КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«ДОСЛІДЖЕННЯ ВПЛИВУ ШУМІВ НА ЗОБРАЖЕННЯ МРТ ГОЛОВНОГО МОЗКУ»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Рябчук М. С. _____

Науковий керівник:
доктор фізико-математичних наук, доцент
Сугакова О.В. _____
асистент
Погорєлов Р. В. _____

Рецензент:
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Дипломна робота за об’ємом складає 36 сторінок, містить 3 рисунки, використано 10 літературних джерел.

Розглянуто: алгоритми для роботи з МРТ знімками мозку, способи зашумлення та очищення знімків.

Зроблено: реалізовано програму, можливості якої допомагають розглянути вигляд знімку мозку при різній зашумленості з функцією завантаження власних знімків для тестування.

Ключові слова: PYTHON, СТАТИСТИКА, ПЕРЕТВОРЕННЯ ФУР’Є, ТЕХНОЛОГІЇ ДЛЯ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ, CUDA.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

МРТ - магнітно-резонансна томографія

Воксель - аналог пікселя у 3D просторі.

ML - Machine Learning(Машинне навчання).

Аліасинг - ефект, що призводить до накладання, або нечіткості різних безперервних сигналів при їхній дискретизації.

CNN - Convolutional neural network(згорткова нейронна мережа).

NumPy - Numeric Python.

DICOM - Digital Imaging and Communications in Medicine.

PACS - picture and archiving communications systems.

SciPy - Scientific Python.

ВСТУП

Магнітно-резонансна томографія (МРТ) є найпопулярнішим методом візуалізації в медицині для отримання томограм внутрішніх органів людини. Даний метод заснований на принципах ядерного магнітного резонансу.

На МРТ знімках є шум, що впливає на якість їх контрастності і різкості, також кластерні пікселі на знімках в сірій градації дозволяють дослідникам і штучним пристроям розрізняти патологічні структури, а ступінь кластеризації визначає рівень якісних ознак зображення. Шум на МРТ зображеннях є випадковим і призводить до руйнування кластеризованих пікселів та розмивання країв. У свою чергу, розмивання країв знижує контраст між різними структурами, що ускладнює використання МРТ-знімків для діагностики захворювань.

Створена програма матиме змогу накладати шуми на знімок МРТ для подальшого аналізу, цим самим спростивши роботу користувачів.

Постановка задачі: створення програми для дослідження впливу шумів на зображення МРТ головного мозку.

РОЗДІЛ 1. АНАЛІЗ ШУМІВ НА ЗНІМКАХ МРТ

1.1 ПОНЯТТЯ МРТ ЗОБРАЖЕНЬ, ВПЛИВ ШУМІВ НА ЗОБРАЖЕННЯ

Сьогодні МРТ - один з основних методів дослідження та лікування людського мозку. Але, часто, в цьому метод, виявляється неточність отриманих результатів. Причиною цього являється недостатньо чітке зображення МРТ. Проблема полягає у тому, що основною характеристикою ураженої тканини є її внутрішня структура, тобто, густина ураженої та не ураженої тканини може бути дуже схожою. Унаслідок цього відповідні ділянки на МРТ-знімку будуть досить подібними і легка розмитість зображення може завадити виявленню проблеми. Наявний шум впливає на якість контрастності і різкості томографічних і рентгенівських зображень, викликає руйнування кластеризованих пікселів і розмивання країв. В свою чергу, ерозія країв знижує контраст між різними структурами, що ускладнює використання зображень для діагностики захворювань.

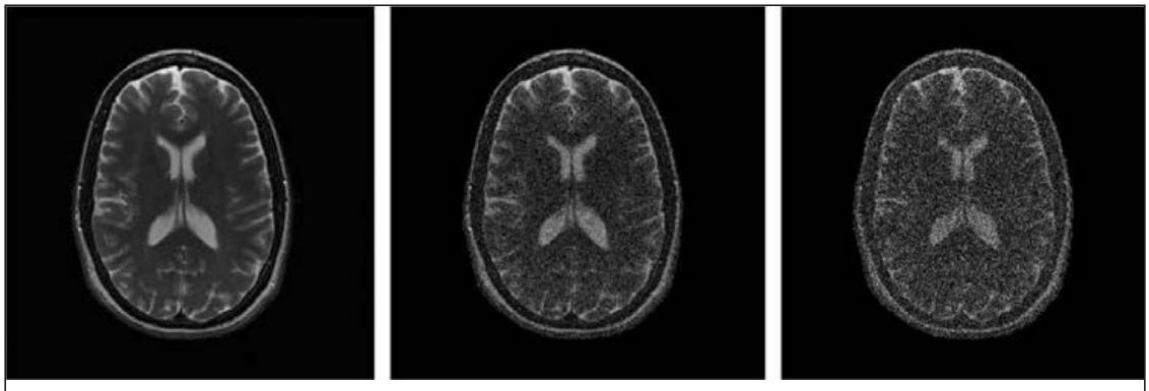

Рис. 1
а - МРТ знімок без шумів,
б, в - МРТ знімки з різною інтенсивністю накладання шумів

1.2 ЯКІСТЬ МРТ ЗОБРАЖЕНЬ, ЩО НА НИХ ВПЛИВАЄ

Якість томограми залежить від її просторової роздільної здатності. Просторова роздільна здатність визначає якість зображення з точки зору того, чи виглядає отримане зображення добре деталізованим, а краї чіткими і не

розмитими. Хоча просторова роздільна здатність томограми залежить від розміру вокселя, водночас вона визначається кількістю кроків кодування частоти та кроків кодування фази, цю сукупність параметрів також називають розміром матриці.

Не лише розмір матриці може впливати на просторову роздільну здатність зображення, але й товщина зрізу та поле зору. Роблячи одні параметри постійними і варіюючи інші, можна отримати томограми з високою просторовою роздільною здатністю за рахунок іншого параметра (наприклад, часу отримання).

1.3 МОДЕЛЬ ОЦІНКИ МРТ ЗОБРАЖЕНЬ

Для оцінки шуму МР-зображення можна використовувати непараметричні статистичні методи. Це забезпечує перевагу перед оцінками параметрів, які є обмежувальними припущеннями щодо форми функції. Шум на МРТ-зображеннях мозку може мати різні типи та джерела, наприклад, електроніка МРТ, рухи пацієнта, залізні конструкції тощо.

Деякі типи шумів, які можуть бути присутні на МРТ знімках мозку, включають:

Гаусовий шум: Це стандартний шум будь-якого електронного пристрою. Він має нормальний розподіл і зазвичай рівномірний по всьому зображенню.

Шум руху: Викликаний рухом пацієнта або дрейфом МРТ. На зображенні видно розмиті або рухомі сліди.

Шум пульсації: викликаний коливаннями кровотоку та може бути помічений у вигляді нерівностей країв кровоносних судин.

Шум артефакту заліза: Викликається наявністю металевих предметів в тілі пацієнта. У місцях, де присутні металеві предмети, цей шум можна розглядати як спотворення або втрату контрасту.

Шум на МРТ знімках мозку може суттєво погіршувати якість зображення.

Тому, щоб позбутися від шумів, можна використовувати наступні методи:

1. *Фільтри шуму:* Шумові фільтри можуть усунути частину шуму на МРТ-зображеннях головного мозку без втрати якості зображення. Розроблено різні типи шумових фільтрів, такі як медіанний фільтр, фільтр Гаусса, анізотропний дифузійний фільтр тощо.
2. *Алгоритми зниження шуму:* Існує багато алгоритмів зменшення шуму для МРТ-зображень мозку, які можуть допомогти усунути шум із зображення. Наприклад, метод шумозаглушення з використанням нелінійної фільтрації Гауса (NL-GF), алгоритм зваженої фільтрації Гауса (WGF) тощо.
3. *Калібрування та нормалізація положення*: калібрування та нормалізація допомагають зменшити шум на МРТ-зображеннях мозку. Нормалізація дивергентного поля (DNP) — це метод, який може зменшити вплив шуму.
4. *Різні методи прискорення сканування*: використання різних методів прискорення сканування, таких як швидка інверсія часу (FTI) і розріджене сканування, може допомогти зменшити шум у зображеннях МРТ мозку. Ці методи дозволяють швидше сканувати зображення, зменшуючи вплив шуму.
5. *Ітеративні методи відновлення зображення*: ітераційні методи реконструкції зображення можуть допомогти усунути шум із МРТ-зображень мозку. Ці методи базуються на ітераційних алгоритмах реконструкції зображень за допомогою статистичних методів.

6. *Ітеративні методи відновлення зображення:* ітеративні методи відновлення зображення можуть допомогти видалити шум на МРТ знімках мозку. Ці методи засновані на ітераційних алгоритмах, які використовуються для відновлення зображення з використанням статистичних методів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ РОБОТИ ДОДАТКУ

Основна мета роботи з типами шумів на знімках МРТ полягає в тому, щоб швидко та точно очищувати зображення чи штучно накладати на нормальний знімок шуми, для подальшого їх перегляду.

Можливості роботи планованої програми:
Простий інтерфейс для користувача створений з використанням технології Qt.
Автоматичне перетворення Фур'є для швидкої візуалізації змін в програмі.
Можливість додати/завантажити власні зображення для аналізу після проходження k-space.

Після запуску програми відкривається доступ до готового зображення, на якому можна змінювати шумові діапазони з допомогою функцій програми.
Є можливість інтерактивно змінювати параметри проходження k-простору та аналізувати отримані результати.
Також доступними є функції завантаження власного знімку для аналізу під час використання програми.

РОЗДІЛ 3. ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО РІШЕННЯ

3.1 ПРОБЛЕМА РЕКОНСТРУКЦІЇ ЗОБРАЖЕНЬ

Магнітно-резонансна томографія широко використовується в медичній діагностиці та є еталонним стандартом у багатьох областях. Водночас він має очевидний недолік: збір даних спочатку відбувається повільно. Сигнали МРТ генеруються ядрами водню за рахунок їх взаємодії із зовнішнім електромагнітним полем. Однак сканери МРТ не можуть безпосередньо вимірювати просторово корельовані сигнали (тобто зображення). Натомість просторова кореляція кодується в частоті та фазі МРТ-сигналу. Цей процес кодування є послідовним за своєю природою, що робить збір даних трудомістким. Нарешті, отримується просторова частотна карта, представлена як k-простір. У найпростішому випадку можна використати зворотне перетворення Фур'є, щоб реконструювати дані k-простору в зображення для клінічної інтерпретації.

Через послідовний характер сканування МРТ час збору даних приблизно пропорційний кількості зразків, зібраних у k-просторі. Тому бажано зібрати якомога менше проб. Однак, якщо частоту дискретизації зменшити нижче бажаного критерію Найквіста, на зображенні з'являться артефакти накладення.

У загальному вигляді відновлення зображення можна сформулювати у вигляді такої оберненої задачі:

$$y = Ax + \epsilon \qquad (1)$$

Де y – виміряні дані k - простору, A – системна матриця, x – це образ і ϵ – є випадковим шумовим членом. Коли дані в k -просторі мають недостатню дискретизацію і спотворені шумом, зворотне завдання рівняння Рівняння (1)

некоректно: рішення може існувати, можуть існувати нескінченні рішення, може бути нестійким стосовно помилок виміру. У результаті пряма інверсія взагалі неможлива. Задача є коректною за Адамаром, якщо: її розв'язок існує; є єдиним; є стійким (тобто неперервно залежить від вихідних даних задачі), або «нестрого» кажучи, малим збуренням вихідних даних відповідають малі відхилення в розв'язку. Натомість оптимальне рішення в сенсі методу найменших квадратів можна отримати, переформулювавши завдання у вигляді наступної мінімізації:

$$\hat{x} = \frac{1}{2} ||Ax - y|| \frac{2}{2} \qquad (2)$$

За останні кілька десятиліть багато зусиль було спрямовано на реконструкцію зображення з недостатньо дискретизованого k-простору. Два поширені методи виділяються своєю важливістю та заслуговують на короткий огляд тут, а саме паралельне зображення та стиснене зондування. Це дозволяє значно скоротити час збору даних при збереженні якості зображення.

Методи паралельного зображення використовують багатоканальні масиви приймачів для компенсації недостатньої дискретизації k-простору. Це можливо тому, що приймальні котушки демонструють просторово змінний зворотний зв'язок, який можна використовувати для розгортання накладених зображень або оцінки відсутніх зразків

k-простір.

Методи паралельної візуалізації, такі як SENSE (кодування чутливості) і GRAPPA (загальне автоматичне калібрування для частково паралельного збору даних), досягли великого успіху і регулярно використовуються в клінічних умовах. Однак збільшення коефіцієнта прискорення призводить до втрати співвідношення сигнал/шум (SNR), що фактично обмежує досяжне прискорення.

Стиснене зондування (CS) дозволяє реконструювати субдискретизовані сигнали за умови, що реконструйований сигнал розріджений у деяких регіонах. Сигнал є розрідженим, якщо він містить мало ненульових елементів порівняно з його розміром. Зображення МРТ зазвичай не стоншені. Однак, як і більшість природних зображень, вони містять багато надмірностей і мають розріджені представлення в інших областях, таких як домени кінцевих різниць або вейвлети. Проблема оптимізації рівняння може включати очікування, що рішення буде розрідженим у деяких областях.(2) як член регуляризації:

$$\hat{x} = \frac{1}{2}\left|\left|Ax - y\right|\right|\frac{2}{2} + \lambda \left|\left|Dx\right|\right|_1 \qquad (3)$$

де □ – є розріджуючим перетворенням, що відображає надмірне зображення в його розріджене представлення, $||.||_1$ – це L_1 норма і λ є параметром регуляризації. Перший член цього рівняння служить для забезпечення того, щоб рішення узгоджувалося з виміряними даними, тоді як другий член сприяв рішенням, які є розрідженими в галузі перетворення. Параметр урівноважує обидва терміни і може бути налаштований для оптимізації якості зображення.

Стиснене зондування засноване не тільки на очікуванні, що правильне рішення рівняння (3) є розрідженим в галузі перетворення, але також і те, що рішення з псевдонімами не є розрідженими. Це призводить до іншої вимоги: щоб артефакт накладання спектрів був некогерентним, тобто щоб він нагадував шум. Цього можна досягти в МРТ, використовуючи нерегулярні чи псевдовипадкові шаблони вибірки у межах апаратних обмежень.

Стиснене зондування можна легко поєднати з паралельною візуалізацією для забезпечення високошвидкісного прискорення. Однак клінічний переклад ускладнюється кількома факторами. По-перше, нелінійна ітераційна

реконструкція часто займає надто багато часу або вимагає обчислювальних ресурсів, які зараз недоступні для більшості клінічних служб. По-друге, зображення, як повідомляється, виглядають неприродно та блоково. Нарешті, налаштування терміну регуляризації є емпіричним процесом, зазвичай залежним від застосування, і може відрізнятися від пацієнта до пацієнта.

Згідно з теоремою Найквіста-Шеннона, загалом неможливо уникнути спектрального перекривання та деградації сигналу при дискретизації нижче частоти Найквіста. Методи реконструкції для прискореної МРТ базуються на тій чи іншій формі попередньої інформації або додаткових обмеженнях реконструйованого сигналу.

Однак попередні дані, які використовуються в паралельному зображенні та стисненому зондуванні, часто є грубими, і більш репрезентативні попередні дані можуть покращити існуючі методи реконструкції. Однак побудувати такі пріори вручну складно. ШІ, з іншого боку, добре виявляє закономірності в даних. Таким чином, це ідеальний інструмент для включення знань, отриманих з історичних даних МРТ, у реконструкцію зображення.

3.2 КОНТРОЛЬОВАНЕ МАШИННЕ НАВЧАННЯ

Відомо, що машинне навчання складається з двох категорій: навчання з учителем або без нього. Контрольоване навчання використовує відомі достовірні дані для вивчення порівняння між двома точками даних, тоді як неконтрольоване навчання використовує шаблони, отримані з вибірки без спостережуваних вихідних даних. Сьогодні в більшості програм реконструкції МРТ використовується контрольоване машинне навчання, але неконтрольовані методи все ще активно досліджуються. Багато підходів до машинного навчання

використовуються для реконструкції зображень МРТ (як під наглядом, так і без нагляду), ці підходи включають методи, які працюють у просторі зображень:

- Що працюють у k–просторі
- Що працюють у різних галузях
- Що вивчають пряме відображення з k-простору в простір зображень та розгорнуті методи оптимізації

3.3 ПАРАЛЕЛЬНІ МЕТОДИ ПРОГРАМУВАННЯ МОВОЮ PYTHON

Ключовою особливістю сучасних графічних відеоадаптерів, що використовують графічні процесори (GPU), є наявність набору потокових мультипроцесорів (SM), які раніше використовувалися лише для алгоритмів і завдань, пов’язаних з обробкою графічних зображень. Методи програмного забезпечення, які програмісти використовують для створення програм у цьому напрямку, використовують пам’ять відеоадаптера для розміщення структур даних, таких як текстури, буфери, і визначення конвеєра обробки, причому кожен етап відповідає за свою конкретну операцію: растеризація, інтерполяція, змішування, теселяція тощо . Обчислення загального призначення на графічних процесорах (GPGPU) базуються на графічних процесорах, які використовують масову паралельну роботу для обробки даних за допомогою алгоритмів загального призначення. Потокові процесори на GPU мають простішу структуру, ніж вузли CPU. Тобто такі вузли менш загального призначення і виконують менше функцій, ніж процесорні вузли.Однак завдяки їх великій кількості це може значно прискорити певний комплекс завдань. Алгоритми, які підтримують концепцію паралелізму даних, досягають найкращого прискорення. Таким чином, GPGPU зазвичай використовується в таких сферах: обробка зображень (зменшення, гістограма, швидке перетворення Фур’є, таблиця сумарної площі); обробка

відеоданих (перекодування, цифрові ефекти, аналіз); лінійна алгебра; моделювання (технічне, фінансове, академічне, деякі бази даних) тощо.

Кожен програмний інтерфейс, який використовується для створення програм GPGPU, має власний рівень абстракції щодо апаратної реалізації цільової архітектури. Тому NVidia CUDA позиціонується компанією-розробником як програмна модель і платформа для загальних паралельних обчислень. Спочатку для цієї технології було створено середовище розробки CUDA SDK, яке базується на мові програмування C з деякими розширеннями. Сьогодні архітектура CUDA набула величезної популярності в обчислювальних системах, які обчислюють великі обсяги даних. Тому рекомендується розширити список мов програмування, які можуть підтримувати дану модель програми. В даний час така можливість є в мові програмування Python, яка широко використовується для програмування числових методів і набирає популярності завдяки відносно простому синтаксису і зручному для читання коду. Python — це об’єктно-орієнтована мова високого рівня, яка підтримує множинне успадкування, переписує інфіксні оператори, і можна переписати як ліві, так і праві операнди. Python має механізми для обробки винятків і їх перехоплення; таким чином, програмісти можуть будувати коректну обробку помилок і створювати надійні програми. Вбудовані механізми самоаналізу дозволяють опитувати інтерфейси об’єктів під час виконання програми. Наприклад, ви можете дізнатися кількість і назви параметрів функції. Python працює майже на всіх відомих платформах, від мобільних до мейнфреймів. Для мови Python добре відомий такий інтерфейс програмування: PyCUDA, який дозволяє виконувати паралельні програми на ядрах CUDA.

3.4 ВИКОРИСТАНІ БІБЛІОТЕКИ

1. CuPy
2. Pydicom
3. Pillow

4. PyQt5
5. SciPy

CuPy — це бібліотека масивів із відкритим кодом для GPU-прискорених обчислень за допомогою Python. CuPy використовує бібліотеки CUDA Toolkit, включаючи cuBLAS, cuRAND, cuSOLVER, cuSPARSE, cuFFT, cuDNN і NCCL, щоб повністю використовувати архітектуру GPU. CuPy прискорює деякі операції більш ніж у 100 разів.

Інтерфейс CuPy сумісний із NumPy та SciPy, у більшості випадків його можна використовувати як заміну.

Особливості бібліотеки:

- Підтримка N-мірних масивів із швидкою векторизацією та індексацією.
- Бібліотека містить безліч математичних функцій, генератори випадкових чисел, операції лінійної алгебри, перетворення Фур'є та багато іншого.
- Підтримується широкий спектр обчислювальних платформ, а також ведеться робота з розподіленими бібліотеками, бібліотеками з графічним процесором (GPU) і розрідженими масивами (sparse arrays).
- Бібліотека має відкритий вихідний код під ліцензією BSD, яка розробляється та підтримується на GitHub.

DICOM (Digital Imaging and Communications in Medicine) – з моменту свого створення на початку 1980-х років він був основним засобом для зберігання та передачі наборів даних медичних зображень. Це набагато більше, ніж формат файлу або протокол передачі, DICOM визначає обидва формати, стандарти якості та протоколи передачі для клінічного використання.

DICOM — це міжнародний стандарт, і він реалізований майже в усіх радіологічних/кардіологічних/радіотерапевтичних пристроях. Це найпоширеніший у світі стандарт обміну повідомленнями в галузі охорони здоров'я, який містить мільярди зображень.

Використання Pydicom:

Хоча DICOM здійснив революцію в медичній візуалізації з моменту його першої публікації в 1993 році, очевидно, що цей цифровий робочий процес має бути інтегрований у зростаючий простір сучасних технологій. Важливо, щоб інструменти, які використовуються для передачі, зберігання та роботи з цими медичними зображеннями, були сучасними. На даний момент ці технології включають:

- модульні робочі процеси на основі контейнерів
- хмарні ресурси
- інтеграція зображень і метаданих у ці ресурси
- простий запит, пошук і переміщення даних для клінічного та дослідницького використання

Хоча PACS (picture and archiving communications systems) також відіграють важливу роль у безпечній передачі медичних зображень, зазвичай буває, що одного PACS може бути недостатньо для підтримки клінічної та дослідницької частини лікарні. Стратегія багатьох лікарень (зазвичай) полягає в тому, щоб найняти постачальника для налаштування та обслуговування цього PACS. Ця традиційна практика стає дорогою, а простір між користувачами (лікарями та дослідниками) та програмним забезпеченням, що керує інструментами, стає більшим. У цій звичайній реальності користувачі не мають права створювати інструменти, необхідні для розробки програмного забезпечення та інструментів для проведення досліджень і клінічної практики.

Pydicom полегшує науковцям з обробки даних, лікарям, аспірантам та іншим зацікавленим розробникам створення програмного забезпечення, яке використовує набори даних і протокол DICOM, сприяє легкій розробці мовою Python.

Pillow - бібліотека, що додає Python досить великі можливості обробки зображень, ідентифікує та читає велику кількість форматів. Pillow ідеально підходить для програм пакетної обробки зображень. Її можна використовувати для створення ескізів, перетворення між форматами файлів, друку зображень тощо.

Бібліотека Pillow містить основні функції обробки зображень:

- містить ряд готових операцій для зображення L і RGB (автоконтраст, обрізка, масштабування, обертання, довільні афінні перетворення і т.д.)
- містить набір зумовлених фільтрів покращення зображення (BLUR, DETAIL, SHARPEN і т.д.).
- при збереженні файлів JPEG можна використовувати параметри якості зображення, еквівалентні параметрам Photoshop.
- вміє робити копіювання вмісту екрана (скріншот екрану).
- підтримує фільтрацію за допомогою набору вбудованих ядер згортки та перетворення колірного простору.
- метод гістограми дозволяє отримати деяку статистику із зображення, яку можна використовувати для автоматичного покращення зображення та для статистичного аналізу.
- вміє завантажувати шрифти TrueType або OpenType з подальшим створенням об'єкта шрифту із заданим розміром для виконання написів на зображенні.
- має просту 2D-графіку для створення нових зображень, ретушування існуючих, а також створення графіки на льоту для використання в Інтернеті.

PyQt – це бібліотека, яка дозволяє використовувати фреймворк Qt GUI (GUI – це графічний інтерфейс користувача) у Python. Сам Qt, як відомо, написано на C++. Використовуючи його в Python, можна створювати програми набагато швидше.

PyQt5 це остання, п'ята версія Qt. Ще можна знайти в інтернеті згадку про PyQt4, але ця версія застаріла і більше не підтримується.

SciPy (Scientific Python) – бібліотека, яка являє собою розширення бібліотеки NumPy, використовується для виконання інженерних, статистичних та наукових розрахунків,в тому числі і для аналізування даних та будування графіків.

Використання SciPy:

SciPy використовується для виконання складних операцій: розрахунку алгебраїчних функцій або різних числових алгоритмів.

У SciPy містить велику кількість функцій та методів для вирішення завдань лінійної алгебри, розрахунку перетворення Фур'є, обробки сигналів та зображень та інші. Вони реалізовані у повному своєму обсязі, що дає більше можливостей для роботи.

SciPy концепція масивів досить функціональна і не має обмежень на однорідність, а отже масиви SciPy можуть бути як гомогенними, так і гетерогенними.

З мінусів: бібліотека SciPy написана мовою Python, через що має нижчу швидкість виконання, але низька швидкість компенсується функціональністю.

РОЗДІЛ 4. РЕЗУЛЬТАТИ ТА ЇХ АНАЛІЗ

Додаток А: містить клас знімку МРТ. Поля класу містять знімок у k-просторі та звичайне зображення. Також виконано розпаралелення за допомогою Cupy.

Конструктор приймає або звичайне зображення або перетворене у k-простір. Якщо передане звичайне зображення, то конструктор перетворює його у k-простір та зберігає у полях класу та навпаки.

Клас також має властивості (properties) для отримання підготовленого зображення для відображення на екрані.

Зображення у K-просторі трансформується наступним чином: для кожної точки рахується абсолютне значення, далі це значення масштабується у логарифмічну шкалу; отримане зображення нормалізується (щоб значення пікселів були в межах від 0 до 255).

Описане вище можна виразити формулою:

$$□□□□□(0{,}1□(1 + |□| * 10^{-3}), 255)$$

У звичайному зображенні лише збільшується контраст, тобто значення кожного пікселю рахується за формулою $\frac{□}{□□□(□)* 250}$, де □□□(□)- максимальне значення серед усіх пікселів.

Додаток Б: розпаралелено за допомогою Cupy. Містить клас, що накладає шум на зображення у K-просторі. Використовує альфа-стійкий процес для генерації шуму. Конструктор приймає параметри альфа стійкого процесу.

Накладання шуму відбувається наступним чином:

1. Генерується комплексний двовимірний масив з шумом за допомогою функції levy_stable.rvs(...)

2. Згенерований шум додається до зображення у K-просторі

Інтерфейс користувача зроблений за допомогою бібліотеки pyqt5.

Додаток В: містить логіку робити.

Зображення відображаються за допомогою QQuickImageProvider. Цей клас дозволяє QML коду отримувати зображення за URI.

Клас ImageProvider (що спадкує QQuickImageProvider) має метод requestPixmap що для кожного URI повертає відповідне зображення

Клас Main - містить логіку роботи застосунку. Він має властивості (pyqtProperty) для параметрів шуму, які оновлюють картинку коли це потрібно.

Має методи для завантаження зображення та генерації нового шуму. Ці властивості і методи використовуються з QML коду.

Вигляд додатку та принцип роботи:

Рис. 2
Фото роботи програми на момент її запуску;
Два верхні зображення є не “зашумленими”
Нижні зображення, при використанні, показують результати накладання шуму.

При запуску, за замовчанням, встановлено зображення та k-простір, повзунки в правому верхньому кутку дають змогу змінювати інтенсивність накладання шумів.

у лівому верхньому кутку розташована кнопка “Open”, з допомогою якої можна завантажити в програму власне зображення та проаналізувати дію шумів на ньому.

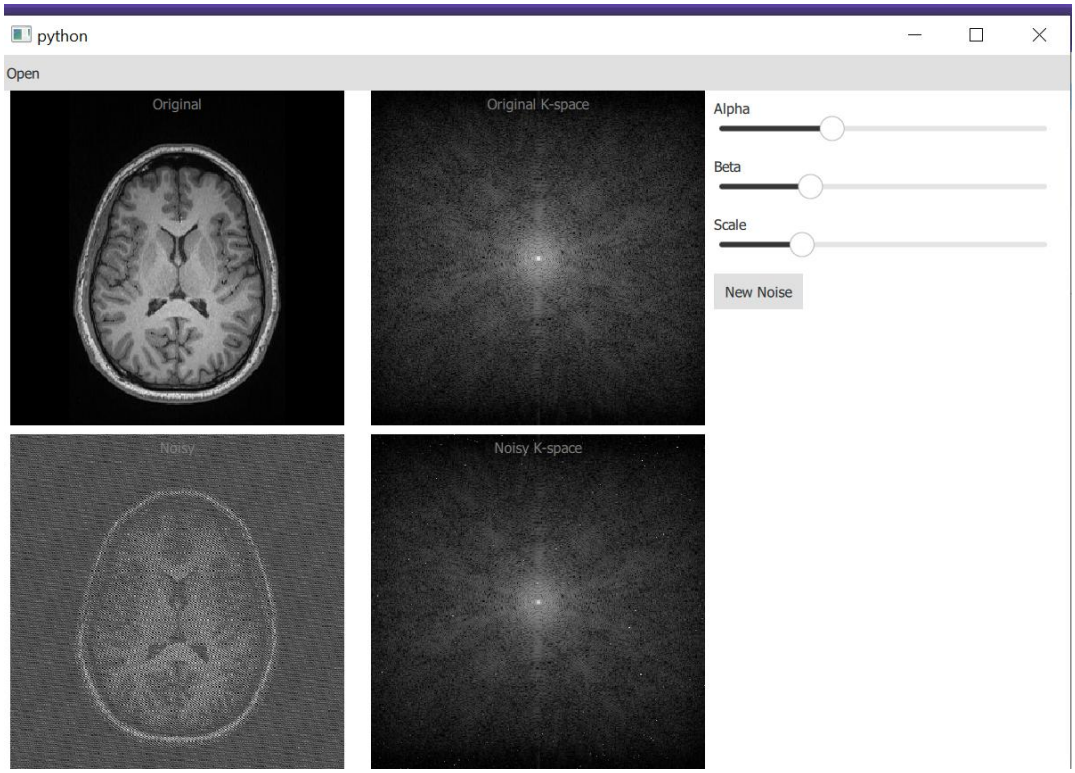

Рис. 3
Фото роботи програми на під час використання;
Два верхні зображення є не “зашумленими”
Нижні зображення показують результати накладання шуму.

ВИСНОВОК

В ході виконання дипломної роботи було проаналізовано поняття та причини виникнення шумів на знімках МРТ.

Було підібрано можливі рішення, описано різні технології та проаналізовано шляхи виконання поставлених завдань.

Розглянуто можливості мови програмування Python, плюси паралельного програмування даною мовою для якісного виконання поставленої задачі.

Під час виконання дипломної роботи було створено та протестовано програму, що має змогу накладати шуми на МРТ знімки мозку для їх подальшого вивчення та аналізу, з додатковою функцією завантаження у додаток власного фото.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Machine learning in Magnetic Resonance Imaging: Image reconstruction [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/pii/S1120179721001095#b0055
2. Komarov A.O., Naguliak O.O., Netreba A.V., Radchenko S.P., Sudakov O.O.: Compensation of the Magnetic Gradient Instability in MRI by Measurements Data for Several Angular Orientations
3. Jethi AK, Murugesan B, Ram K, Sivaprakasam M. Dual-Encoder-Unet For Fast Mri Reconstruction. 2020 IEEE 17th International Symposium on Biomedical Imaging Workshops (ISBI Workshops)2020.
4. Mardani M, Monajemi H, Papyan V, Vasanawala S, Donoho D, Pauly J. Recurrent generative adversarial networks for proximal learning and automated compressive image recovery. arXiv preprint arXiv:171110046, 2017
5. Погорілий С. Д., Семьонов Б. О. ДОСЛІДЖЕННЯ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ МОВОЮ PYTHON З ВИКОРИСТАННЯМ РІЗНИХ ПЛАТФОРМ
6. Погорілий С. Д. Програмне конструювання : підручник / С. Д. Погорілий ; за ред. О. В. Третяка. – 2-ге вид. – Київ : ВПЦ «Київський університет», 2007. – 438 с. – (Автоматизація наукових досліджень).
7. Погорілий С. Д. Генетичний алгоритм розв’язання задачі маршрутизації в мережах / С. Д. Погорілий, Р. В. Білоус // Проблеми програмування. – 2010. – № 2–3 : Матеріали сьомої міжнародної науково-практичної конференції з програмування «УкрПРОГ’2010».
8. Програмування числових методів мовою Python : навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко ; за ред. А. В. Анісімова. – Київ : ВПЦ «Київський університет», 2013.
9. Farber R. CUDA Application Design and Development / Rob Farber. – Waltham, MA : Morgan Kaufmann, 2011.
10. GPU Accelerated Computing with Python [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/howto-cuda-python.

ДОДАТКИ

ДОДАТОК А

З файлу: mri_image.py

```
import numpy as np
import cupy as cp

class MriImage:
   def init(self, pixels=None, kspace=None):
       if pixels is not None:
           self.pixels = cp.asarray(pixels)
           self.kspace = cp.zeros_like(pixels,
dtype=cp.complex64)
           self.pixels_to_kspace(pixels, self.kspace)
       elif kspace is not None:
           self.kspace = cp.asarray(kspace)
           self.pixels = cp.zeros_like(kspace,
dtype=cp.float32)
           self.kspace_to_pixels(kspace, self.pixels)
       else:
           raise ValueError("Provide either pixels or
kspace argument")

       self.kspace_abs = cp.zeros_like(self.kspace,
dtype=cp.float32)

   @staticmethod
   def kspace_to_pixels(kspace: cp.ndarray, out:
cp.ndarray):

cp.absolute(cp.fft.fftshift(cp.fft.ifft2(cp.fft.ifftshift
(kspace))), out=out)
```

```
    @staticmethod
    def pixels_to_kspace(img: cp.ndarray, out:
cp.ndarray):
        out[:] =
cp.fft.fftshift(cp.fft.fft2(cp.fft.ifftshift(img)))

    @staticmethod
    def normalize(f: cp.ndarray):
        fmin = float(cp.min(f))
        fmax = float(cp.max(f))
        if fmax != fmin:
            coeff = fmax - fmin
            f[:] = cp.floor((f[:] - fmin) / coeff * 255.)

    @property
    def kspace_image(self):
        # Prepare kspace display - get magnitude then
scale and normalize
        # K-space scaling:
https://homepages.inf.ed.ac.uk/rbf/HIPR2/pixlog.htm
        cp.absolute(self.kspace, out=self.kspace_abs)
        if cp.any(self.kspace_abs):
            scaling_c = cp.power(10., -3)
            cp.log1p(self.kspace_abs * scaling_c,
out=self.kspace_abs)
            self.normalize(self.kspace_abs)
        return cp.asnumpy(cp.require(self.kspace_abs,
cp.uint8))

    @property
    def image(self):
        self._scale_pixels(self.pixels)
        return cp.asnumpy(cp.require(self.pixels,
cp.uint8))
```

```
    @staticmethod
    def _scale_pixels(f: cp.ndarray):
        fmax = cp.max(f)
        fmin = cp.min(f)
        if fmax != fmin:
            f[:] = cp.multiply(cp.divide(f, fmax), 255);
```

ДОДАТОК Б

З файлу: alpha_stable_noise.py

```
from mri_image import MriImage

import cupy as cp
from scipy.stats import levy_stable

class AlphaStableNoise:
    def init(self, alpha = (0.67, 0.67), beta = (0.5,
0.5), scale=(0.1, 0.1)):
        self.alpha = alpha
        self.beta = beta
        self.scale = scale

    def add_to(self, mri):
        magnitude = cp.abs(mri.kspace)
        size = magnitude.shape

        # Generate a random alpha stable distribution
        real_noise = levy_stable.rvs(self.alpha[0],
self.beta[0], scale=self.scale[0], size=size)
        img_noise = levy_stable.rvs(self.alpha[1],
self.beta[1], scale=self.scale[1], size=size)
        noise = real_noise + 1j * img_noise
```

```
        return MriImage(kspace=mri.kspace +
cp.asarray(noise))
```

ДОДАТОК В

З файлу: main.py

```
from mri_image import MriImage
from alpha_stable_noise import AlphaStableNoise

import PIL as pil
import pydicom
import numpy as np
import pathlib
import sys
from os.path import join

from uuid import uuid4

from PyQt5 import QtQuick
from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot,
pyqtProperty, QVariant, QUrl, Qt
from PyQt5.QtGui import QImage, QPixmap, QColor
from PyQt5.QtQml import QQmlApplicationEngine
from PyQt5.QtWidgets import QApplication, QMessageBox

class ImageProvider(QtQuick.QQuickImageProvider):
    def init(self):
        QtQuick.QQuickImageProvider.init(self,
QtQuick.QQuickImageProvider.Pixmap)
        self.image = None
        self.noisy_image = None
```

```
    def requestPixmap(self, id_str: str, requested_size):
        img = None
        try:
            if id_str.startswith('image'):
                if self.image is not None:
                    img = QImage(self.image.image,

self.image.image.shape[1],

self.image.image.shape[0],

self.image.image.strides[0],

QImage.Format_Grayscale8)

            elif id_str.startswith('kspace'):
                if self.image is not None:
                    img = QImage(self.image.kspace_image,

self.image.kspace_image.shape[1],

self.image.kspace_image.shape[0],

self.image.kspace_image.strides[0],

QImage.Format_Grayscale8)

            elif id_str.startswith('noisy_image'):
                if self.noisy_image is not None:
                    img = QImage(self.noisy_image.image,

self.noisy_image.image.shape[1],

self.noisy_image.image.shape[0],

self.noisy_image.image.strides[0],
```

```
QImage.Format_Grayscale8)

            elif id_str.startswith('noisy_kspace'):
                if self.noisy_image is not None:
                    img =
QImage(self.noisy_image.kspace_image,

self.noisy_image.kspace_image.shape[1],

self.noisy_image.kspace_image.shape[0],

self.noisy_image.kspace_image.strides[0],

QImage.Format_Grayscale8)

            else:
                raise NameError

        except Exception as e:
            print(e)

        if img is None:
            img = QPixmap(requested_size)
            img.fill(QColor('white'))

        return QPixmap(img), QPixmap(img).size()


def read_file(path: str, dtype: np.dtype = np.float32) ->
np.ndarray:
    try:
        with pil.Image.open(path) as f:
            img_file = f.convert('F')  # 'F' mode: 32-bit
floating point pixels
```

```
            img_pixel_array =
np.array(img_file).astype(dtype)
        return img_pixel_array
    except FileNotFoundError:
        return
    except:
        pass

    try:
        with pydicom.dcmread(path) as dcm_file:
            img_pixel_array =
dcm_file.pixel_array.astype(dtype)
        img_pixel_array.setflags(write=True)
        return img_pixel_array
    except pydicom.errors.InvalidDicomError:
        pass

    try:
        return np.load(path)
    except Exception as e:
        raise e

def show_message(text, fatal=False):
    box = QMessageBox()
    box.setIcon(QMessageBox.Critical)
    box.setTextFormat(Qt.RichText)
    box.setText(text)
    box.setWindowTitle("Error")
    box.exec_()
    if fatal:
        sys.exit()
    else:
        return box.result()
```

```
class MainApp(QObject):
    orig_img_changed = pyqtSignal()
    noisy_img_changed = pyqtSignal()

    def init(self, context, parent=None):
        super().init(parent)
        self.ctx = context
        self.image_provider = ImageProvider()

        self.noise = AlphaStableNoise()
        self.image = None
        self.noisy_image = None
@pyqtSlot(QVariant, name="load_image")
    def load_image(self, url: list):
        image_url = url.toLocalFile()

        try:
            file_data = read_file(image_url)
            is_image = len(file_data.shape) <= 2
        except Exception as e:
            print(e)
            show_message(f"Cannot load file
({image_url})")
            return

        if is_image:
            self.image = MriImage(pixels=file_data)
        else:
            # Extract 2D data slices from 3D array
            file_data = file_data[0, :, :]
            self.image = MriImage(kspace=file_data)

        self.image_provider.image = self.image
        self.orig_img_changed.emit()
```

```
        self.update_noise()

    @pyqtSlot(name="update_noise")
    def update_noise(self):
        try:
            self.noisy_image =
self.noise.add_to(self.image)
            self.image_provider.noisy_image =
self.noisy_image
            self.noisy_img_changed.emit()
        except Exception as e:
            show_message(str(e))

    @pyqtProperty('QString', notify=orig_img_changed)
    def orig_img_source(self):
        return "image://imgs/image" + uuid4().hex

    @pyqtProperty('QString', notify=orig_img_changed)
    def orig_kspace_img_source(self):
        return "image://imgs/kspace" + uuid4().hex

    @pyqtProperty('QString', notify=noisy_img_changed)
    def noisy_img_source(self):
        return "image://imgs/noisy_image" + uuid4().hex

    @pyqtProperty('QString', notify=noisy_img_changed)
    def noisy_kspace_img_source(self):
        return "image://imgs/noisy_kspace" + uuid4().hex

    @pyqtProperty(float)
    def alpha(self):
        return self.noise.alpha
```

```
    @alpha.setter
    def alpha(self, new):
        self.noise.alpha = (new, new)
        self.update_noise()

    @pyqtProperty(float)
    def beta(self):
        return self.noise.beta

    @beta.setter
    def beta(self, new):
        self.noise.beta = (new, new)
        self.update_noise()

    @pyqtProperty(float)
    def scale(self):
        return self.noise.scale

    @scale.setter
    def scale(self, new):
        self.noise.scale = (new, new)
        self.update_noise()


def main():
    app_path = pathlib.Path(file).parent.absolute()

    app = QApplication([])

    engine = QQmlApplicationEngine()
    ctx = engine.rootContext()

    mainapp = MainApp(ctx)
```

```
    mainapp.load_image(QUrl.fromLocalFile(join(app_path,
"mri_images", "default.dcm")))
    ctx.setContextProperty("application", mainapp)

    engine.addImageProvider("imgs",
mainapp.image_provider)
    engine.load(join(app_path, "ui", "window.qml"))

    sys.exit(app.exec_())

if name == "main":
    main()
```