\frac{\Gamma(4)}{\Gamma(4-\nu)}\left(x^{3-\nu}+(1-x)^{3-\nu}\right)-\frac{3\Gamma(5)}{\Gamma(5-\nu)}\left(x^{4-\nu}+(1-x)^{4-\nu}\right)\right.$$
$$\left.+\frac{3\Gamma(6)}{\Gamma(6-\nu)}\left(x^{5-\nu}+(1-x)^{5-\nu}\right)-\frac{\Gamma(7)}{\Gamma(7-\nu)}\left(x^{6-\nu}+(1-x)^{6-\nu}\right)\right] \tag{3.2}$$

```
nu = 1.5

def fpde(x, y, int_mat):
    """(D_{0+}^nu + D_{1-}^nu) u(x) = f(x)"""
    if isinstance(int_mat, (list, tuple)) and len(int_mat) == 3:
        int_mat = tf.SparseTensor(*int_mat)
        lhs = tf.sparse_tensor_dense_matmul(int_mat, y)
    else:
        lhs = tf.matmul(int_mat, y)
    rhs = (
        gamma(4) / gamma(4 - nu) * (x ** (3 - nu) + (1 - x) ** (3 - nu))
        - 3 * gamma(5) / gamma(5 - nu) * (x ** (4 - nu) + (1 - x) ** (4 - nu))
        + 3 * gamma(6) / gamma(6 - nu) * (x ** (5 - nu) + (1 - x) ** (5 - nu))
        - gamma(7) / gamma(7 - nu) * (x ** (6 - nu) + (1 - x) ** (6 - nu))
    )
    return lhs - rhs[: tf.size(lhs)]

def func(x):
    """Analytical solution to given FPDE"""
    return x ** 3 * (1 - x) ** 3

# set boundary condition
geom = dde.geometry.Interval(0, 1)
bc = dde.icbc.DirichletBC(geom, func, lambda _, on_boundary: on_boundary)

# create FPDE object that contains all info about given FPDE
data = dde.data.FPDE(geom, fpde, nu, bc, [101], meshtype="static", solution=func)

# create multilayer perceptron with 4 hidden layers, 20 neurons each
net = dde.nn.FNN([1] + [20] * 4 + [1], "tanh", "Glorot normal")

# transform output to satisfy boundary conditions automatically
net.apply_output_transform(lambda x, y: x * (1 - x) * y)

# create and train the model
model = dde.Model(data, net)
model.compile("adam", lr=1e-3)
losshistory, train_state = model.train(iterations=10000)
dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

Рисунок 3.14. Розв’язок рівняння (3.1) за допомогою fPINN

Аналітичний розв'язок цього рівняння відомий: $u(x) = x^3(1 - x)^3$. Але для прикладу на рис. 3.14 продемонстровано його розв'язок за допомогою fPINN реалізованої в бібліотеці DeepXDE.

Для тренування використовується N точок $x_j = j / N$, де $j = 1, 2, ..., N — 1$. При цьому нам не потрібно враховувати точки на краях, оскільки наближений розв'язок $u^*(x)$ сформовано таким чином, щоб він автоматично задовольняв крайові умови $u^*(x) = x(1 - x)u_{NN}(x; \mu)$, де $u_{NN}(x; \mu)$ — вивід нейронної мережі з вагами μ для точки x. Щоб задати це, в коді було використано метод об'єкту `FNN` `apply_output_transform`.

Як бачимо мережа fPINN дозволяє доволі легко знайти наближений розв'язок $u^*(x)$. Однак в нашому випадку потрібно наблизити сам оператор $(-\Delta)^{\nu/2}$, який хоч і обчислюється при пошуку наближеного розв'язку всередині мережі, але оскільки застосування fPINN для його апроксимації є досить нетиповим, таке використання вимагає глибшого дослідження. Тому в цій роботі також було розглянуто іншу мережу, яка вже розроблялась саме для апроксимації операторів — DeepONet.

3.6.2 DeepONet

Для застосування DeepONet потрібно організувати датасет таким чином:

- Вхідні дані для branch мережі — матриця розміру (розмір датасету, N_u), містить вхідні функції, задані дискретно у вигляді N_u точок
- Вхідні дані для trunk мережі — матриця розміру (N_u, розмірність), вказує координати x значень функції $u(x)$ які подаються на вході branch мережі, у випадку зображення містить пари (x, y) координат кожного пікселя в датасеті
- Очікуваний результат — матриця розміру (розмір датасету, N_u), результат застосування оператора, який повинна вивчити мережа, до вхідних даних, в нашому випадку обчислюється за допомогою алгоритму описаного раніше

Ці дані зберігаються в структурі `dde.data.TripleCartesianProd` та разом з самою мережею `dde.nn.DeepONetCartesianProd` передаються в конструктор `dde.Model` для створення відповідної моделі (рис. 3.15).

```
data = dde.data.TripleCartesianProd(
    X_train=X_train, y_train=y_train, X_test=X_test, y_test=y_test
)

# Choose a network
branch_input = 100
trunk_input = 1
n_neurons = 40
n_iterations = 20000

net = dde.nn.DeepONetCartesianProd(
    [branch_input, n_neurons, n_neurons],
    [trunk_input, n_neurons, n_neurons],
    "relu",
    "Glorot normal",
)

# Define a Model
model = dde.Model(data, net)

# Compile and Train
model.compile("adam", lr=0.001, metrics=["mean l2 relative error"])
losshistory, train_state = model.train(iterations=n_iterations)

# Plot the loss trajectory
dde.utils.plot_loss_history(losshistory)
plt.show()
```

Рисунок 3.15. Тренування мережі DeepONet за допомогою бібліотеки DeepXDE

3.7 Висновки

- Для мережі U-Net було підібрано коефіцієнт навчання 0.001, а розмір порції 4. Модель було натреновано протягом 100 епох на навчальній вибірці. Точність сегментації було перевірено на тестовому наборі використовуючи метрики Джаккара та Дайса.
- Після цього було реалізовано оператор на основі похідної Рісса. Цей оператор з похідною порядку 0.5 та розміром маски 3x3 було застосовано до всього датасету та повторено тренування мережі U-Net з такими самими налаштуваннями, щоб порівняти результати.
- Також було розглянуто можливість апроксимації згаданого оператора за допомогою fPINN та DeepONet та проведено відповідні експерименти.

РОЗДІЛ 4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Аналіз результатів вирішення задачі сегментації

Розглянемо наскільки добре справляється натренована мережа з поставленою прикладною задачею — сегментацією внутрішньочерепних крововиливів за даними комп’ютерної томографії. Для цього використовуються 20% раніше відділених зображень, що не застосовувались для навчання мережі.

Протягом тренування, після кожної епохи ШНМ було застосовано для сегментації випадкового зображення з тестової вибірки, для того щоб подивитись як міняється якість сегментації з часом. На рисунках 4.1-4.4 наведені деякі з цих зображень, де зліва можна бачити вхідне зображення, посередині очікуваний результат, а справа вивід мережі.

Спочатку можна побачити результат обробки зображення мережею з випадковими значеннями параметрів. Вже за першу епохи ШНМ навчилась бінаризовувати зображення, однак поки виділяє не ті області, що потрібно. Більш-менш успішно виділений крововилив було помічено після дев’ятої епохи. Однак він є доволі вираженим і на вхідному зображенні. Після сотої епохи, тобто вкінці тренування можна побачити, що дана модель може знаходити і дуже слабо виражені, на перший погляд не помітні, крововиливи.

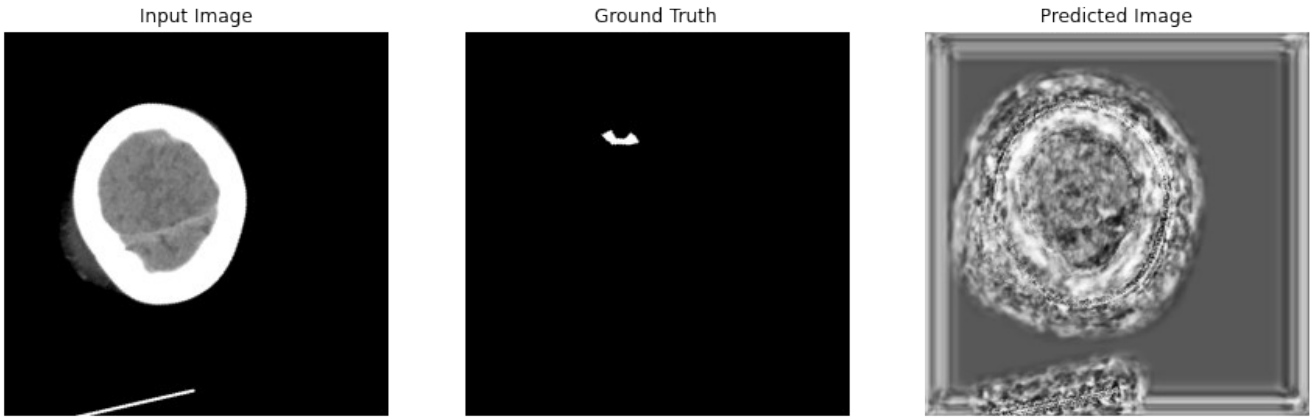


Рисунок 4.1. Вивід ШНМ до початку тренування

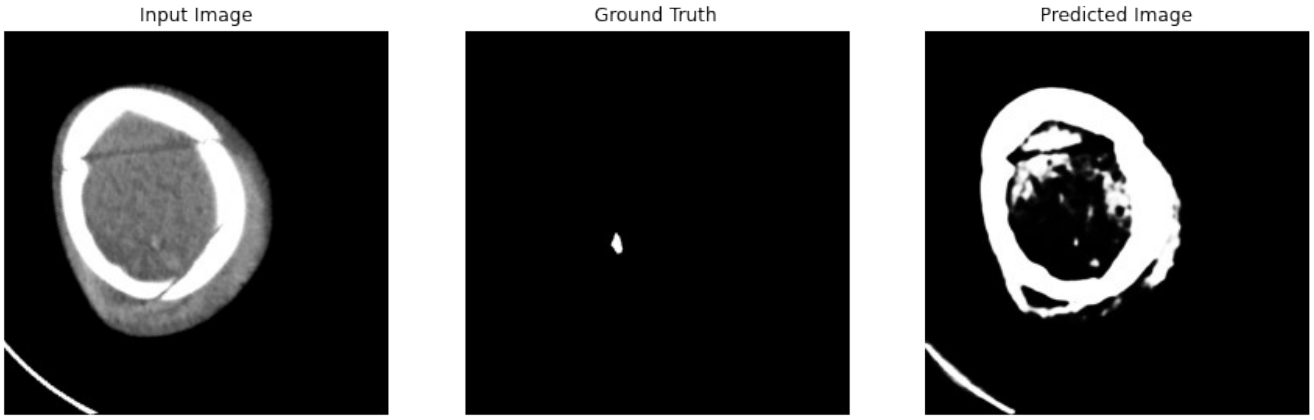


Рисунок 4.2. Вивід ШНМ після епохи 1

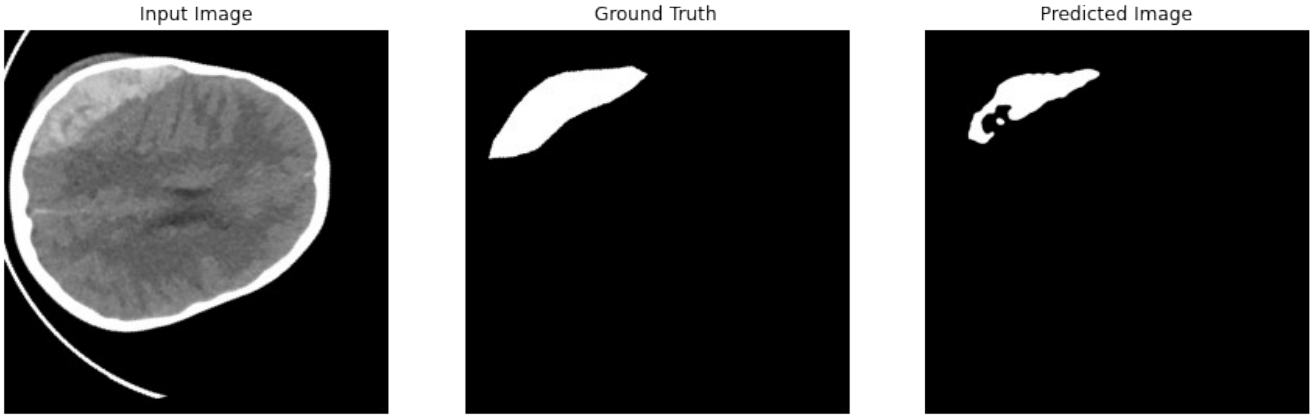


Рисунок 4.3. Вивід ШНМ після епохи 9

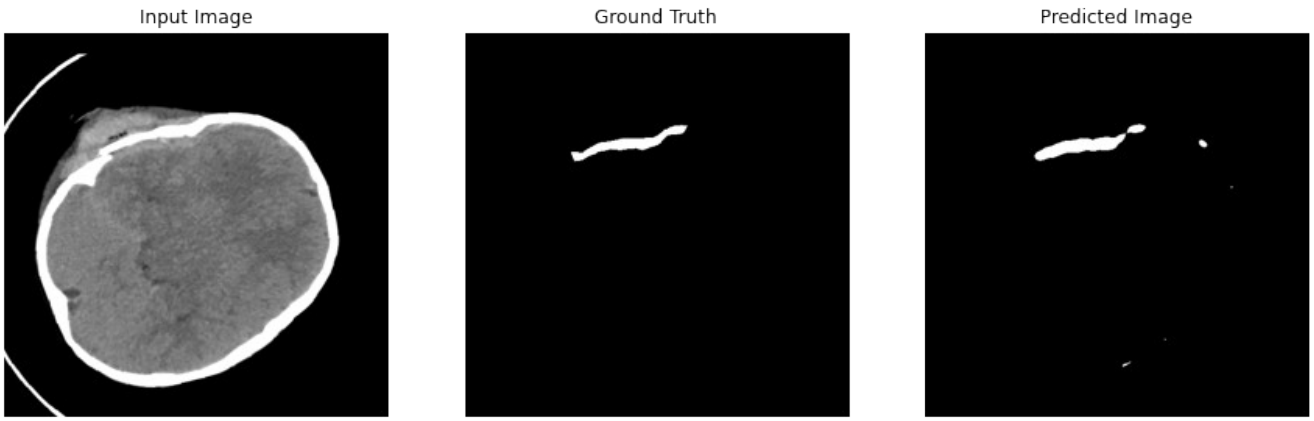


Рисунок 4.4. Вивід ШНМ після епохи 100

В таблиці 4.1 наведено результати тестування натренованої протягом 100 епох мережі. Для порівняння результатів таблиця подана у такому ж форматі, як таблиця 2.1. Але без значень чутливості та специфічності, оскільки ці метрики не

мають змісту для мережі, що, як вже згадувалось, тренувалась та тестувалась лише на зображеннях з наявними крововиливами.

Таблиця 4.1. Оцінка точності сегментації

	Коефіцієнт Жаккара	**Коефіцієнт Дайса**
Min	0.000	0.000
Max	0.780	0.876
STD	0.289	0.333
Average	0.353	0.452

Можна побачити, що максимальне та середнє значення обох метрик для дослідженої мережі більші ніж отримані в роботі [12]. Але не варто забувати, що тут розглядалась спрощена задача, сегментації крововиливів з зображень де вони гарантовано є. Проте, як вже згадувалось існують моделі ШНМ, здатні з високою точністю відсіяти зображення здорових пацієнтів. А те, що для сегментації лише серед зображень з ураженими ділянками були досягнуті кращі результати, наштовхує на ідею доцільності застосування комбінованої моделі, в якій одна частина буде відповідати за класифікацію, а інша за сегментацію. Дослідження такої моделі може стати одним з напрямків подальшого розширення теми викладеної у цій роботі.

4.2 Аналіз покращення якості зображення за допомогою похідної Рісса

Тепер розглянемо результати застосування оператора на основі похідної Рісса до нашого датасету.

На рисунку 4.5 наведено результати застосування оператора FCD з різним розміром вікна на основі похідної Рісса порядку 0.5 до зображення з датасету. Як бачимо для найменшого можливого вікна текстура мозку стає більш помітною. Зі збільшенням розміру вікна зростає контрастність, але разом з нею посилюється вплив шуму. Такий висновок також відповідає результатам, отриманим авторами оператора FCD (таблиця 1.3). Тому для подальших досліджень рекомендується використовувати невеликі маски розміром 3x3 або 5x5.

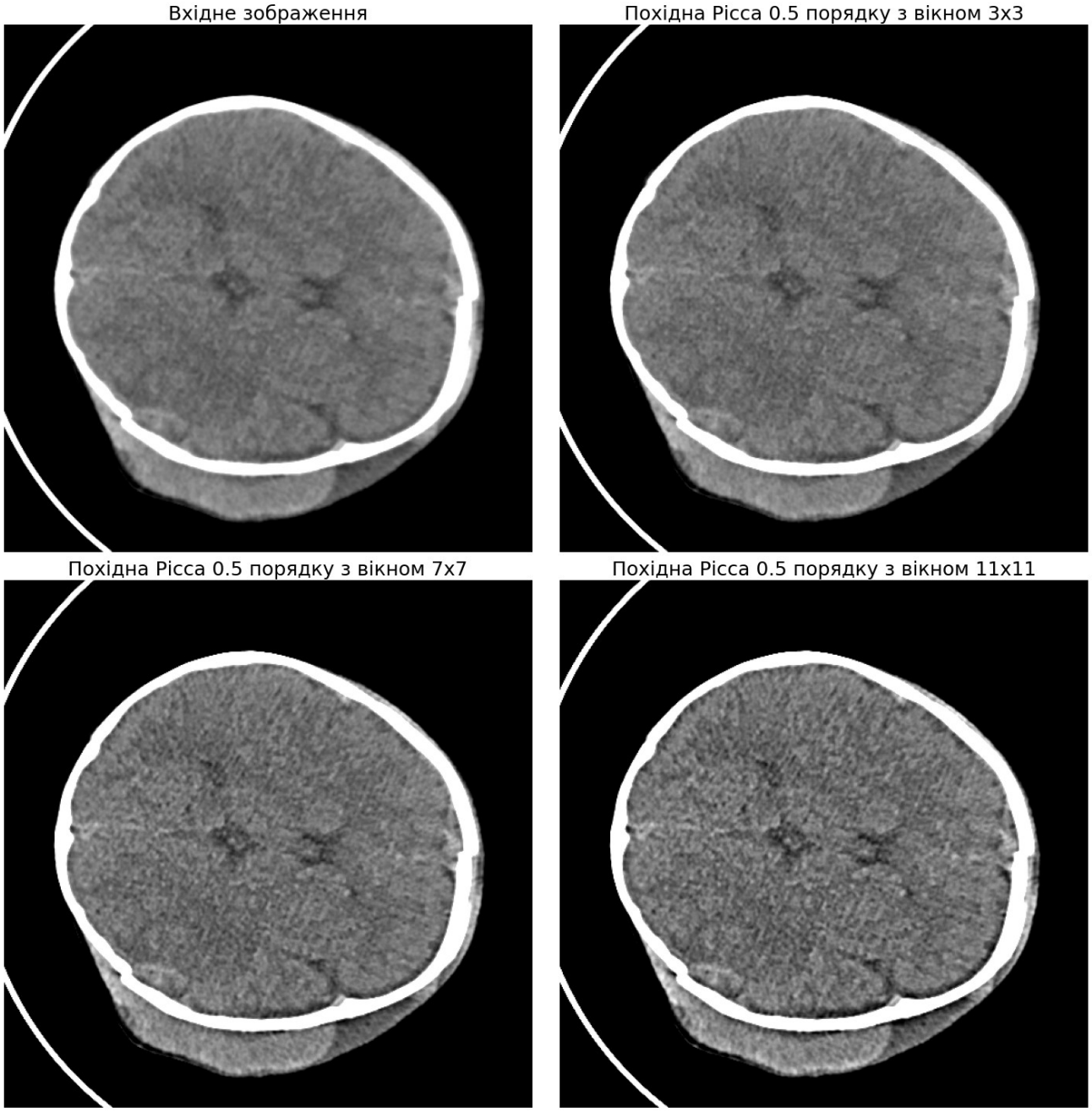


Рисунок 4.5. Результати застосування оператора FCD з різним розміром вікна на основі похідної Рісса порядку 0.5 до зображення з датасету

Окрім цього перевірялись також різні значення порядку похідної. На рисунку 4.6 можна бачити результати застосування 5x5 оператора FCD на основі похідної Рісса порядків 0.1, 0.5 та 0.9. Візуально результат для похідної порядку 0.5 виглядає найкращим, оскільки для порядку 0.1 підсилення текстури майже не помітні, а для 0.9 починають сильніше проявлятися шуми.

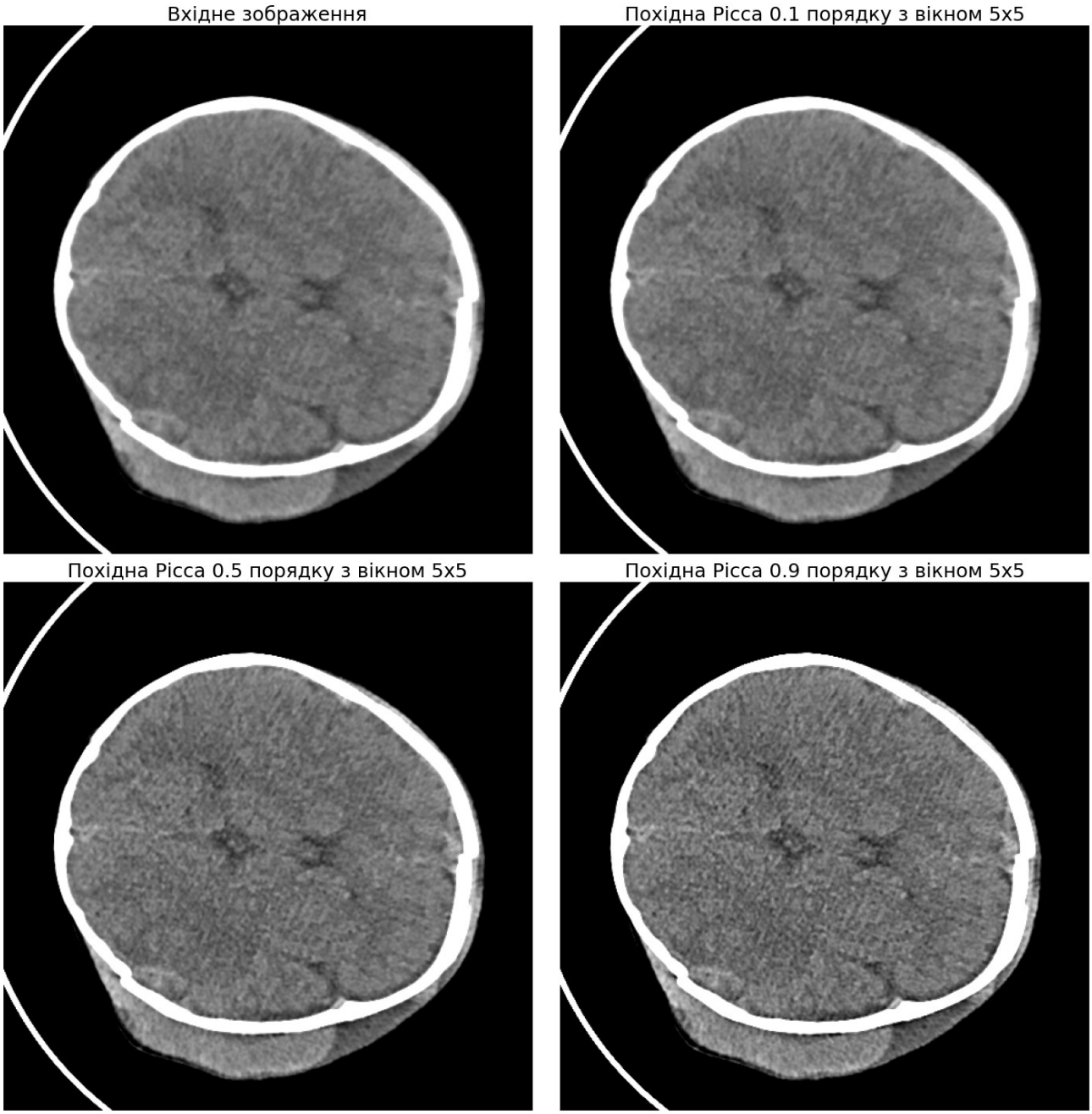


Рисунок 4.6. Результати застосування 5x5 оператора FCD на основі похідної Рісса порядків 0.1, 0.5 та 0.9

4.3 Вплив оператора FCD на якість сегментації

Окрім візуального аналізу результатів, нас цікавить чи допоможе застосування оператора FCD у вирішенні прикладної задачі сегментації внутрішньочерепних крововиливів, яку ми розглядали раніше. Для того щоб це перевірити весь тренувальний та тестовий набори даних було оброблено оператором FCD з вікном розміру 3x3 на основі похідної Рісса 0.5 порядку та

застосовано оброблені дані до тренування ШНМ U-Net, так само як це описано в пункті 3.3.

Таблиця 4.2. Оцінка точності сегментації для даних оброблених оператором FCD

	Коефіцієнт Жаккара	**Коефіцієнт Дайса**
Min	0.000	0.000
Max	0.854	0.921
STD	0.276	0.317
Average	0.398	0.507

Отримані результати наведено у таблиці 4.2. Порівнюючи ці результати з наведеними у таблиці 4.1, можемо дійти висновку, що дійсно застосування оператора на основі дробових похідних, окрім візуального покращення текстури, дозволяє ще й покращити якість сегментації.

4.4 Результати застосування fPINN та DeepONet для обчислення похідних дробового порядку

Оскільки fPINN розроблялась для розв’язання диференціальних рівнянь, а застосування цієї мережі для апроксимації операторів дробового порядку досі не розглядалось, інтерфейсом бібліотеки DeepXDE не передбачено такого використання. Модифікація алгоритму таким чином, щоб отримати шукану апроксимацію виявилась складною задачею, що потребує детальніше дослідження, що виходить за межі даної роботи.

Проте в процесі дослідження було розглянуто мережу DeepONet яка якраз і призначена для апроксимації довільних операторів. Тому її було адаптовано для вирішення задачі, що розглядається в цій роботі.

Спочатку було розглянуто невеликий датасет, наведений в документації бібліотеки DeepXDE в якості прикладу. Він складається з 150 випадково згенерованих функцій $u(x)$ для тренування і 1000 — для перевірки, кожна з яких задана в 100 точках від 0 до 1 з кроком 0.01. В прикладі цей датасет використовується для вивчення оператору інтегралу. В цій ж роботі він був модифікований для вивчення апроксимації похідної Рісса.

Для цього для кожної функції з датасету чисельно була обчислена похідна Рісса 0.5 порядку з розміром вікна 3. Таким чином ми отримали набір очікуваних даних для тренування.

В результаті навчання ШНМ з 40 нейронами в кожному шарі протягом 20000 ітерацій вдалося досягти значення функції втрат близького 10^{-4}, як на навчальній, так і на тестовій вибірках (рис. 4.7).

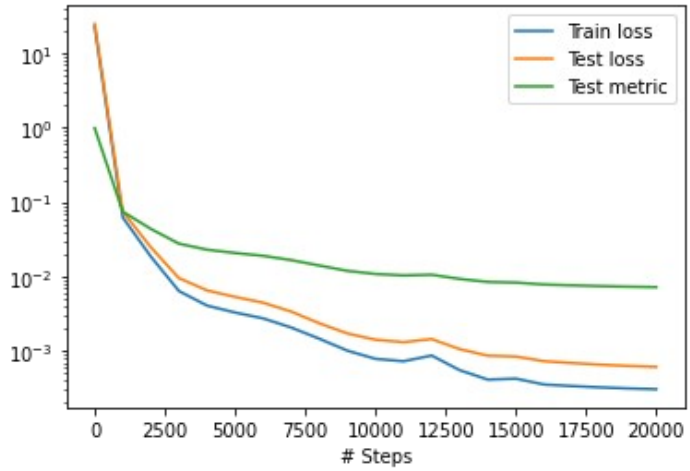


Рисунок 4.7. Значення функції втрат на кожній ітерації

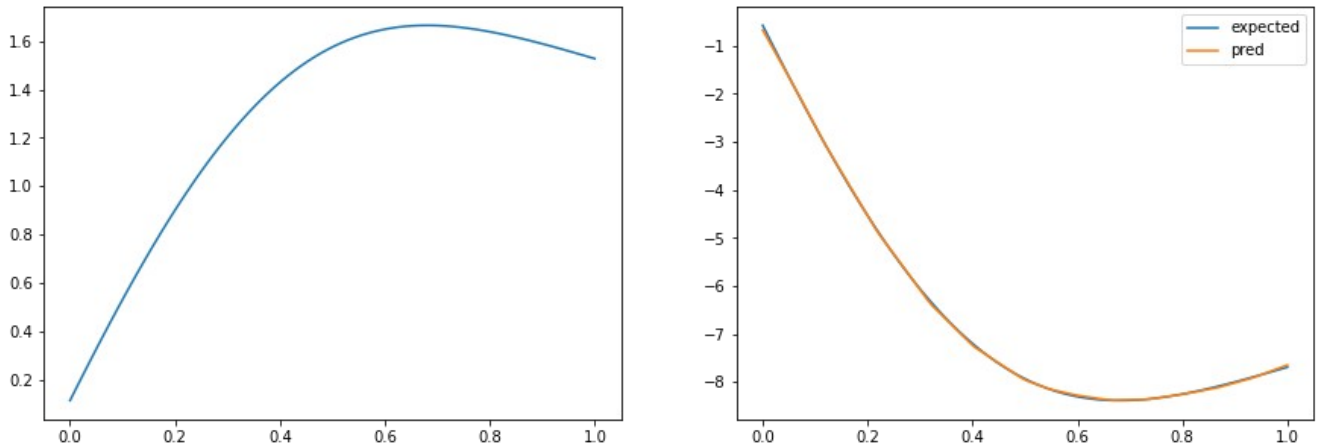


Рисунок 4.8. Результат для штучного датасету: ліворуч — вхідна функція з тестового набору; праворуч — результат апроксимації похідної Рісса за допомогою DeepONet (оранжевим) та очікуваний результат (синім)

На рисунку 4.8 бачимо приклад апроксимації похідної Рісса для функції з тестового набору. Як бачимо на графіку праворуч, отриманий результат майже повністю збігається з очікуваним.

Оскільки ця мережа продемонструвала хороший результат в апроксимації похідних Рісса для штучно згенерованих функцій, наступним кроком її було застосовано до датасету медичних зображень, що досліджується в цій роботі.

Для цього зображення були поділені на рядки які використовувались в ролі вхідних функцій для мережі DeepONet. Як і для попереднього датасету до них було застосовано оператор Рісса порядку 0.5 з розміром вікна 3, для того щоб отримати набір очікуваних даних для навчання ШНМ.

Однак такі дані виявились заскладними для мережі, тому кількість нейронів в прихованих шарах було збільшено до 512. Це дозволило отримати кращий результат, хоч він все ж і не настільки якісний як для згенерованих функцій.

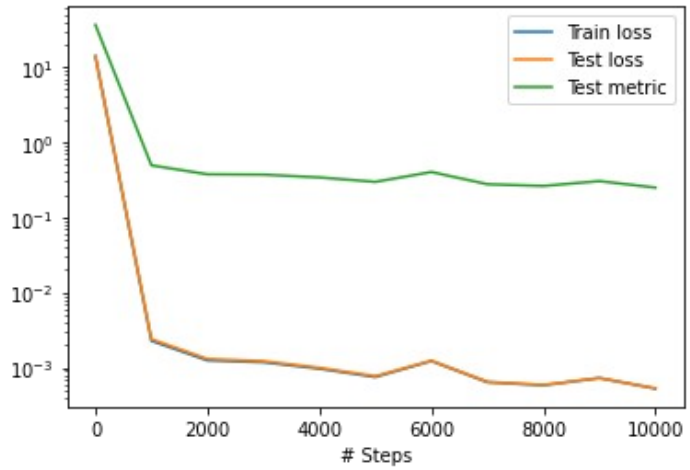


Рисунок 4.9. Значення функції втрат на кожній ітерації

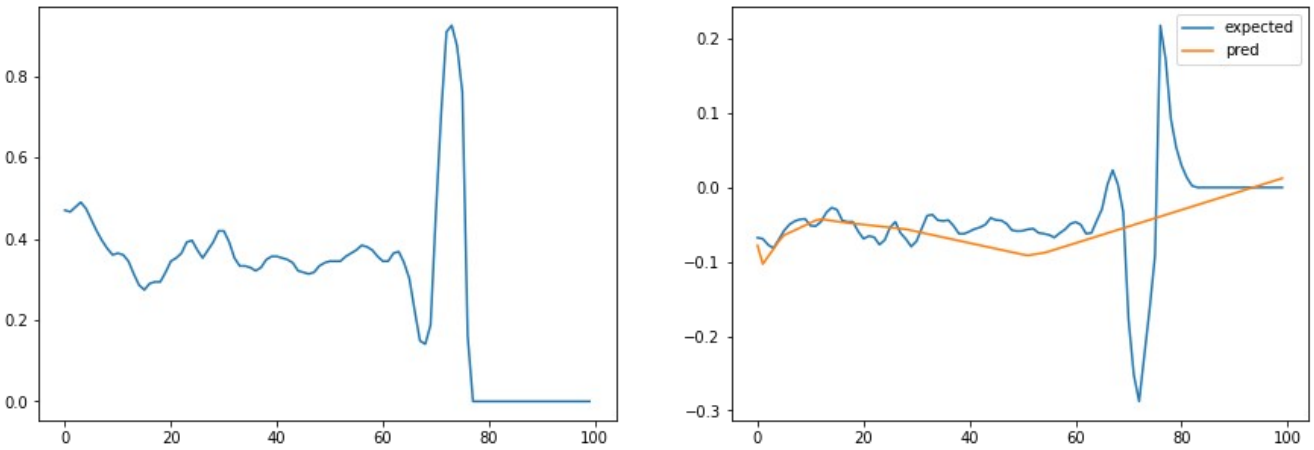


Рисунок 4.10. Результат для справжнього датасету: ліворуч — вхідна функція з тестового набору; праворуч — результат апроксимації похідної Рісса за допомогою DeepONet (оранжевим) та очікуваний результат (синім)

Як бачимо з рисунку 4.9 так само вдалося досягти значення функції втрат близьке до 10^{-4}. Однак все ж результат для апроксимації похідної для векторів з тестового набору (рис. 4.10) виглядає значно гірше ніж для штучного датасету (рис. 4.8). Отримана апроксимація виглядає як похідна усереднена біжучим вікном. Тобто загалом правильно характеризує функцію, але не передає дрібних деталей. Через це таку апроксимацію не вийде використовувати для роботи з зображеннями.

Це можна пояснити тим, що на відміну від штучних функцій, які змінювались дуже плавно, інтенсивності пікселів на зображенні можуть мати дуже різкі перепади, що ускладнює процес навчання. Але оскільки видно, що результати таки прямують до очікуваних значень, можливо застосування складніших архітектур в якості основи для DeepONet допоможе покращити точність апроксимації.

4.5 Висновки

- Загалом можна сказати, що мережа U-Net впоралась з задачею сегментації внутрішньочерепних крововиливів за КТ-сканами. Хоч і зрозуміло, що це є швидше дослідницьким прототипом, ніж готовою до застосування в такій відповідальній області як медицина системою.
- Також було розглянуто вплив оператора FCD на обраний датасет. Візуально досліджено вплив порядку похідної Рісса ($0 < \nu < 1$) та розміру вікна на якість обробленого зображення. Як оптимальні параметри обрано похідну Рісса порядку 0.5 з невеликим вікном розміру 3x3 або 5x5.
- Продемонстровано покращення якості сегментації для зображень до яких застосовувався оператор FCD порядку 0.5 з розміром вікна 3x3.
- Використання fPINN для апроксимації оператора FCD в межах інтерфейсу бібліотеки DeepXDE вимагає подальших досліджень.
- Мережу DeepONet було успішно застосовано до апроксимації похідних Рісса порядку 0.5 з розміром вікна 3 на штучному датасеті. Однак її застосування до зображень все ще вимагає певних вдосконалень.

ВИСНОВКИ

В цій роботі було проаналізовано сучасний стан справ одразу в кількох суміжних областях пов’язаних з тематикою магістерської роботи:

- сегментації медичних зображень
- покращенні якості зображення на основі фрактальних операторів дробового порядку
- дослідженні та застосуванні фізико-інформованих нейронних мереж.

В результаті огляду літератури в цих напрямках, було прийнято рішення використовувати архітектуру U-Net для вирішення задачі сегментації внутрішньочерепних крововиливів за даними КТ. А також розглянути вплив оброблення цих даних за допомогою оператора на основі похідної дробового порядку Рісса на якість сегментації та дослідити можливість застосування ШНМ fPINN та DeepONet для апроксимації цього оператора.

Для програмної реалізації було використано мову програмування Python та фреймворк PyTorch. Для реалізації навчання U-Net було використано бібліотеку MedicalTorch, а для fPINN та DeepONet — DeepXDE. Оскільки навчання ШНМ вимагає багато обчислювальних ресурсів, експерименти проводились за допомогою хмарного сервісу Google Colab, що надає безкоштовний доступ до віртуальних машин з GPU. Тому в роботі також було розглянуто особливості застосування технологій CUDA, що дозволяють розпаралелювати обчислення за допомогою GPU.

Основні результати отримані в даній роботі:

- Розроблена модель для сегментації внутрішньочерепних крововиливів показала непогані результати за метриками Джаккара та Дайса у вирішенні поставленої задачі. Звісно, про автоматичну діагностику реальних пацієнтів говорити ше рано, однак було показано, що цей напрямок має потенціал і з певними подальшими вдосконалення, можливе створення системи, яка на основі ШНМ буде допомагати лікарям звернути увагу на проблемні ділянки на КТ-скані.

- Досліджено ефект від застосування оператора FCD різних порядків та з різним розміром вікна для покращення текстури КТ-сканів. Продемонстровано, що застосування оператора FCD 0.5 порядку з вікном 3x3 дозволяє покращити якість сегментації крововиливів.
- Успішно застосовано мережу DeepONet для апроксимації оператора Рісса для штучно згенерованих функцій. При застосуванні цієї ШНМ до зображень з датасету було отримано відповідне наближення похідної Рісса, однак його точності не достатньо, для того щоб використовуватись на практиці, тому цей підхід все ще потребує вдосконалення.

Застосування fPINN до вирішення задачі, що розглядалась потребує подальшого вивчення. Використання цих мереж для апроксимації похідних дробового порядку є доволі нестандартним застосуванням цієї технології. І хоча ця апроксимація використовується всередині алгоритму, щоб її отримати потрібно модифікувати його структуру. Глибше дослідження цього може бути одним з варіантів продовженням розглянутої теми.

Іншим потенційним напрямком досліджень є вдосконалення точності апроксимації за допомогою мережі DeepONet. Наприклад можна спробувати використовувати складніші архітектури ШНМ в якості складових DeepONet.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Coagulopathy and haemorrhagic progression in traumatic brain injury: advances in mechanisms, diagnosis, and management / [Maegele M., Schochl H., Menovsky T. et al.] // Lancet Neurol. — 2017. — P.630-647.

[2] Sokolovskyy Y., Development of software and algorithms of parallel learning of artificial neural networks using CUDA technologies / Sokolovskyy Y., Manokhin D., Kaplunsky Y., Mokrytska O. // Technology Audit and Production Reserves, 5(2(61). – 2021. – P.21-25. Available from: https://doi.org/10.15587/2706-5448.2021.239784

[3] Godfrey Hounsfield [Electronic resource] / Dr Raphael Ambros, Richard Waltham et al. — 2021. — Available from: https://radiopaedia.org/articles/godfrey-hounsfield?lang=us

[4] Computed tomography [Electronic resource] / Dr Daniel J Bell, Ass. Pr. Mirjan M. Nadrljanski et al. — 2021. — Available from: https://radiopaedia.org/articles/computed-tomography

[5] FDA: Computed Tomography (CT) [Electronic resource]. — 2019. — Available from: https://www.fda.gov/radiation-emitting-products/medical-x-ray-imaging/computed-tomography-ct

[6] Hounsfield unit [Electronic resource] / Dr Daniel J Bell, Kyle Greenway er al. — 2021. — Available from: https://radiopaedia.org/articles/hounsfield-unit

[7] Манохін Д. Програмно алгоритмічне забезпечення розпаралелення навчання штучних нейронних мереж з використанням технологій CUDA / Д. Манохін // Міжнародна студентська наукова конференція з питань прикладної математики та комп'ютерних наук (МСНКПМК-2021), 6-7 травня 2021 р. – Львів:2021. – С. 21-25. Режим доступу: https://ami.lnu.edu.ua/wp-content/uploads/2021/05/Ministerstvo-osvity-i-nauky-Ukrainy.docx

[8] Deep neural networks segment neuronal membranes in electron microscopy images / [Ciresan D. C., Gambardella L. M., Giusti A. et al.] // NIPS. — 2012. — P.2852-2860.

[9] Ronnenberger O. U-Net: Convolutional Networks for Biomedical Image Segmentation / Ronnenberger O., Fischer P., Brox T. — 2015. — Available from: https://arxiv.org/pdf/1505.04597.pdf

[10] Development and Validation of Deep Learning Algorithms for Detection of Critical Findings in Head CT Scans / [Chilamkurthy S., Ghosh R., Tanamala S. et al.]. — 2018. — Available from: https://arxiv.org/pdf/1803.05854.pdf

[11] RSNA Intracranial Hemorrhage Detection / Radiological Society of North Ameriaca. — 2019.— Available from: https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection/overview

[12] Intracranial Hemorrhage Segmentation Using A Deep Convolutional Model / [Hssayeni, M. D., Croock, M. S., Salman, A. D. et al.]. — 2020. Available from: https://www.mdpi.com/2306-5729/5/1/14

[13] Hssayeni M. Computed Tomography Images for Intracranial Hemorrhage Detection and Segmentation (version 1.3.1) [Electronic resource]. *PhysioNet*. — 2020. — Available from: https://doi.org/10.13026/4nae-zg36

[14] PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals [Electronic resource] / [Goldberger A., Amaral L., Glass L. et al.]. — 2000. — Available from: https://physionet.org/

[15] The use of a Riesz fractional differential-based approach for texture enhancement in image processing / [Yu Q., Liu F., Turner I. et al.] // ANZIAM Journal. — 2012. — Vol.54. — P590-607. — Available from: https://doi.org/10.21914/anziamj.v54i0.6325

[16] Hamid J., Texture Enhancement for Medical Images Based on Fractional Differential Masks / J. Hamid, I. Rabha // Discrete Dynamics in Nature and Society. — 2013. — Available from: http://dx.doi.org/10.1155/2013/618536.

[17] Pu Y.-F. Fractional Differential Mask: A Fractional Differential-Based Approach for Multiscale Texture Enhancement / Y.-F. Pu, J.-L. Zhou, X. Yuan // IEEE Transactions on Image Processing. — 2010. — Vol.19. — P.491-511. — Available from: http://dx.doi.org/10.1109/TIP.2009.2035980

[18] Shao Ch. A Quantum Model for Multilayer Perceptron / Ch. Shao. — 2018. — Available from: https://www.researchgate.net/publication/327392288_A_Quantum_Model_for_Multilayer_Perceptron

[19] Jadon Sh. Introduction to different activation functions for deep learning / Sh. Jadon // Medium. — 2018. — Vol.16. — Available from: https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

[20] Ian Goodfellow Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — MIT Press, 2016. — 781pp.

[21] Cybenko G. Approximation by superpositions of a sigmoidal function / G. Cybenko // Mathematics of Control, Signals, and Systems. — 1989. — Vol.2(4). — P.303-314. — Available from: https://doi.org/10.1007/BF02551274

[22] Li F.-F. Lecture 11: Detection and Segmentation / F.-F. Li, J. Johnson, S. Yeung. — 2017. — Available from: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[23] Pang G. fPINNs: Fractional Physics-Informed Neural Networks / G. Pang, L. Lu, G. Karniadakis // SIAM Journal on Scientific Computing. — 2019. — Vol.41. — P.2603-2626. — Available from: https://doi.org/10.1137/18M1229845

[24] Chen T. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems / T. Chen and H. Chen // IEEE Transactions on Neural Networks. — 1995. — Vol.6(4). — P.911–917.

[25] Lu L. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators / L. Lu, P. Jin, G. Karniadakis et al. // Nature Machine Intelligence. — 2021. — Vol.3. — P.218-229. — Available from: https://doi.org/10.1038%2Fs42256-021-00302-5

[26] Chainer: a Next-Generation Open SourceFramework for Deep Learning / [Seiya Tokui, Kenta Oono, Shohei Hido, et al.]. — 2015.

[27] MedicalTorch: Release v0.2 [Electronic resource] / Christian S. Perone, Elvis Saravia, Pedro Lemos Ballester et al. — 2018. — Available from: http://doi.org/10.5281/zenodo.1495335

[28] PyTorch Documentation [Electronic resource]. — 2022. — Available from: https://pytorch.org/docs/stable/index.html

[29] Colaboratory Frequently Asked Questions [Electronic resource]. — 2022. — Available from: https://research.google.com/colaboratory/faq.html

[30] DeepXDE Documentation [Electronic resource]. — 2022. — Available from: https://deepxde.readthedocs.io/en/latest/