

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

« ____ » _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»**

спеціальності 121 Інженерія програмного забезпечення

**на тему: «Автоматизована система класифікації травм на основі машинного
навчання»**

Виконав:

студент IV курсу, групи ТВ-23

Лисенко Олександр Сергійович

(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник:

старший викладач Дацюк О.А.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Консультант:

доцент, к.т.н. Залевська О.В.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент:

_____ (посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ

«___» _____ 2026р.

ЗАВДАННЯ

на дипломну роботу студенту

_____ Лисенку Олександр Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Автоматизована система класифікації травм на основі машинного навчання

керівник роботи Дацюк Оксана Антонівна, старший викладач

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026р. № 1992-с

2. Строк подання студентом роботи «10» червня 2026р.

3. Вихідні дані до роботи: мова програмування Python, середовище VS Code, набір знімків магнітно-резонансної томографії колінних суглобів

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): проаналізувати предметну область за темою дипломної роботи, дослідити існуючі методи обробки та класифікації медичних зображень, ознайомитись з програмними засобами та інструментами, що використовуються для розробки систем класифікації; написати програмний код системи.

5. Перелік ілюстративного матеріалу: знімки програмного продукту, знімки магнітно-резонансної томографії колінних суглобів

6. Дата видачі завдання «31» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	
2	Дослідження предметної області	01.11.2025 – 31.11.2025	
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	
4	Постановка вимог до проектування системи	01.01.2026 – 31.01.2026	
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	
6	Тестування	04.05.2026 – 10.05.2026	
7	Захист програмного продукту	11.05.2026 – 15.05.2026	
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	
9	Передзахист	01.06.2026 – 05.06.2026	
10	Захист	15.06.2026 – 19.06.2026	

Студент

(підпис)

Олександр ЛИСЕНКО

(ім'я, прізвище)

Керівник роботи

(підпис)

Оксана ДАЦЮК

(ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 60 сторінок, 32 рисунки, 4 таблиці, 2 додатки та 16 посилань.

Метою роботи є створення автоматизованої системи класифікації травм колінних суглобів з використанням машинного навчання.

Об'єктом роботи є процес автоматизованого аналізу та класифікації медичних зображень колінних суглобів.

Предметом роботи є методи та моделі машинного навчання для автоматичного виявлення та класифікації травм колінного суглоба.

Завдання роботи, які необхідно виконати для досягнення мети:

1. Аналіз існуючих рішень для автоматизації медичної класифікації.
2. Підготовка вхідних даних для навчання.
3. Створення архітектури мережі машинного навчання.
4. Навчання моделі й оцінка її роботи.
5. Розробка графічного інтерфейсу користувача.

Практичне значення одержаних результатів полягає в застосуванні методів машинного навчання у сфері медицини для визначення травм і патологій колінного суглоба. Розроблено програмний продукт, який дозволяє виконати повний цикл тренування власної моделі машинного навчання: від підготовки медичних наборів даних до тренування моделей, і також надає можливість використати натреновані моделі для класифікації нових МРТ-знімків.

Ключові слова: Python, машинне навчання, нейронна мережа, класифікація, колінний суглоб, магнітно-резонансна томографія, обробка зображень, конвертація зображень, DICOM.

ABSTRACT

Structure and scope of the thesis. The work contains 60 pages, 32 figures, 4 tables, 2 appendices, and 16 references.

The purpose of the work is to create an automated knee joint injury classification system using machine learning.

The object of the work is the process of automated analysis and classification of medical images of knee joints.

The subject of the work is machine learning methods and models for the automatic detection and classification of knee joint injuries.

Tasks of the work to be completed to achieve the purpose:

1. Analysis of existing solutions for the automation of medical classification.
2. Preparation of input data for training.
3. Creation of the machine learning network architecture.
4. Model training and evaluation of its performance.
5. Development of a graphical user interface.

The practical significance of the obtained results lies in the application of machine learning methods in the medical field to determine knee joint injuries and pathologies. A software product has been developed that allows for a complete cycle of training a custom machine learning model: from preparing medical datasets to training models, and also provides the capability to use trained models for the classification of new MRI scans.

Keywords: Python, machine learning, neural network, classification, knee joint, magnetic resonance imaging, image processing, image conversion, DICOM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	9
1 ПОСТАНОВКА ЗАВДАННЯ.....	10
1.1 Актуальність завдання	11
Висновки до розділу 1	12
2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	14
2.1 MedGemma.....	14
2.2 AI-Rad Companion	15
2.3 Merge від Merative	17
Висновки до розділу 2	18
3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	20
3.1 Мова програмування Python.....	20
3.2 Pydicom	21
3.3 NumPy	22
3.4 PyQt.....	22
3.5 Logging.....	23
3.6 PyTorch.....	24
3.7 Scikit-learn.....	25
3.8 CNN та архітектура ResNet.....	25
Висновки до розділу 3	27
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ.....	29
4.1 Взаємодія користувача та системи.....	29
4.2 Архітектура системи	32
4.2.1 Модуль роботи із зображеннями	33
4.2.2 Модуль машинного навчання	36
4.2.3 Модуль взаємодії та діагностики	39
4.3 Структура нейронних мереж	41

4.3.1 2D-мережа.....	42
4.3.2 3D-мережа.....	45
4.4 Проблеми розроблених мереж.....	49
4.4.1 Нестача даних.....	49
4.4.2 Втрата даних на знімках	51
Висновки до розділу 4	53
5 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ ПРОГРАМНОЮ СИСТЕМОЮ	54
5.1 Графічний інтерфейс.....	54
5.2 Інтерфейс командного рядка	56
Висновки до розділу 5	57
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59
ДОДАТОК А Лістинг окремих файлів розробленої системи.....	61
ДОДАТОК В Презентація	73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ML	Machine Learning (Машинне навчання)
ІІІ	Іштучний інтелект
DICOM	Digital Imaging and Communications in Medicine (Цифрові зображення та комунікації в медицині)
CPU	Central Processing Unit (Центральний процесор)
GPU	Graphics Processing Unit (Графічний процесор)
UML	Unified Modeling Language (Уніфікована мова моделювання)
CNN	Convolutional Neural Network (Згорткова нейронна мережа)
ResNet	Residual Neural Network (Залишкова нейронна мережа)
MPT	Магнітно-резонансна томографія
КТ	Комп'ютерна томографія
NPY	NumPy
PNG	Portable Network Graphics
GUI	Graphical User Interface (Графічний інтерфейс користувача)
CLI	Command-line Interface (Інтерфейс командного рядка)
UML	Unified Modeling Language (Уніфікована мова моделювання)
3D	3-dimensional (3-вимірний)
2D	2-dimensional (2-вимірний)
LLM	Large Language Model (Велика мовна модель)
FC	Fully Connected Layer (Повнозв'язний шар)

ВСТУП

У зв'язку з розвитком штучного інтелекту (ШІ), машинного навчання (ML) та науки про дані у медичному секторі відбуваються значні зміни, особливо у сфері діагностики [1]. Сучасну медичну діагностику можна охарактеризувати накопиченням великих обсягів даних, значну частину яких займають саме медичні знімки: магнітно-резонансної томографії (МРТ), комп'ютерної томографії (КТ), рентгенівські знімки тощо. Традиційно такі знімки оцінювалися спеціально навченими лікарями. Але зі збільшенням навантаження на систему охорони здоров'я, навантаження аналогічно зростає і на лікарів, особливо в кризові часи.

У зв'язку зі зростанням робочого навантаження на працівників у сфері охорони здоров'я в Україні, пов'язаного з російським вторгненням, виникає гостра потреба у автоматизованих системах аналізу медичних зображень. Найбільш перспективним інструментом для цього завдання є технологія ШІ, зокрема методи глибокого машинного навчання та комп'ютерного зору. Завдяки архітектурі сучасних згорткових нейронних мереж (CNN) вже можна ефективно визначати ознаки та закономірності патологій з медичних знімків [2].

Розроблена система полегшує роботу медичних працівників шляхом автоматизації частини їхньої роботи та працює, як технологія «другої думки». Також система за необхідності може бути адаптована до поточних вимог на місці використання.

Робота містить п'ять розділів:

1. Постановка завдання.
2. Аналіз існуючих рішень.
3. Засоби розробки програмного забезпечення.
4. Опис програмної реалізації.
5. Взаємодія користувача із програмною системою.

1 ПОСТАНОВКА ЗАВДАННЯ

Метою роботи є створення автоматизованої системи класифікації травм колінних суглобів з використанням машинного навчання. Із самої назви можна зробити висновок, що система повинна працювати зі спеціалізованими медичними даними, у випадку розробленої системи – це МРТ-знімки колінних суглобів.

Зі збільшенням використання ІІІ в медицині все більше уваги надається темі безпеки даних у сфері охорони здоров'я. В Україні цій темі вже давно приділяють увагу у зв'язку з прискореною цифровізацією державних послуг. Наприклад, офіційний інтернет-портал Міністерства охорони здоров'я України вже має окремий розділ в базі знань eHealth із практичними рекомендаціями щодо кіберзахисту та дотримання вимог законодавства [3]. Особливо гостро проблема безпеки даних стосується моделей машинного навчання, адже переважна більшість спеціалізованих знімків (до яких входить і МРТ) використовують формат DICOM, який зберігає разом із самим знімком метадані – текст, який описує зображення та містить персональні дані пацієнта [4]. Тому задля етичності роботи дані потрібно попередньо анонімізувати, залишивши лише саме зображення знімка.

Створювана система повинна використовувати саме механізми та моделі машинного навчання, тобто машина повинна самостійно навчатися на наданих даних.

Система повинна мати та застосовувати комплексний механізм логування роботи, що, у свою чергу, допомагає в документуванні роботи системи та її налагодженні. Особливо такий механізм допомагає в оцінці та порівнянні роботи різних ML-моделей.

Для зручної роботи із системою необхідно створити графічний інтерфейс користувача (GUI), який дозволить виконати базовий функціонал системи: перегляд завантаженого знімка та його аналіз для класифікації травм і патологій.

Для доступу до повного розробленого функціоналу системи необхідно мати незалежний від основного GUI, прихований від базового цільового користувача

інтерфейс. На цю роль гарно підходить інтерфейс командного рядка (CLI), який зможе запустити лише більш досвідчений в роботі з програмним забезпеченням користувач. Це також слугуватиме бар'єром входу, що не дозволить випадковим особам через нестачу знань зламати базовий функціонал системи.

1.1 Актуальність завдання

Автоматизація клінічного робочого процесу за допомогою спеціальних програмних систем дозволяє зменшити щоденне робоче навантаження на медичних працівників. Це, у свою чергу, покращує розподіл робочого часу, дозволяє надавати кваліфіковану допомогу більшій кількості пацієнтів та знижує рівень професійного стресу і вигорання, що пов'язані з монотонним аналізом сотень знімків томографії.

Розроблена система здатна працювати як зручне джерело «другої думки» для лікаря під час проведення діагностики. Це зменшує потребу в тривалих колегіальних консультаціях між декількома профільними фахівцями у складних клінічних випадках. Це суттєво скорочує загальний час від моменту сканування до встановлення кінцевого діагнозу, усуваючи зайві бюрократичні та організаційні кроки, які зазвичай відволікають від роботи відразу кількох працівників медичного закладу.

Система демонструє практичну цінність та ефективність навіть за умов дефіциту та обмеженої кількості специфічних медичних даних, доступних для тренування. Оскільки збір великих анотованих баз МРТ-знімків – це тривалий і дорогий процес, архітектура розробленого програмного продукту була оптимізована під роботу в умовах незбалансованих вибірок. Завдяки такій оптимізації результати аналізу, які надає система, слугують надійним інструментом первинної діагностики. Це допомагає миттєво звернути увагу лікаря на знімки, які з високою вірогідністю містять приховані чи малопомітні ознаки патологій.

На сучасному ринку медичних інформаційних систем і програмного забезпечення для радіології присутня суттєва прогалина. Більшість комерційних

рішень є закритими, дорогими та громіздкими, що робить їх недоступними для більшості медичних закладів. Існує явна потреба в доступних та відносно легких системах, які можна швидко налаштувати та розгорнути під локальні технічні потужності конкретної клініки чи лабораторії.

З огляду на це, розробка цієї системи є вкрай актуальним завданням як для практикуючих медичних працівників, так і для студентів медичних спеціальностей. Продукт має пряме прикладне значення для галузей ортопедії, травматології, ревматології та радіології.

Висновки до розділу 1

У першому розділі було розглянуто та сформульовано основні вимоги до створюваної автоматизованої системи класифікації травм на основі машинного навчання:

1. Робота з медичними MPT-знімками формату DICOM.
2. Анонімізація даних пацієнтів.
3. Використання технологій машинного навчання для класифікації травм.
4. Логування процесу роботи системи.
5. Наявність GUI для використання натренованих моделей для класифікації травм (базовий функціонал).
6. Наявність CLI для запуску повного циклу тренування нових ML моделей (повний функціонал).

Обґрунтовано актуальність розробленої системи через її доступність, здатність знижувати щоденне монотонне навантаження на медичний персонал та виступати надійним джерелом «другої думки» для перевірки складних клінічних випадків. Особливу цінність має адаптація архітектури програмного продукту до роботи в умовах дефіциту та незбалансованих медичних даних, що дозволяє використовувати систему як ефективний інструмент первинної діагностики.

Було приділено увагу питанням безпеки та етичності роботи з персональними даними та визначено необхідність попередньої анонімізації даних через наявність конфіденційної інформації про пацієнта у їхніх метаданих.

Окреслено основні задачі майбутньої програмної системи. Встановлено, що вона має базуватися на технології машинного навчання, містити комплексний механізм логування, а також мати дворівневу структуру керування: графічний інтерфейс для виконання базових функцій лікарем та прихований інтерфейс командного рядка для захисту основного функціоналу від дій недосвідчених користувачів.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Перед початком роботи над власною системою класифікації травм, доцільно ознайомитися з уже розробленими, схожими по функціоналу, системами для формування загальної картини ринку. Разом з цим, такий аналіз дозволяє краще спланувати програмний продукт, який може стати альтернативою для вже наявних систем.

На ринку систем машинного навчання для класифікації травм можна виділити таких представників: MedGemma, AI-Rad Companion, програмний комплекс Merge від Merative.

2.1 MedGemma

MedGemma – це відкрита колекція мультимодальних моделей, що були оптимізовані для спільного розуміння медичних зображень і тексту [5]. Всі моделі знаходяться у вільному доступі на платформі Hugging Face, рисунок 2.1.

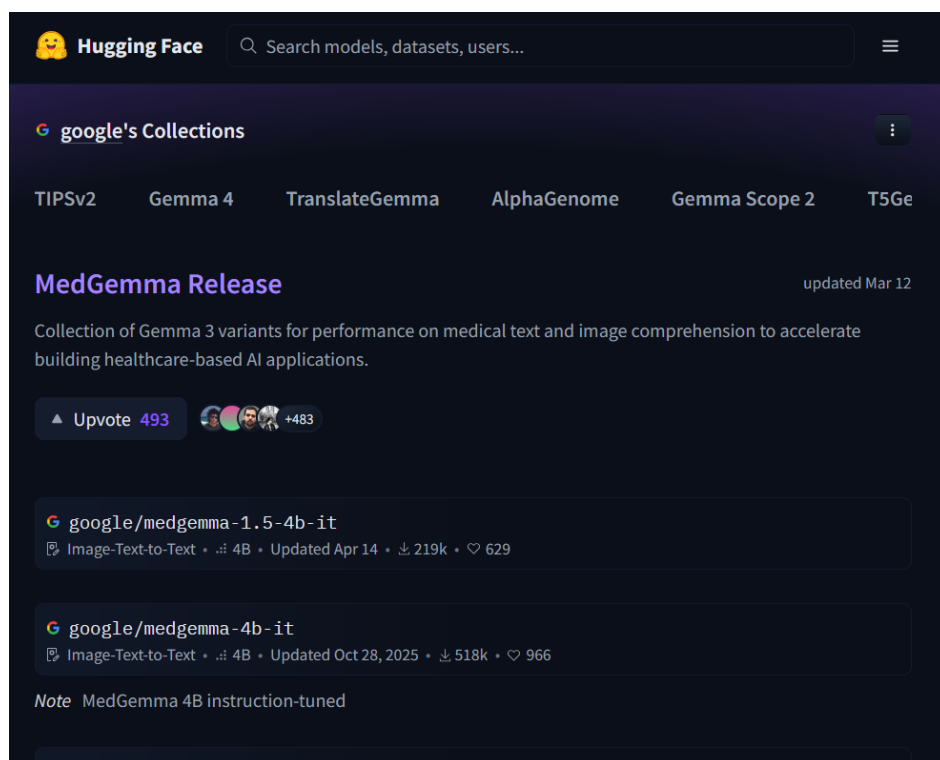


Рисунок 2.1 – Моделі MedGemma на платформі Hugging Face

Хоч це і не повноцінна автономна система, а лише набір моделей, всі вони вузькоспеціалізовані для роботи з медичними даними. В основі цієї колекції лежать великі мовні моделі сімейства Gemma від Google, які були додатково натреновані на великих масивах медичних зображень та відповідних їм клінічних текстових звітах.

Серед можливих варіантів використання розробниками надаються такі приклади: аналіз високорозмірної медичної візуалізації для КТ, МРТ та гістопатологічних сканів; контекстуальний аналіз рентгенівських знімків грудної клітини; розуміння складної медичної документації та генерація відповідей на клінічні запитання. Головною особливістю MedGemma є її мультимодальність, що дозволяє моделі не просто класифікувати зображення, а «спілкуватися» з користувачем на основі візуальних даних.

Оскільки дана робота має на меті також і навчальний характер, було вирішено не використовувати вже готові моделі. Замість цього розробка зосереджується на створенні власної архітектури або налаштуванні під специфічні потреби згорткові нейронні мережі та моделі сімейства ResNet.

2.2 AI-Rad Companion

AI-Rad Companion – це лінійка програмних продуктів, які подаються як способи зменшити робоче навантаження та підвищити точність діагностики під час роботи з медичними зображеннями [6]. Сторінку офіційного сайту лінійки зображено на рисунку 2.2.

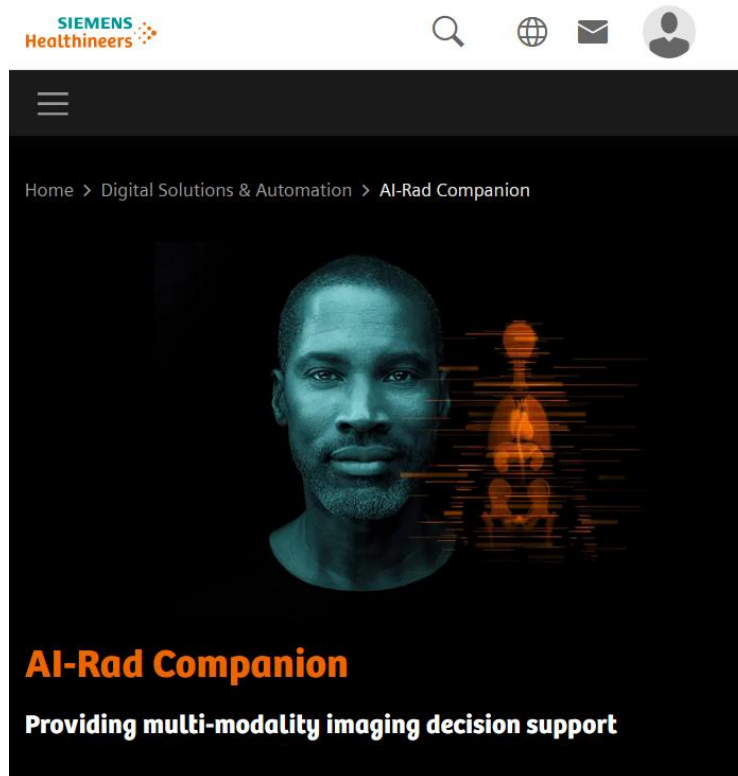


Рисунок 2.2 – Сторінка офіційного сайту Siemens Healthineers для AI-Rad Companion

Вся лінійка програм функціонує як система: вони дозволяють переглядати зображення, сегментувати медичні знімки, і, залежно від продукту, в них доступне виявлення та підсвічування патологій. З недоліків – кожен програмний продукт лінійки спеціалізований для роботи лише з конкретними органами та станами. На даний момент в лінійці присутні продукти для роботи зі знімками мозку, КТ-знімками органів грудної клітини, рентгенівськими знімками грудної клітини, МРТ-знімками простати і плануванням променевої терапії внутрішніх органів.

Однією з переваг системи є використання технології кінематографічного рендерингу та автоматична генерація об'ємних 3D-рендерів. Кожен окремий продукт лінійки оснащений власним набором інструментів для проведення автоматичних вимірювань, таких як підрахунок об'ємів новоутворень, оцінка щільності тканин або виявлення кальцифікацій. Отримані параметри аналізуються системою та автоматично експортуються у фінальні структуровані звіти.

Також варто зазначити, що AI-Rad Companion це комерційний продукт, побудований на хмарних технологіях. Він розповсюджується за моделлю «програма як послуга» або через ліцензійну підписку і також потребує зв'язку з цифровою екосистемою Siemens Healthineers.

На противагу таким потужним комерційним рішенням, розроблена у даній роботі програмна система має кілька принципових переваг для кінцевого користувача. По-перше, це економічна доступність: система створена на основі безкоштовних технологій із відкритим кодом, що дозволяє зекономити на витратах для ліцензійних підписок, які є основною перешкодою для широкого застосування програмного забезпечення такого типу. По-друге, хмарна архітектура AI-Rad Companion вимагає постійного підключення до швидкісного інтернету, тоді як представлена система є автономною та може функціонувати локально на робочому місці лікаря.

Окремою перевагою є низький поріг входження та простота використання порівняно з комерційними продуктами компанії Siemens Healthineers, які вимагають тривалого навчання для роботи з ними. Розроблений продукт пропонує лаконічний GUI, орієнтований виключно на вирішення базових завдань первинної діагностики без зайвих надбудов.

2.3 Merge від Merative

Програмний комплекс Merge, компанії Merative – це масштабна екосистема управління медичними даними. Вона включає в себе різноманітні компоненти, включно з інструментами для роботи з візуалізацією, радіологією, кардіологією, даними пацієнтів, а також набір інструментів для роботи з DICOM-файлами [7]. Вигляд сторінки сайту Merative з описом Merge наведено на рисунку 2.3.

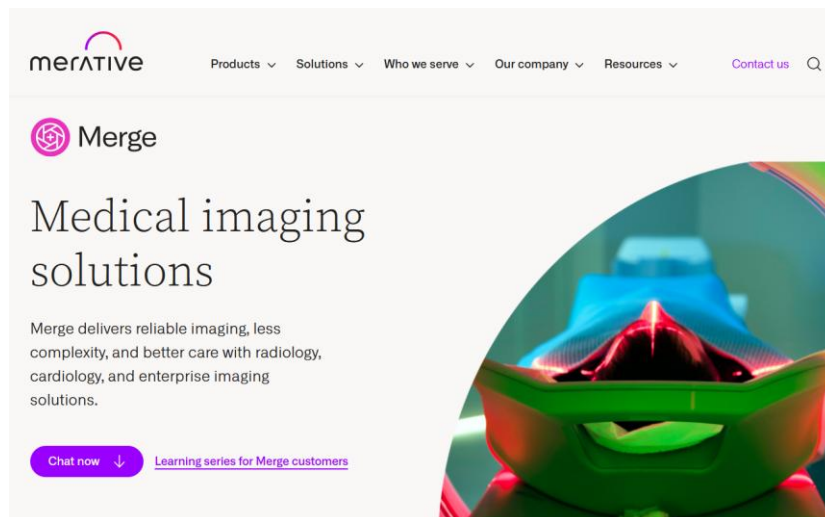


Рисунок 2.3 – Сторінка сайту компанії Merative з описом Merge

Платформа побудована на базі хмарної архітектури та підтримує алгоритми сортування зображень. Наприклад платформа дозволяє автоматично розподіляти МРТ- та КТ-скани за пріоритетністю і типом патологій. Завдяки модульній структурі кожен медичний заклад може налаштовувати робочі процеси під власні потреби, що дозволяє швидко отримати доступ до архіву знімків пацієнта безпосередньо з робочого місця лікаря. Окрім цього, система підтримує просунуті алгоритми штучного інтелекту для автоматичного виділення контурів органів на знімках та проведення складних анатомічних вимірювань у реальному часі.

Merge – це комерційна, багатофункціональна та підтримувана програмна екосистема, що вже активно використовується організаціями охорони здоров'я по всьому світу. Як наслідок, плата за користування програмним продуктом такого рівня буде відповідною, що, на жаль, не дозволить ознайомитися із системою навіть професійним лікарям з досвідом, не беручи до уваги студентів.

Висновки до розділу 2

У другому розділі було проаналізовано вже доступні на ринку автоматизовані системи для медичної класифікації різних патологій і травм, а

також досліджено існуючі моделі машинного навчання, створені для роботи з медичними даними.

Доступні системи, хоч і пропонують потужний функціонал та навіть уже впроваджуються у медичних закладах по всьому світу, досі залишаються малодоступними для загалу. Перш за все це зумовлено високою вартістю підписки на такі продукти. Іншим фактором є складність подібних систем, що потребує додаткових знань навіть для базового користування. Також варто зазначити, що для роботи усіх наведених систем необхідне постійне підключення до хмарних сервісів компаній-дистриб'юторів. Відсутність підключення, погане інтернет-з'єднання або проблеми на серверах компаній унеможливить користування такими системами.

Тому у висновку до проведеного аналізу можна сказати, що на ринку є місце для більш доступної альтернативи у вигляді локальної, простої та зрозумілої системи класифікації травм.

З ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На етапі планування системи важливо визначитися з технологічними інструментами, які будуть використовуватися під час розробки. Особливо варто звернути увагу на специфіку завдання роботи з медичними даними та зображеннями, що потребує власного набору інструментів.

Створювана система працюватиме з МРТ-знімками, що зберігаються у форматі DICOM, для цього буде використано спеціалізований пакет для мови Python: Pydicom. На етапі тренування моделі машинного навчання, модель повинна показувати найкращий результат при оптимальному часі роботи, тому на даному етапі буде використано Python бібліотеку PyTorch, а для оцінки натренованої моделі буде використано бібліотеку Sklearn. Графічний інтерфейс повинен бути простим та зрозумілим не технічно-спеціалізованому користувачу, і стабільно працювати. Для досягнення такого ефекту буде використано Python бібліотеку PyQt. Для логування роботи системи використовуватиметься вбудований модуль Python – logging.

3.1 Мова програмування Python

Python фактично вважається стандартом у галузі аналізу даних та штучного інтелекту. Логотип мови наведений на рисунку 3.1.



Рисунок 3.1 – Логотип Python

Python має екосистему потужних бібліотек для наукових обчислень. На додачу до бібліотек, у мові присутні модулі для асинхронної роботи, і такий багатий функціонал пишеться відносно простим та читабельним синтаксисом. Тому відразу було обрано саме цю мову програмування, як тільки з'явилася потреба в розробці ML-проекту.

3.2 Pydicom

Pydicom – це чистий Python-пакет для роботи з DICOM-даними [8]. Цей пакет дозволяє витягнути лише дані зображення з DICOM-файлу МРТ-знімку, ігноруючи всі метадані, що були збережені в самому файлі скану, фактично анонімізуючи інформацію про пацієнтів. Також він дозволяє працювати із даними пікселів зображення за допомогою бібліотеки NumPy. Логотип Pydicom наведено на рисунку 3.2.



Рисунок 3.2 – Логотип Pydicom

Завдяки Pydicom система може коректно інтерпретувати коефіцієнти масштабування пікселів та іншу службову інформацію, необхідну для правильного відображення відтінків сірого на знімках коліна, що безпосередньо впливає на точність подальшого розпізнавання закономірностей ознак.

3.3 NumPy

Отримані за допомогою бібліотеки Pydicom дані пікселів потребують інструменту для подальших математичних операцій та візуалізації. Стандартом для цієї сфери є бібліотека NumPy, яка дозволяє працювати з високоефективними структурами даних. Логотип NumPy представлено на рисунку 3.3.

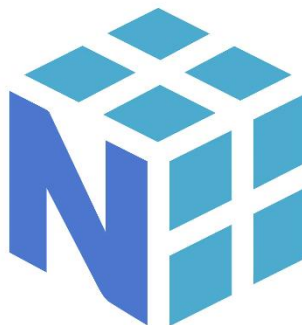


Рисунок 3.3 – Логотип NumPy

В основі бібліотеки лежать два фундаментальні об'єкти: об'єкт N-вимірного масиву (`ndarray`) та об'єкт універсальної функції (`ufunc`). Для задачі медичної діагностики важливим є саме `ndarray`, який визначається як однорідна колекція елементів, що індексуються за допомогою N цілих чисел [9]. Бібліотека дозволяє ефективно працювати з великими багатовимірними масивами, а також надає високопродуктивні інструменти для лінійної алгебри.

У рамках розроблюваної системи класифікації інструментарій NumPy задіяний для нормалізації та аугментації знімків, зміни їхніх лінійних розмірів та конвертації сирих масивів медичних пікселів у тензори, придатні для обробки моделями глибокого навчання.

3.4 PyQt

PyQt – це Python-оболонка, яка дозволяє використовувати фреймворк для створення графічного інтерфейсу Qt. Цей фреймворк має набір готових віджетів,

які значно полегшують створення інтерфейсів [10]. Фреймворк вважається стандартом в індустрії через розвинутий функціонал та приємний вигляд створюваних інтерфейсів. Логотип PyQt представлено на рисунку 3.4.



Рисунок 3.4 – Логотип PyQt

Вибір даного інструменту обумовлений його потужною архітектурою, яка дозволяє створити плавний та охайний GUI. Для системи, що працює з машинним навчання це досить важливо, оскільки інтерфейс повинен без затримок обробляти завантаження важких графічних файлів та інтерактивно відображати результати роботи ML-моделей.

Завдяки модулю PyQt було розроблено зручне вікно перегляду знімків, а також інтегровано елементи керування для запуску аналізу, Що спрощує процес користування додатком для користувачів без втрати загальної стабільності.

3.5 Logging

Модуль логування (або журналювання) – це важливий компонент для відлагоджування програмного забезпечення. Це стандартний модуль мови програмування Python, який дозволяє налаштовувати під власні потреби механізм логування: від зберігання записів у файл до налаштування форми повідомлень [11].

У розробленій програмній системі модуль Logging виконує роль централізованої системи журналювання роботи всіх внутрішніх процесів. Він налаштований для паралельного виведення повідомлень у консоль розробника та

запису у локальний текстовий лог-файл. Це дозволяє фіксувати хронологію роботи усіх створених програмних модулів системи. Такий детальний журнал роботи став незамінним помічником для оцінки та порівняння різних конфігурацій ML-моделей, а також допомагає швидко виявляти різноманітні збої та помилки під час роботи системи через CLI-інтерфейс.

3.6 PyTorch

PyTorch – це популярна сучасна бібліотека машинного навчання з відкритим вихідним кодом, що використовується для розробки та розгортання нейронних мереж. Логотип PyTorch представлено на рисунку 3.5.



Рисунок 3.5 – Логотип PyTorch

PyTorch було вибрано саме з двох основних причин. По-перше, вона дозволяє значно прискорити обчислення за допомогою використання графічного процесора, який працює в десятки разів швидше з паралельними задачами (до яких відноситься тренування моделей штучного інтелекту), ніж центральний процесор. По-друге, бібліотека надає способи числової оптимізації загальних математичних виразів, які повсюдно використовуються під час глибокого навчання [12].

У контексті даної роботи фреймворк виступає технологічним ядром системи: за його допомогою реалізовано шари згортки, механізми пропускових з'єднань та функції активації нейромережі.

3.7 Scikit-learn

Scikit-learn – це безкоштовна бібліотека для машинного навчання з відкритим кодом. Вона побудована на базі таких технологій, як NumPy, SciPy та Matplotlib. Бібліотека надає широкий набір ефективних інструментів для статистичного моделювання, включаючи алгоритми класифікації, регресії, кластеризації, а також засоби для попередньої обробки даних та оцінки якості побудованих моделей [13]. Логотип scikit-learn представлено на рисунку 3.6.



Рисунок 3.6 – Логотип scikit-learn

У межах розроблюваної системи класифікації травм колінних суглобів інструментарій Scikit-learn використовується на етапі оцінки ефективності навченої нейромережі. Бібліотека дозволяє автоматично згенерувати аналітичний звіт щодо якості розпізнавання кожної окремої патології колінного суглоба, обчислюючи такі метрики, як точність, повнота та F1-міра для кожного класу знімків.

3.8 CNN та архітектура ResNet

При вирішенні завдань комп'ютерного зору, до якого відноситься класифікація медичних знімків, класичним способом є використання згорткових нейронних мереж [14]. Принцип роботи CNN базується на послідовному застосуванні операцій згортки та пулінгу (англ. pooling). Під час згортки фільтр проходить по зображенню, обчислює скалярний добуток і формує карти ознак, які

фіксують локальні закономірності: від простих ліній та контурів на перших шарах до складних геометричних форм і текстур анатомічних структур на глибших рівнях. Приклад архітектури CNN наведено на рисунку 3.7.

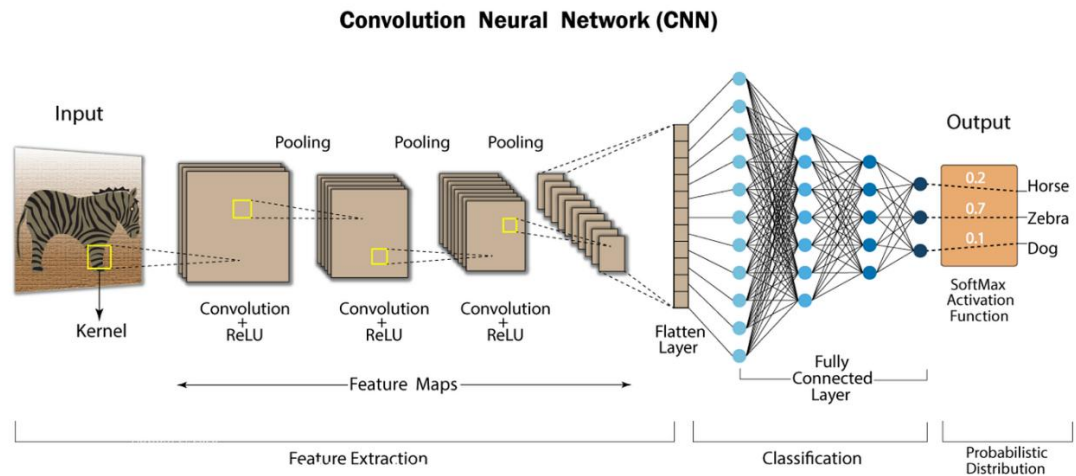


Рисунок 3.7 – Приклад архітектури CNN

Головні переваги традиційних CNN – це їхня стійкість до зсуву зображень та спільне використання ваг фільтрами, що зменшує кількість параметрів мереж. Проте у них є недолік: при спробі покращити якість класифікації шляхом простого збільшення кількості шарів, з'являється проблема затухання градієнта. Під час зворотного поширення помилки градієнт, проходячи через десятки шарів, експоненціально зменшується, через що перші шари мережі майже перестають навчатися, а точність моделі на певному етапі починає стрімко падати.

Для подолання цього обмеження класичних CNN була розроблена архітектура залишкових нейронних мереж (ResNet). Замість того, щоб змушувати шари мережі напряду вивчати цільове відображення, ResNet оптимізує так зване залишкове відображення. Практично це реалізовано за допомогою пропускових з'єднань (англ. skip connections), які дозволяють пропускати вхідний сигнал в обхід одного або кількох згорткових шарів, додаючи його безпосередньо до вихідного результату обчислень. Для кращого розуміння, схему порівняння між блоками CNN та ResNet наведено на рисунку 3.8 [15].

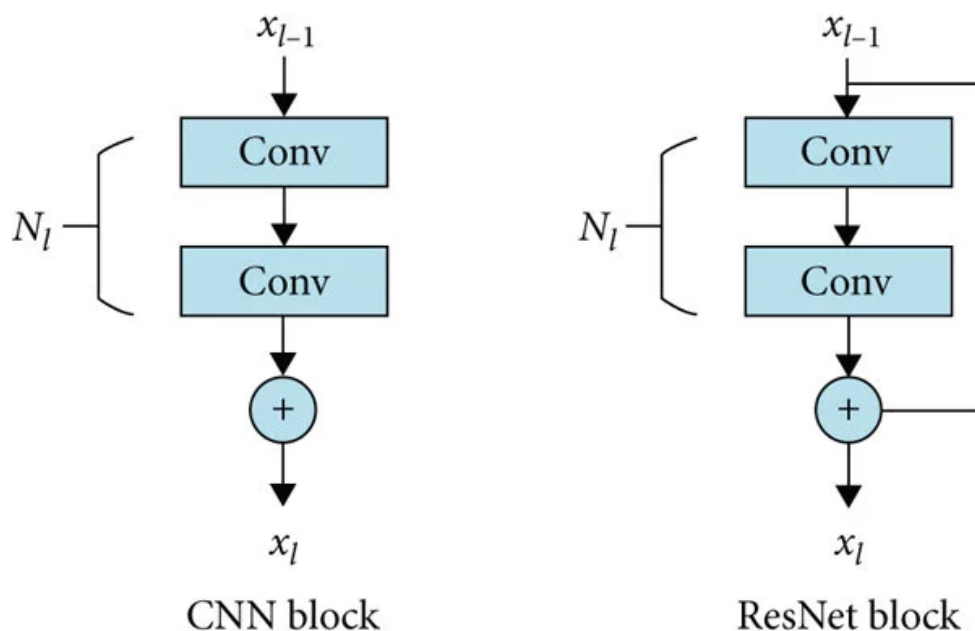


Рисунок 3.8 – Порівняння структури блоків CNN та ResNet

Основними перевагами архітектури ResNet є можливість навчання надзвичайно глибоких мереж, що містять 50, 101 чи навіть більше шарів, без ризику деградації моделі, оскільки градієнт може безперешкодно протікати через пропускні з'єднання назад до перших шарів системи.

Серед недоліків ResNet можна виділити збільшення вимог до обчислювальної пам'яті під час тренування, оскільки система змушена зберігати в пам'яті значення активацій з попередніх шарів для їхнього подальшого додавання, а також загальне зростання часу навчання при використанні великої кількості шарів.

Висновки до розділу 3

У третьому розділі було проаналізовано комплекс засобів розробки, використаних для створення автоматизованої системи класифікації травм, а також визначено конкретні завдання, для вирішення яких ці інструменти залучаються.

Було описано загальний механізм роботи згорткових мереж та пояснено залишкове відображення архітектури ResNet, разом із аналізом переваг та недоліків кожного з підходів.

Окрім теоретичного обґрунтування моделей глибокого навчання, у розділі було охарактеризовано набір прикладного програмного забезпечення. Зокрема, обґрунтовано вибір мови програмування Python як базової платформи для розробки, а також розглянуто спеціалізовані бібліотеки: PyTorch – для проектування, тренування та запуску нейромереж, та Scikit-learn – для обчислення фінальних метрик ефективності та генерації звітів. Проаналізовано роль інструментів Pydicom для безпечної роботи та попередньої анонімізації медичних файлів і NumPy – для високопродуктивних обчислень та матричних операцій із масивами даних. Додатково було розглянуто принципи побудови графічного інтерфейсу за допомогою фреймворку PyQt та реалізацію системи моніторингу за допомогою вбудованого модуля Logging.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Перед початком розробки програмного продукту важливо спланувати його архітектуру й варіанти використання, визначити його основні компоненти та зв'язки між ними. Щоб виконати таку задачу, необхідно використати різноманітні типи діаграм, наприклад, UML діаграма варіантів використання, яка допомогла спланувати та графічно показати доступні для користувача функції системи.

Разом з плануванням сценаріїв для користування продуктом, в даному розділі описано внутрішню будову програмної системи. Для цього розроблено загальну діаграму структури системи та окремих її модулів. На ній показано архітектурну композицію проекту, специфіку взаємодії логічних блоків та рух даних між ними. На відміну від класичної діаграми класів, цей підхід зосереджується на високорівневих компонентах і описує взаємозв'язки між графічним інтерфейсом, модулями обробки даних та модулем машинного навчання.

Оскільки основним елементом системи є машинне навчання, окремо наведено діаграми структур нейронних мереж використаних у системі. Вони показують внутрішню топологію моделей, конфігурацію їхніх шарів, а також як сигнали проходять через внутрішні блоки.

4.1 Взаємодія користувача та системи

Головним актором та цільовим користувачем розроблюваної системи є лікар-радіолог, який безпосередньо взаємодіє з графічним інтерфейсом для автоматизації свого щоденного робочого процесу. Другорядним користувачем виступає інженер-розробник, який використовує інтерфейс командного рядка для підготовки медичних даних та тренування нових моделей на додаткових наборах даних.

Для кращого розуміння доступного в системі функціоналу було створено UML діаграму варіантів використання зображену на рисунку 4.1.

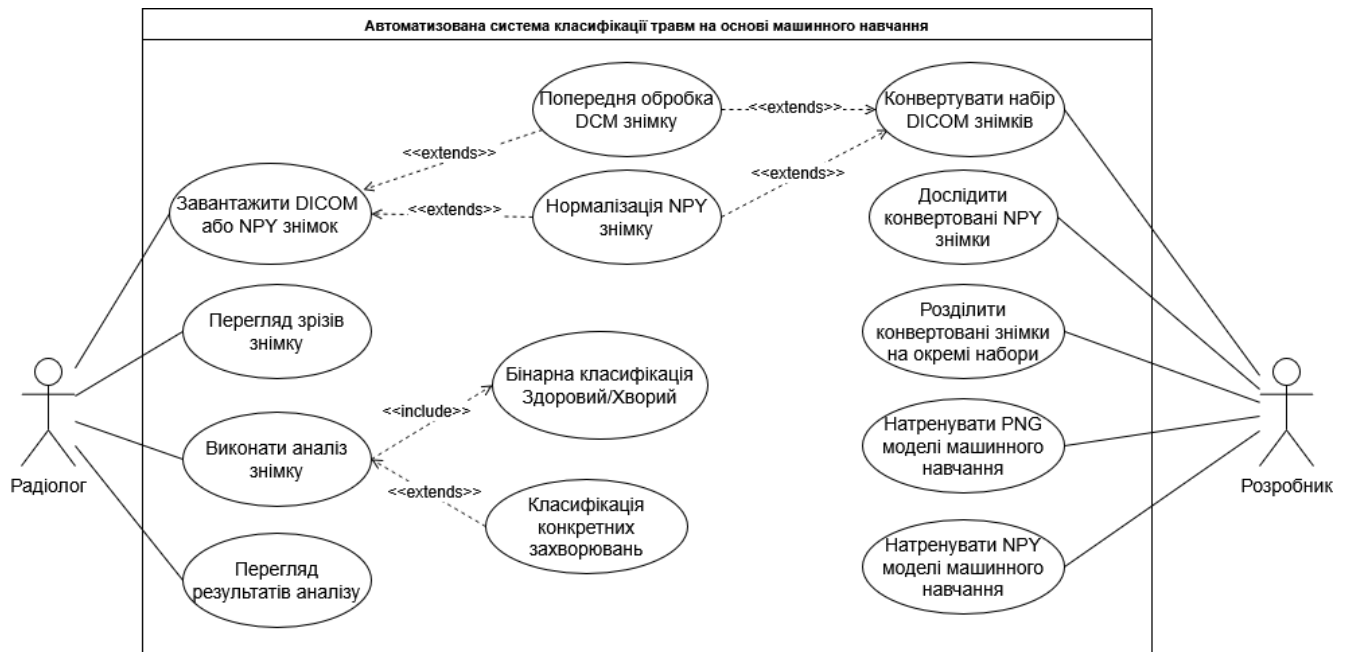


Рисунок 4.1 – Діаграма варіантів використання системи

Як видно з рисунку 4.1, користувачам із різними ролями доступний чітко розмежований функціонал. Для лікаря-радіолога графічний інтерфейс надає повний доступ до базових діагностичних функцій системи, а саме:

1. Завантаження та робота з медичними зображеннями формату DICOM або NumPy (NPY).
2. Пошаровий перегляд зрізів завантаженого об'ємного знімка коліна.
3. Запуск аналізу МРТ-знімка (класифікація травми).
4. Візуалізація та перегляд результатів проведеного аналізу.

На діаграмі також присутні зв'язки розширення («extends») для базової функції роботи з медичними зображеннями. У випадку завантаження файлу у сирому форматі DICOM система виконує повний цикл попередньої обробки DICOM-знімка, тоді як при імпорті знімка у вигляді готового NumPy-масиву запускається окремий процес нормалізації даних.

Процес патологоанатомічного аналізу, відповідно до архітектури системи, складається з двох послідовних етапів. Перший етап обов'язково включає (зв'язок «include») бінарну класифікацію для визначення стану суглоба (здоровий чи

хворий). Якщо коліно на знімку класифікується системою як хворе, відбувається перехід на другий етап – класифікацію конкретних травм і патологій. Такий розподіл не дозволяє виконувати зайві обчислення для завідомо здорових суглобів, що пришвидшує час відгуку системи.

Для користувача-розробника консольний CLI-інтерфейс надає доступ до інженерного функціоналу, який включає:

1. Конвертацію наборів DICOM-зображень.
2. Дослідження отриманих після конвертації NPY-масивів зображень.
3. Розділення конвертованих даних на окремі вибірки для машинного навчання (навчальну, валідаційну та тестову).
4. Тренування ML-моделей, що працюють із PNG-зображеннями.
5. Тренування ML-моделей, що працюють з багатовимірними NPY-масивами зображень.

Таким чином, архітектура системи чітко ізолює складні інженерні інструменти розробника від робочого простору лікаря-радіолога. Таке відокремлення є корисним із двох основних причин:

1. Приховування складнішого консольного функціоналу від кінцевого користувача, який з високою вірогідністю не має потрібних знань для роботи з ним.
2. Надання можливості натренувати власну ML-модель для роботи з іншим набором даних.

Якщо перший пункт робить графічний інтерфейс простішим, то другий робить створену систему відкритою до модифікацій безпосередньо на місці її розгортання. Це дозволяє за потреби адаптувати розроблену систему під окремі потреби цільового користувача або медичного закладу, що робить її гнучкішою у використанні.

4.2 Архітектура системи

Для відображення логічно-зв'язної архітектури розробленої системи та візуалізації взаємодії її внутрішніх компонентів було створено відповідну діаграму, що зображена на рисунку 4.2.

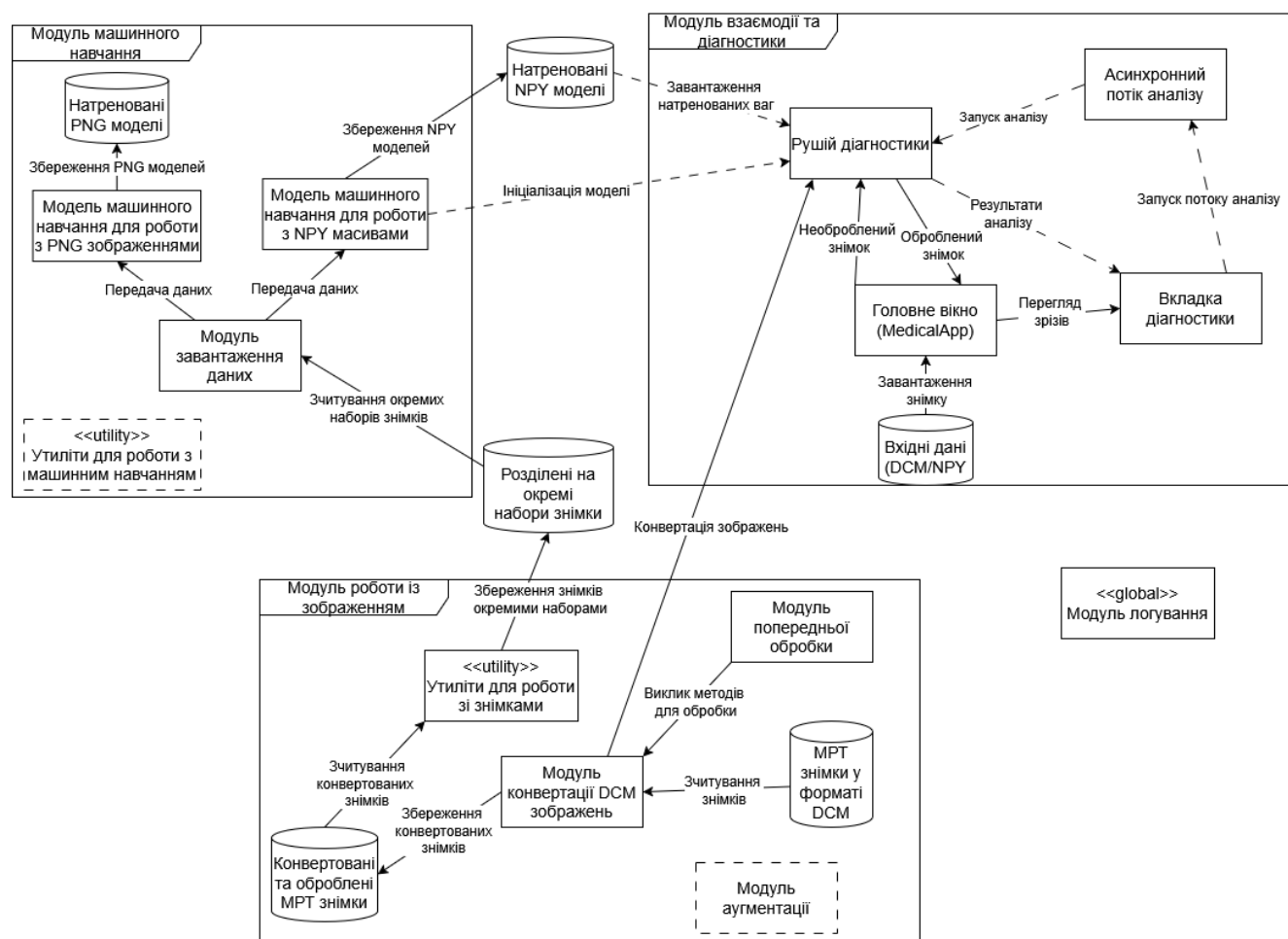


Рисунок 4.2 – Архітектура автоматизованої системи класифікації травм

Система складається з чотирьох програмних модулів:

1. Модуль роботи із зображеннями.
2. Модуль машинного навчання.
3. Модуль логування.
4. Модуль взаємодії та діагностики (графічного інтерфейсу).

Модулі роботи із зображеннями та машинного навчання мають слабку залежність через спільний стан – підготовлені та розділені на окремі набори дані,

які зберігаються на локальному диску. Перший з них відповідає за зчитування сирих МРТ-знімків у форматі DICOM, їхню попередню обробку, конвертацію та аугментацію за допомогою створених утиліт. Натомість модуль машинного навчання фокусується виключно на отриманні вже очищених даних через модуль завантаження та безпосередньому тренуванні PNG або NPY-моделей із подальшим збереженням їхніх ваг.

Модуль взаємодії та діагностики, окрім спільного стану з модулем машинного навчання (натреновані моделі), також ініціалізує класи з інших підсистем та використовує їхні методи для реалізації власного функціоналу. Основним ядром тут виступає рушій діагностики, що безпосередньо взаємодіє з головним вікном програми MedicalApp та вкладкою діагностики. Щоб уникнути зависання інтерфейсу під час виконання аналізу, його запуск винесено в окремий асинхронний потік.

Модуль логування на діаграмі архітектури має позначення глобального компонента («global»), оскільки він імпортується та паралельно використовується усіма іншими модулями для моніторингу роботи системи.

CLI-інтерфейс для інженера-розробника не має окремого ізольованого модуля, адже він функціонує як консольна надбудова, що дозволяє централізовано викликати утиліти обробки та функції для повного циклу тренування ML-моделей безпосередньо з командного рядка.

Після ознайомлення із загальною архітектурою системи можна перейти до детального розбору структури та логіки роботи кожного окремого модуля.

4.2.1 Модуль роботи із зображеннями

Першим структурним компонентом, з яким взаємодіє система під час циклу підготовки даних, є модуль роботи із зображеннями (рис. 4.3).

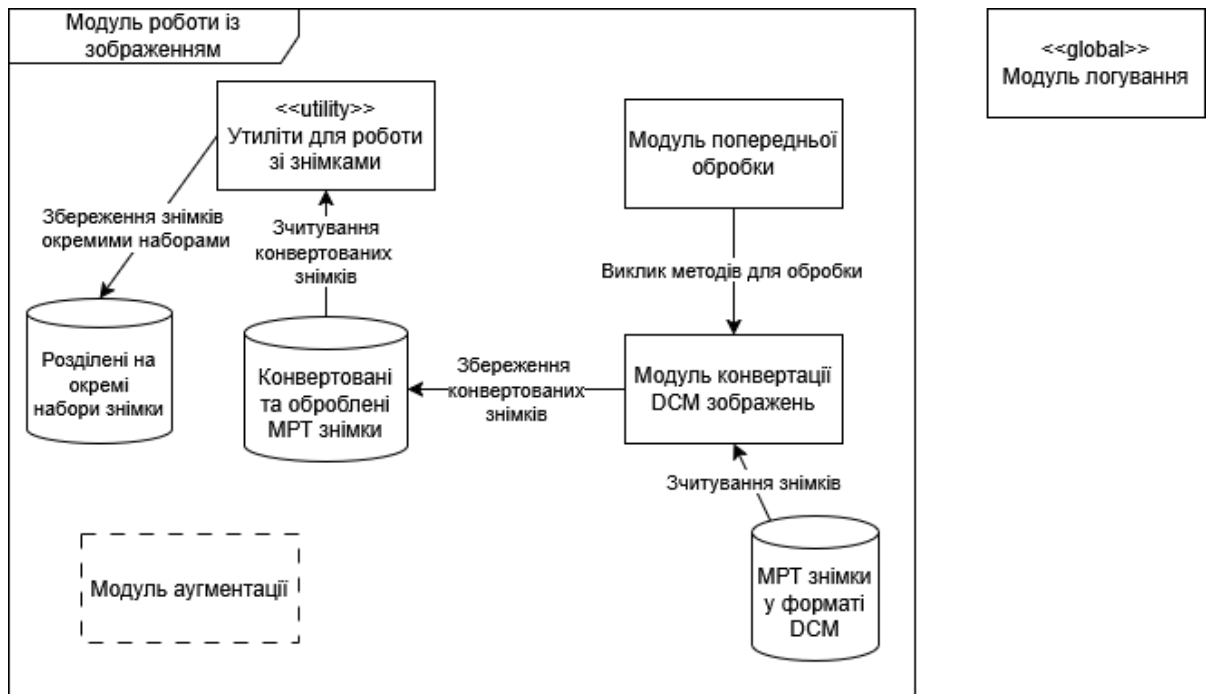


Рисунок 4.3 – Архітектура модуля роботи із зображеннями

Він відповідає за повний цикл підготовки навчальних даних. Спочатку система зчитує сирі файли у форматі DICOM, після чого викликає методи модуля попередньої обробки для очистки та уніфікації зображень. Оброблена інформація конвертується та зберігається у вигляді NPY-масивів або PNG-файлів у проміжних сховищах диска. Наступним кроком є запуск спеціалізованих утиліт для автоматичного розділення отриманих знімків на три незалежні вибірки: тренувальну, валідаційну та тестувальну. Весь процес роботи підсистем фіксується глобальним модулем логування.

Головну роль в уніфікації вхідних даних відіграє модуль попередньої обробки, який виконує низку послідовних математичних та геометричних трансформацій над завантаженими знімками. По-перше, здійснюється вейвлет-фільтрація об'єму для усунення специфічного для МРТ-апарату шуму. По-друге, алгоритм виділяє область інтересу на знімку шляхом обчислення обмежувальної рамки й відсікає зайвий порожній простір навколо коліна. Наступним кроком є просторовий ресемплінг за допомогою трилінійної інтерполяції для приведення фізичного розміру вокселів до єдиного стандарту. Фінальний етап обробки реалізує

просторову нормалізацію 3D-тензорів за допомогою симетричного додавання нулів або центрального обрізання, що гарантує сувору відповідність розмірності вхідних даних архітектурним вимогам нейромережі. Приклад конвертованого PNG зрізу, який отримується в кінці циклу підготовки даних наведено на рисунку 4.4.



Рисунок 4.4 – PNG зображення зрізу отримане в кінці циклу підготовки даних

Варто зазначити, що у структурі архітектури також присутній ізольований модуль аугментації зображень, призначений для штучного розширення обсягу навчального набору даних шляхом випадкових геометричних трансформацій, додавання шумів та лінійної зміни контрастності. Проте під час практичного тестування було виявлено, що на використаних при розробці даних цей компонент не надає очікуваного захисту від перенавчання, але при цьому суттєво збільшує час, необхідний для тренування мережі через високе навантаження на ресурси обладнання.

Програмна реалізація конвеєра аугментації даних розподілена на два автономні потоки обробки залежно від цільового формату збереження: двовимірних PNG-зрізів та багатовимірних числових масивів. Перед початком генерації модифікованих копій для графічних файлів викликається спеціалізований метод фільтрації інформаційних даних, який відсікає пусті зрізи.

Для обох типів вибірок накладаються ідентичні геометричні й фотометричні трансформації, що включає базові операції та їхні поєднання. До базового набору відносяться: дзеркальне відображення по горизонталі, варіація рівнів яскравості, а також накладання адитивного гауссового шуму для симуляції апаратних дефектів сканування.

4.2.2 Модуль машинного навчання

Наступним компонентом системи є модуль машинного навчання, архітектура якого представлена на рисунку 4.5. Даний модуль відповідає за безпосередній процес побудови, налаштування та навчання моделей на основі раніше підготовлених даних.

Розділені набори знімків потрапляють у модуль завантаження даних. Система спроектована таким чином, щоб підтримувати паралельну обробку двох типів даних: PNG-зображень та NumPy-масивів. Це дозволяє тренувати специфічні моделі для кожного формату, які, після тренування, зберігаються у відповідних сховищах. Весь процес супроводжується логуванням за допомогою глобального модуля логування.

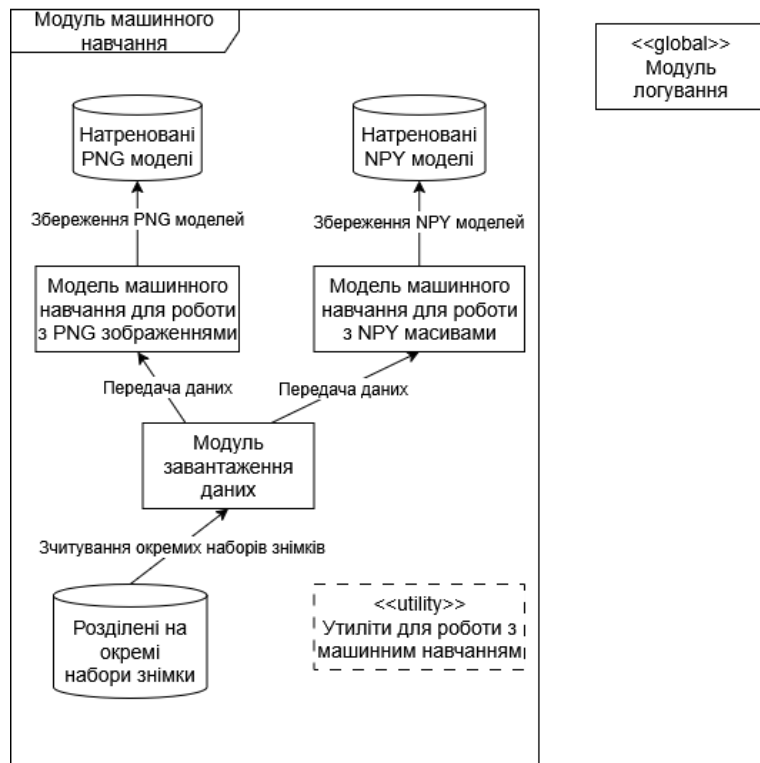


Рисунок 4.5 – Архітектура модуля машинного навчання

Головним елементом підсистеми завантаження є універсальний клас KneeDataset, який повністю автоматизує сканування структури файлів, зчитування інформації та призначення папкам категорій за допомогою числових міток (рис. 4.6).

```
# Binary classification mapping
if self.stage == 1:
    self.group_map = {
        'healthy': 0,
        'гонартроз': 1,
        'хондромаліяція виростків': 1,
        'хондромаліяція надколінка': 1,
        'меніски': 1,
        'часткове пошкодження пхз': 1,
        'медіапательна складка': 1
    }
    self.classes = ['Healthy', 'Pathology']
elif self.stage == 2:
    self.group_map = {
        'гонартроз': 0,
        'хондромаліяція виростків': 1,
        'хондромаліяція надколінка': 2,
        'меніски': 3,
        'часткове пошкодження пхз': 4,
        'медіапательна складка': 5
    }
    self.classes = [
        'Гонартроз', 'Хондромаліяція_виростків', 'Хондромаліяція_надколінка',
        'Меніски', 'Часткове_пошкодження_пхз', 'Медіапательна_складка'
    ]
```

Рисунок 4.6 – Призначення папкам категорій числових міток класів

Особливістю цього класу є підтримка дворівневої логіки діагностики. На першому етапі відбувається динамічне групування всіх виявлених патологій у єдиний узагальнений клас для проведення бінарної класифікації (здоровий / хворий). На другому етапі конвеєр автоматично пакує дані для детального розпізнавання шести окремих типів травм колінного суглоба. Для пришвидшення навчання, в архітектурі реалізовано механізм кешування тензорів у оперативній пам'яті, для зменшення циклів звернення до диска, а також інтегровано процедуру трансформацій «на льоту» з випадковим обрізаннями, поворотами та змінами яскравості для захисту PNG-моделей від перенавчання.

Для створення фінальних ітераторів у модулі було створено централізовану функцію `load_dataset` (рис. 4.7), яка вирішує низку завдань щодо стабілізації обчислювального процесу.

```
def load_dataset(  
    base_data_path: str | Path, batch_size: int = 16,  
    mode: str = "png", stage: int = 1, cache_in_ram: bool = False  
):  
    """  
    Creates and returns DataLoader objects for training, validation, and test sets.  
  
    Key features:  
    1. Automatic path splitting: base_path/{split}/{mode}.  
    2. Class balancing: Uses WeightedRandomSampler for training set, <br>  
    which solves the imbalance between healthy cases and rare pathologies.  
    3. Optimization: Adjusts pin_memory and num_workers depending on RAM cache usage.  
  
    Returns:  
    |   output (tuple): (train_loader, val_loader, test_loader, list_of_classes)  
    """
```

Рисунок 4.7 – Функція «load_dataset»

Оскільки медичні вибірки мають виражений дисбаланс (кількість знімків здорових суглобів суттєво перевищує кількість знімків із патологіями), функція автоматично обчислює обернені вагові коефіцієнти для кожного класу та створює збалансований семплер `WeightedRandomSampler`. Через нього, під час формування партій для тренувального завантажувача, модель буде отримувати рівномірну кількість об'єктів усіх категорій, у такий спосіб запобігаючи сильному зміщенню ваг мережі у бік класу з більшою кількістю даних. Функція додатково оптимізує

параметри виділення потоків та утилізації пам'яті GPU, адаптуючись до поточного режиму використання оперативної пам'яті.

4.2.3 Модуль взаємодії та діагностики

Останнім компонентом системи є модуль взаємодії та діагностики, архітектура якого наведена на рисунку 4.6. Даний модуль надає зручний інтерфейс для користувача та координує роботу між графічною оболонкою і рушієм аналізу.

Процес роботи починається із завантаження вхідних даних у форматах DICOM або NPY через головне вікно програми MedicalApp. Після вибору файлу додаток взаємодіє з рушієм діагностики, який відповідає за попередню обробку зображень та ініціалізацію відповідних моделей зі сховища натренованих NPY-моделей. Підготовлені дані відображаються у вкладці діагностики, де користувач може переглядати окремі пошарові зрізи МРТ-знімка за допомогою повзунка навігації.

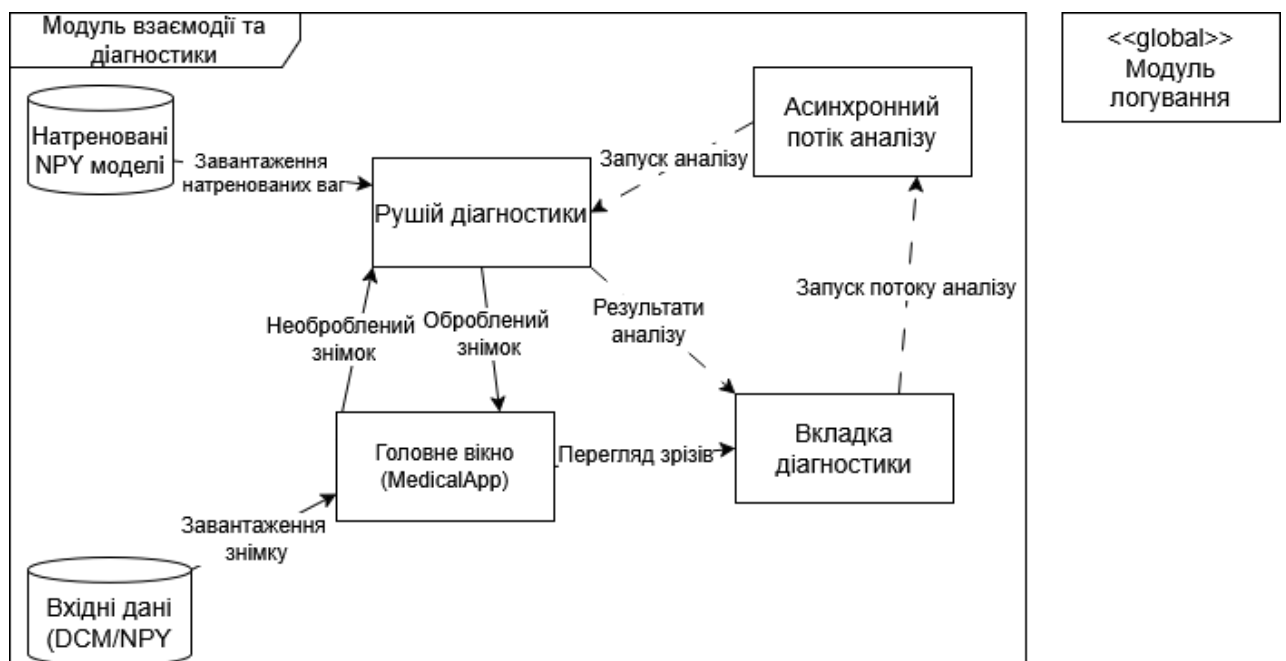


Рисунок 4.8 – Архітектура модуля взаємодії та діагностики

З інженерної точки зору архітектура інтерфейсу побудована наближено до шаблону «модель-вигляд-контролер» (англ. Model-view-controller), де клас

головного вікна виступає центральним оркестратором та менеджером стану системи, а вкладка діагностики відповідає за безпосереднє візуальне відображення. Візуальна вкладка розподілена на дві функціональні зони: ліва панель керування містить інтерактивні блоки запуску аналізу та віджет виведення результатів першого етапу аналізу, під яким розташована матриця індивідуальних прогрес-барів для відображення ймовірнісних оцінок шести патологій другого етапу аналізу. Права панель повністю відведена під візуалізацію шарів МРТ-скану із зображенням фіксованого розміру 512 на 512 пікселів. Синхронізація між положенням повзунка та виведенням поточної матриці зрізу здійснюється через спеціалізовану утиліту, яка виконує швидке низькорівневе перетворення сирого двовимірного масиву NumPy в об'єкт QImage у форматі 8-бітного зображення у відтінках сірого із застосуванням згладжувальної трансформації.

Головною особливістю модуля є повна асинхронність виконання діагностики. Оскільки робота глибоких нейромереж із тривимірними тензорами сильно навантажує апаратні ресурси, запуск обчислень у головному потоці неминуче призвів би до повного заморожування інтерфейсу. Для запобігання цьому в системі створюється фоновий робочий потік. При натисканні кнопки запуску аналізу головне вікно блокує повторні кліки, змінює текстовий статус програми та делегує вхідний тензор у фоновий потік. Потік викликає методи класифікації рушія KleeInferenceEngine, виконує апаратну синхронізацію CUDA-ядер за допомогою PyTorch та, після успішного завершення розрахунків, використовує механізм сигналів PyQt і повертає структурований словник результатів назад у головний потік. Це дозволяє динамічно оновити індикатори виконання та статус діагностики в реальному часі. Весь описаний життєвий цикл взаємодії компонентів фіксується глобальним модулем логування.

4.3 Структура нейронних мереж

Під час розробки системи були спробовані різні варіанти архітектури нейронних мереж як для двовимірних PNG-зображень (2D-мережі), так і для тривимірних NPY-масивів (3D-мережі). На початковому етапі за допомогою інструментів PyTorch створювалися власні нестандартні нейромережі, що дозволило ознайомитися з особливостями роботи фреймворку. Паралельно було розроблено базові функції для запуску процесу тренування моделей, алгоритми для їхньої оцінки, а також реалізовано механізм ранньої зупинки навчання при тривалій відсутності прогресу на валідаційній вибірці, що дозволило суттєво економити час на тренуваннях.

Саме на цьому етапі було виявлено головну проблему системи – швидке перенавчання ML-моделей через невеликий обсяг доступних даних. Усе подальше проектування та вибір архітектур центрувалися навколо цього обмеження.

Після визначення основної проблеми, було прийнято рішення використати вже готову модель для роботи із зображеннями і відкалібрувати її для класифікації МРТ-знімків колінного суглобу. Для цього я вибрав модель ResNet, а саме ResNet-18 бо ця версія моделі краще працює з невеликими обсягами даних. ResNet було обрано через механізм пропускних з'єднань, що допомагає запобігти проблемі зникнення градієнта і через це покращити ситуацію з перенавчанням. Її було використано при розробці 2D мережі, що дозволило ознайомитися з процесом тонкого настроювання (англ. fine-tuning) попередньо натренованої моделі та способами якими це можна зробити.

Проте під час тестування значним недоліком такого підходу стало жорстке обмеження у можливостях структурної модифікації мережі. Зважаючи на це, для 3D-мережі було вирішено реалізувати власну архітектуру, що наслідує концепцію пропускних з'єднань сімейства ResNet, проте не має обмежень на модифікації, що дозволяє вільно налаштовувати параметри під особливості використовуваних медичних даних.

4.3.1 2D-мережа

Першою було реалізовано мережу, що працює з двовимірними PNG-зрізами. Було спроектовано модель KneeResNet на базі класичної архітектури ResNet-18. Схему архітектури оригінального ResNet-18 наведено на рисунку 4.9 [16].

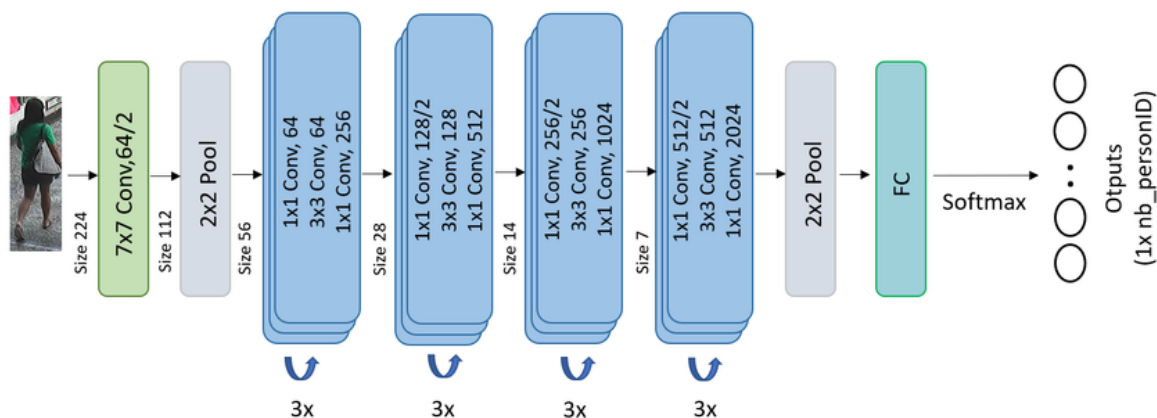


Рисунок 4.9 – Архітектура ResNet-18

З метою адаптації базової моделі до роботи з МРТ-знімками було проведено структурну модифікацію, яка складається з двох основних етапів. По-перше, перший згортковий шар було замінено на новий еквівалент з ідентичними геометричними параметрами (ядро згортки 7 на 7, крок 2), але він приймає лише один вхідний канал для коректної обробки чорно-білих МРТ-сканів. По-друге, стандартний повнозв'язний шар (FC) було повністю демонтовано, а замість нього інтегровано власний глибокий класифікатор. Нова аналітична голова складається з лінійного шару на 256 нейронів, блоку пакетної нормалізації, функції активації ReLU та шару регуляризації Dropout для запобігання перенавчанню (коефіцієнт відсікання встановлюється на рівні 0.4 для бінарного аналізу та піднімається до 0.7 під час складного диференціального діагностування).

Для вирішення проблеми незбалансованості класів у наборі даних, замість стандартної крос-ентропії було програмно реалізовано модифіковану функцію втрат FocalLoss. Її математична логіка базується на динамічному зменшенні ваги легко класифікованих (фонових) об'єктів та фокусуванні обчислювального

градієнта на важких, помилково розпізнаних зразках патологій через коефіцієнт згасання γ (формула 4.1).

$$FL = (1 - p_t)^\gamma \cdot CE, \quad (4.1)$$

де p_t – оцінена моделлю ймовірність для правильного класу ($p_t = e^{-CE}$),

γ – коефіцієнт фокусування, що регулює ступінь зменшення ваги простих прикладів.

CE – значення стандартної крос-ентропії для поточного зразка.

Додатково у функцію інтегровано механізм згладжування міток, що штучно розмиває жорсткі межі цільових векторів (на рівні 0.05 для першого етапу та 0.3 для другого), що підвищує здатність мережі до узагальнення на нових клінічних даних. Контроль за процесом навчання здійснюється власним класом EarlyStopping. Цей монітор виконує ранню зупинку оптимізації за двома незалежними критеріями:

1. За відсутності прогресу у зменшенні функціональної втрати на валідаційному наборі даних впродовж заданої кількості епох.
2. Коли числовий розрив між точністю тренувального та валідаційного наборів перевищує поріг у 25%, фіксуючи початок деградації та перенавчання моделі.

Програмна архітектура підтримує два повністю автономні конвеєри запуску глибокого навчання:

1. Первинний конвеєр (класифікація норма / патологія).
2. Вторинний конвеєр (класифікація кожної окремої патології).

Первинний конвеєр відповідає за тренування бінарного класифікатора першого етапу (норма / патологія). Навчання починається у режимі замороженого «кістяка» (англ. backbone), проте за допомогою методу розморозки шарів реалізує динамічне покрокове розморозування глибинних згорткових блоків (шар №4 та шар №3). Це відбувається автоматично, як тільки лінійний класифікатор стабілізується, а лічильник ранньої зупинки досягає п'яти епох без покращень. При

розморожуванні шарів, конвеєр миттєво перебудовує оптимізатор AdamW та скидає планувальник CosineAnnealingLR для делікатного калібрування ваг. У таблиці 4.1 наведено результати оцінки роботи моделі бінарної класифікації для PNG зображень на тестовому наборі даних.

Таблиця 4.1 – Оцінка моделі бінарної класифікації PNG зображень

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Здорові	0.78	0.88	0.83	1568
Патологія	0.76	0.61	0.68	992
Загальна точність	–	–	0.77	2560
Середнє макро	0.77	0.74	0.75	2560
Середнє зважене	0.77	0.77	0.77	2560

Вторинний конвеєр реалізує концепцію послідовного передавального навчання (англ. Sequential Transfer Learning) для диференціальної діагностики шести специфічних патологій. Він виконує операцію «модельної хірургії»: повністю імпортує натреновані на першому етапі ваги екстрактора ознак, після чого замінює лише фінальний лінійний шар під нову розмірність шести цільових класів. Обчислювальний процес оптимізовано за допомогою технології змішаної точності (англ. Mixed Precision) від компанії NVIDIA, що дозволяє зменшити використання відеопам'яті під час розрахунків. Після завершення кожного конвеєра викликається функція комплексної оцінки, яка за допомогою метрик Scikit-learn записує кінцевий звіт у загальний журнал системи. У таблиці 4.2 наведено оцінку роботи моделі диференціальної класифікації PNG зображень на тестовому наборі даних.

Таблиця 4.2 – Оцінка моделі диференціальної класифікації PNG зображень

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Гонартроз	0.37	0.38	0.37	160
Хондромаліяція виростків	0.05	0.01	0.02	160
Хондромаліяція надколінка	0.00	0.00	0.00	160

Продовження таблиці 4.2

1	2	3	4	5
Меніски	0.04	0.03	0.03	160
Часткове пошкодження пхз	0.00	0.00	0.00	160
Медіапателярна складка	0.16	0.55	0.25	192
Загальна точність	–	–	0.17	992
Середнє макро	0.10	0.16	0.11	992
Середнє зважене	0.11	0.17	0.12	992

Як видно з таблиці 4.2, точність диференціальної класифікації дуже низька порівняно з бінарною класифікацією. Причини цього детальніше описані в підрозділі 4.4.

4.3.2 3D-мережа

Після розробки 2D-мережі, отримані знання були використані для розробки власної тривимірної нейронної мережі KneeResidualAttentionNet, яка працює безпосередньо з об'ємними NumPy-масивами. Ця архітектура демонструє найкращу узагальнювальну здатність серед усіх протестованих моделей, через що її кінцеві ваги інтегровані у графічний інтерфейс системи як основне діагностичне ядро. Детальну структурно-логічну схему розробленої мережі наведено на рисунку 4.10.

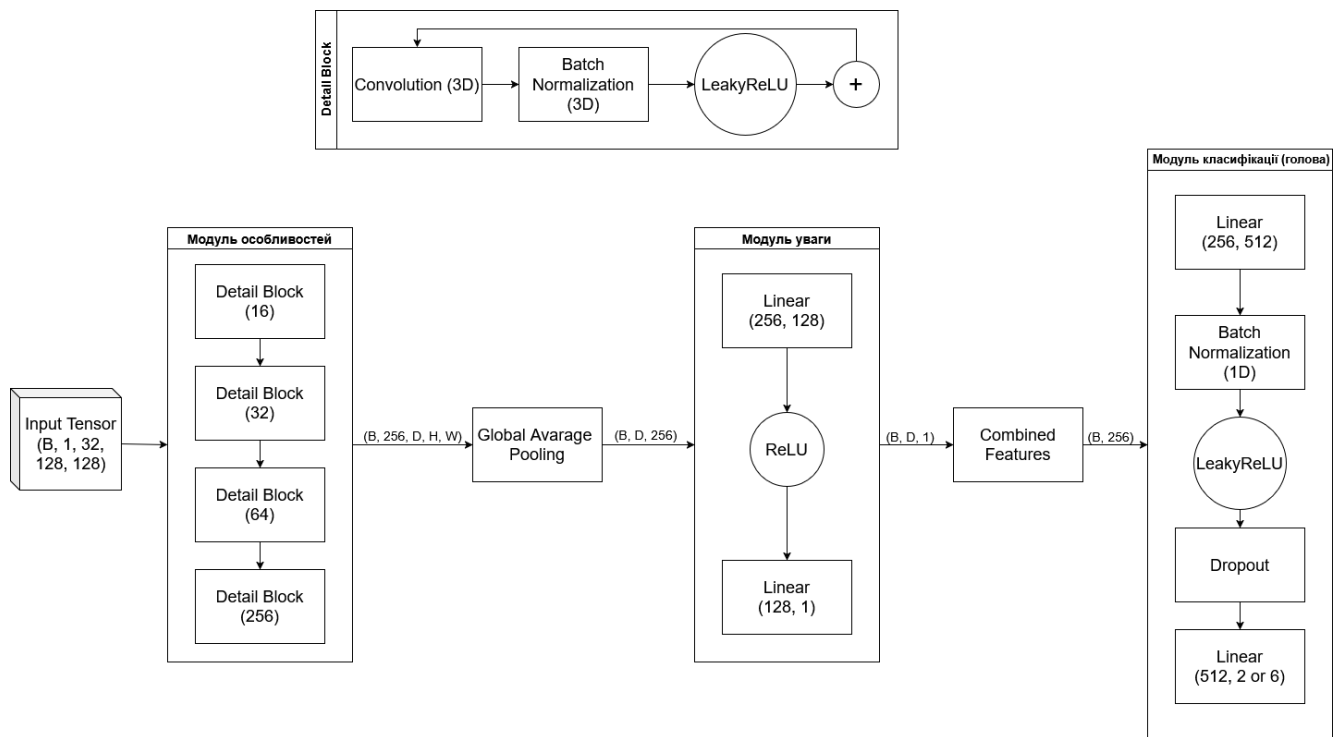


Рисунок 4.10 – Схема структури нейронної мережі для класифікації тривимірних масивів даних

Вхідний тривимірний тензор МРТ-скану повинен бути розмірністю $(B, 1, 32, 128, 128)$, де параметри відповідають розміру партії, кількості каналів, глибині, висоті та ширині зрізів. Він передається у модуль виділення ознак. Тут дані послідовно проходять через каскад із чотирьох блоків залишкового навчання `DetailBlock3D` із прогресуючим нарощуванням кількості фільтрів: від 16 до 32, 64 та 256 на фінальному рівні. Кожен такий блок реалізує класичну концепцію залишкових мереж, поєднуючи тривимірну згортку `Conv3D` із розміром ядра 3 на 3 на 3, пакетну нормалізацію `BatchNorm3D` та функцію активації `LeakyReLU` з коефіцієнтом нахилу 0.1. Для стабілізації проходження градієнта інтегровано механізм прямого зв'язку. Якщо вхідна та вихідна розмірності шару всередині блоку не збігаються через зміну кроку згортки, конвеєр автоматично застосовує проєкційну згортку з ядром 1 на 1 на 1 для лінійного вирівнювання просторових карт ознак.

Після отримання високорівневих ознак, тензор розмірністю $(B, 256, D, H, W)$ стискається через шар глобального тривимірного усередненого пулінгу до стану $(B, 256, D)$. Наступним етапом є проходження даних через унікальний модуль уваги, що фокусується на глибинних зрізах томографії. Масив транспонується у розмірність $(B, D, 256)$, після чого два послідовні лінійні шари з проміжною функцією активації ReLU стискають ознаки, формуючи вектор розподілу ваг уваги розміром $(B, D, 1)$. Пропускання цих ваг через функцію Softmax дозволяє мережі оцінити та зважити інформаційну цінність кожного окремого фізичного зрізу МРТ-суглоба. Кінцевий агрегований вектор ознак обчислюється через математичне зважене підсумовування, генеруючи компактну матрицю $(B, 256)$.

Фінальний етап класифікації виконується у модулі класифікації, який працює як «голова» мережі, куди передаються комбіновані ознаки. Перший лінійний шар розгортає ознаки до 512 елементів, після чого вони проходять крізь одновірну пакетну нормалізацію та шар випадіння Dropout для придушення ефекту перенавчання на малих вибірках. Архітектура використовує технологію «модельної хірургії». Під час першого етапу вихідний лінійний шар конфігурується на два класи для швидкої бінарної класифікації норми від патології. У разі переходу до диференціальної діагностики специфічних патологій блоки екстрактора ознак повністю заморожуються, а фінальний шар замінюється на нову лінійну структуру з шістьма виходами під конкретні захворювання, що реалізує механізм передавального навчання.

Результати оцінки роботи моделей KneeResidualAttentionNet на тестовому наборі даних наведені в таблиці 4.3 для бінарної класифікації та 4.4 для диференціальної класифікації.

Таблиця 4.3 – Результати оцінки роботи моделі для бінарної класифікації NPY масивів

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Здорові	0.81	0.96	0.88	49
Патологія	0.91	0.65	0.75	31
Загальна точність	—	—	0.84	80
Середнє макро	0.86	0.80	0.82	80
Середнє зважене	0.85	0.84	0.83	80

Таблиця 4.4 – Результати оцінки роботи моделі для диференціальної класифікації NPY масивів

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Гонартроз	0.60	0.60	0.60	5
Хондромаліяція виростків	0.10	0.20	0.13	5
Хондромаліяція надколінка	0.20	0.20	0.20	5
Меніски	0.00	0.00	0.00	5
Часткове пошкодження пхз	0.33	0.20	0.25	5
Медіапательна складка	0.29	0.33	0.31	6
Загальна точність	—	—	0.26	31
Середнє макро	0.25	0.26	0.25	31
Середнє зважене	0.25	0.26	0.25	31

За результатами оцінок, моделі що працюють з тривимірними NumPy масивами загалом показують вищі показники практично за всіма параметрами. Але вони мають спільні з 2D моделями проблеми.

4.4 Проблеми розроблених мереж

Усі розроблені мережі мають спільні проблеми, що негативно впливають на точність їхньої роботи. У розробленій системі можна виділити дві основні проблеми:

1. Нестача даних для навчання.
2. Стирання ознак на знімках.

4.4.1 Нестача даних

Основна проблема – це невелика, як для машинного навчання, кількість даних у наборі хвороб. На рисунку 4.11 наведено розподіл даних, використаних для тренування мереж.

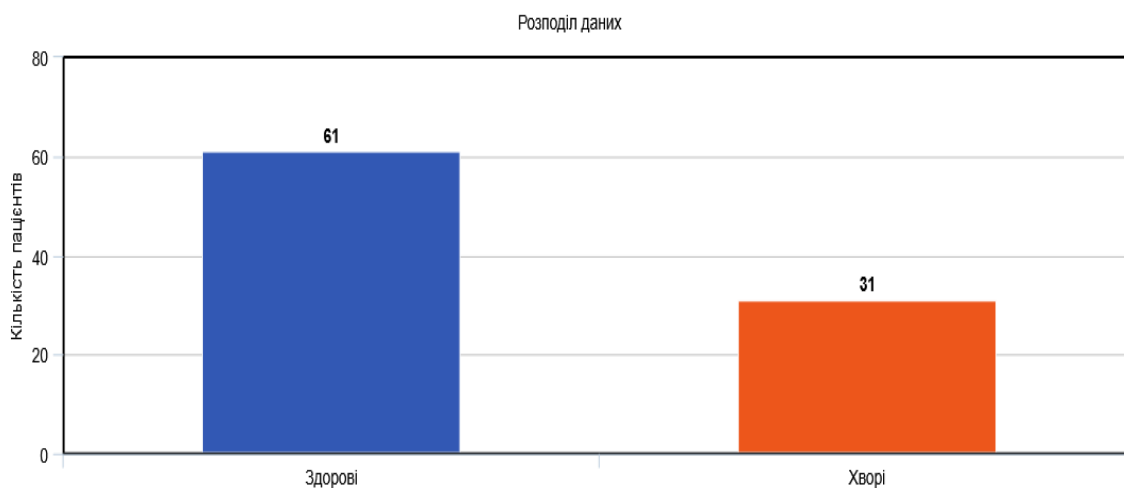


Рисунок 4.11 – Розподіл даних, використаних для тренування ML-мереж

Як видно з розподілу на рисунку 4.11, кількість пацієнтів у наборі даних зі знімками здорових колін в рази більша, ніж у наборі даних із патологіями. 61 пацієнт – це лише частина від всього набору даних зі здоровими знімками, бо під час розробки довелося розробити механізм обмеження на кількість здорових пацієнтів, яких можна взяти для машинного навчання з метою уникнення дуже сильного дисбалансу даних на користь здорових знімків.

Натомість 31 пацієнт у наборі даних патологій – це всі пацієнти, що присутні в усіх шести класах, за якими відбувається класифікація, і де вони також нерівномірно розподілені (рисунок 4.12).

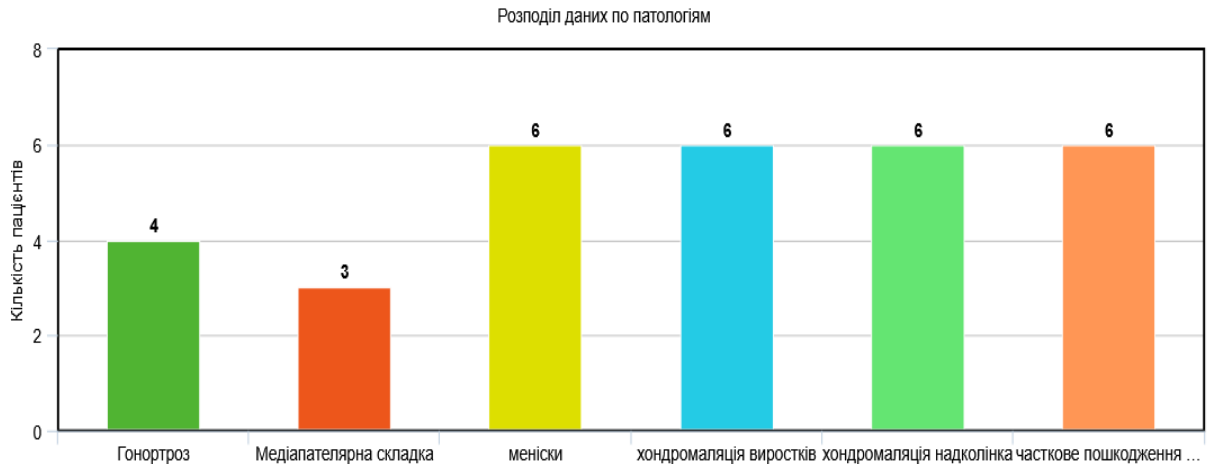


Рисунок 4.12 – Розподіл пацієнтів за класами в наборі даних патологій

Але варто зазначити, що насправді пацієнтів у наборі даних патологій ще менше, бо є пацієнти, які повторюються в різних класах (рисунок 4.13). Без урахування дублікатів кількість пацієнтів у наборі даних патологій становить 27.

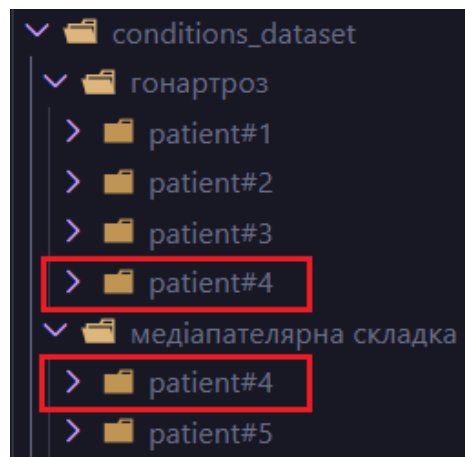


Рисунок 4.13 – Приклад дублювання пацієнтів між класами патологій

Оскільки це медичний набір даних, то такі повторення означають, що пацієнт має декілька патологій відразу. Проте при машинному навчанні з такою малою кількістю даних це суттєво збільшує схильність моделі до перенавчання, де вона

перестає узагальнювати ознаки, а просто запам'ятовує вигляд коліна конкретних пацієнтів.

Через дисбаланс на користь здорових знімків оцінка моделі для бінарної класифікації показує, що параметр повноти для здорових знімків значно більший, ніж для хворих (таблиця 4.3). Але вона все одно класифікує знімки з точністю у 84% через достатню кількість даних в обох наборах разом, проте із нахилом частіше класифікувати знімки як здорові.

У моделі для диференціальної класифікації через нестачу даних для навчання загалом точність не перевищує вірогідності випадкового вгадування. Але на додачу до нестачі даних у наборі, що в 2D-моделі, що в 3D-моделі присутні класи, які взагалі ніколи не були класифіковані. Це пов'язано з другою проблемою – втратою даних на знімках.

4.4.2 Втрата даних на знімках

Під час конвертації знімків із формату DICOM було виявлено, що деякі знімки складаються переважно з порожнечі. Такі знімки було названо розвідними, і реалізовано функціонал для ігнорування таких знімків (рисунок 4.14).

```
batch_conversion.WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm  
batch_conversion.WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm  
batch_conversion.WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm  
batch_conversion.WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm  
batch_conversion.WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm
```

Рисунок 4.14 – Записи про пропускання розвідних знімків під час конвертації

Також у деяких DICOM-знімках була пошкоджена структура файлу або відсутні теги, що позначають початок даних зображення (рисунок 4.15). Такі знімки також пропускаються під час конвертації.

```
load_file.WARNING: Corrupted structure in IMG-0002-00001.dcm: No tag to read at file position 1004898  
load_file.WARNING: Skipping non-image DICOM (Report/SR): IMG-0005-00001.dcm
```

Рисунок 4.15 – Записи про знімки з пошкодженою структурою файлу або відсутнім зображенням

Такий підхід зменшив загальну кількість доступних для навчання даних, але збільшив їхню якість. Проте навіть уже конвертовані знімки мають свої проблеми. Після процесу попередньої обробки на деяких знімках з'явилися графічні артефакти (рисунок 4.16). Причиною таких артефактів є функція для зменшення МРТ-шуму на знімках, яка використовує вейвлет-фільтрацію, але експерименти з різними конфігураціями фільтрації не показали суттєвих покращень.

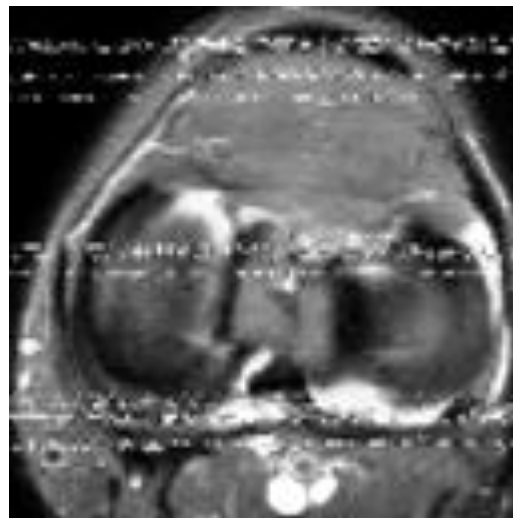


Рисунок 4.16 – Артефакти після попередньої обробки

Також варто зазначити, що більшість знімків стискаються до розміру (32, 128, 128) для 3D ML-моделей і до (128, 128) для 2D-моделей відповідно. Це призводить до «затирання» й так малопомітних ознак, таких як, наприклад, пошкодження менісків. Таке стиснення зумовлене обмеженнями апаратного забезпечення, на якому розроблялася система, та необхідністю прискорити тренування моделей.

Висновки до розділу 4

У четвертому розділі було розглянуто та описано програмну реалізацію автоматизованої системи класифікації травм, починаючи з проектування і закінчуючи аналізом критичних обмежень розроблених алгоритмів.

На основі побудованої UML-діаграми варіантів використання було сплановано взаємодію ключових користувачів із системою, що дозволило розмежувати повсякденний діагностичний функціонал для лікаря-радіолога (GUI) від інженерного інструментарію розробника (CLI). Описано загальну міжмодульну архітектуру програмної системи.

Охарактеризовано внутрішню логіку роботи та програмний код трьох модулів системи: модуля роботи із зображеннями, модуля машинного навчання та модуля взаємодії й діагностики. У межах аналізу ML-компонентів розкрито структуру двовимірної мережі на базі модифікованої архітектури ResNet-18 та власної тривимірної мережі KneeResidualAttentionNet.

Окрему увагу приділено недолікам розроблених ML-моделей та виділено дві основні причини, що негативно впливають на точність їхньої роботи. Ними стали дефіцит наявних медичних даних, що призводить до швидкого перенавчання розроблених моделей, а також вимушене просторове стиснення вхідних даних.

5 ВЗАЄМОДІЯ КОРИСТУВАЧА ІЗ ПРОГРАМНОЮ СИСТЕМОЮ

Оскільки програмний продукт розроблявся як комплексна автоматизована система машинного навчання, щоб не перевантажувати кінцевого користувача надлишковим функціоналом, було прийняте рішення розділити інтерфейс на два варіанти: графічний, що запускається за замовчуванням, та консольний, для запуску якого потрібно прописати команду з відповідним прапорцем.

5.1 Графічний інтерфейс

Після запуску застосунку на екрані з'явиться головне вікно, що складається з кнопки завантаження МРТ-скану, кнопки запуску аналізу, зони перегляду зрізів МРТ-знімка та зони результатів класифікації. Вигляд головного вікна після запуску наведено на рисунку 5.1.

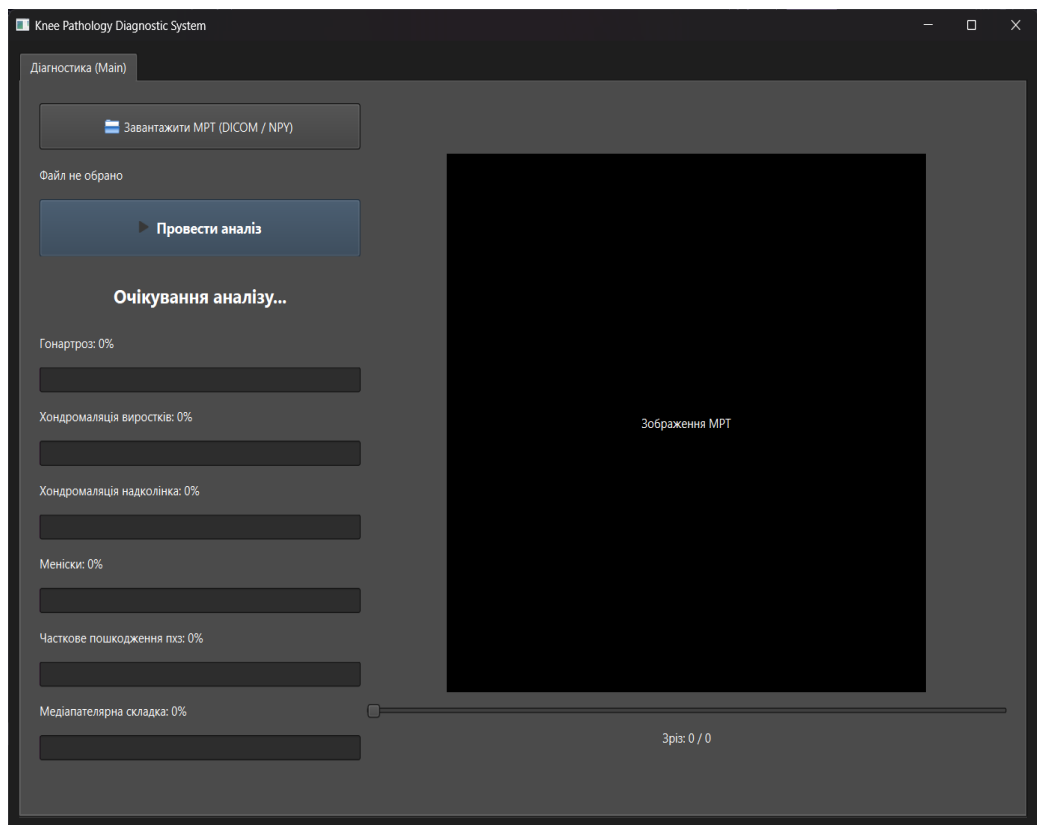


Рисунок 5.1 – Головне вікно GUI після запуску

Після завантаження МРТ-скану у вікні можна буде переглянути його зрізи (рис. 5.2).

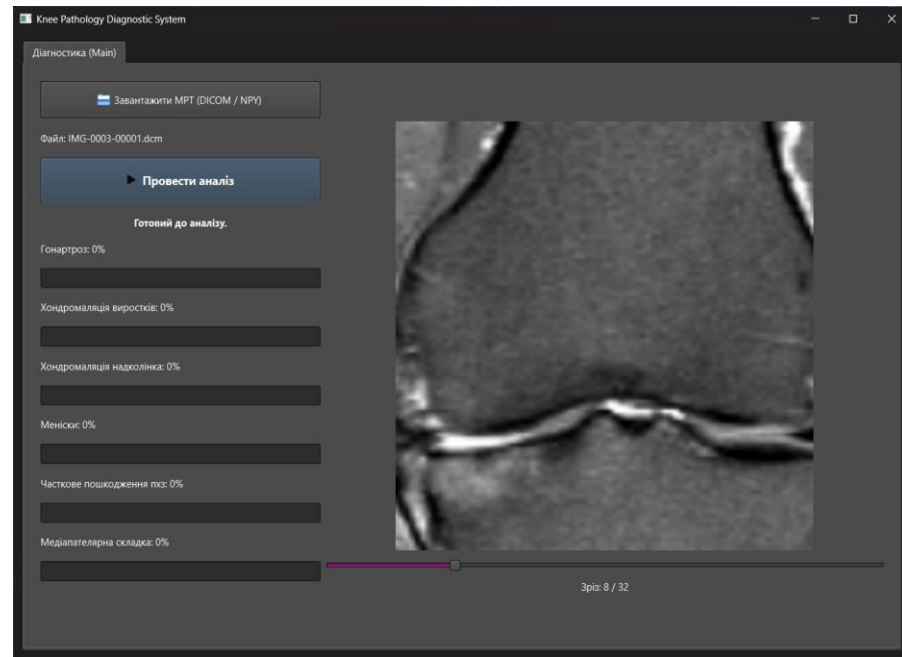


Рисунок 5.2 – Вікно застосунку з попереднім переглядом зрізів МРТ-скану

Після натискання кнопки для проведення аналізу буде проведено бінарну класифікацію для визначення, чи це знімок хворого коліна, чи здорового. Вигляд вікна при визначенні здорового коліна наведено на рисунку 5.3.

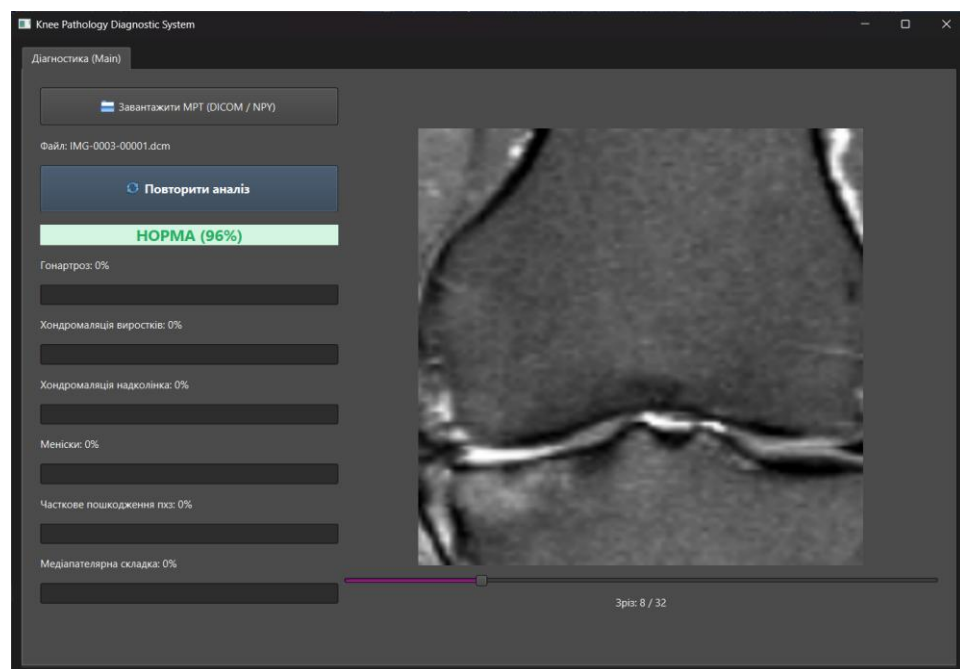


Рисунок 5.3 – Вигляд вікна після визначення здорового коліна

Якщо знімок був класифікований як патологічний, система проведе класифікацію за конкретними хворобами (рисунок 5.4).



Рисунок 5.4 – Вигляд вікна після класифікації конкретної патології

Графічний інтерфейс повністю відповідає створеній діаграмі варіантів використання. Він виконує поставлені базові функції і є достатньо простим для роботи користувача без технічних навичок.

5.2 Інтерфейс командного рядка

Другим варіантом взаємодії з програмним продуктом є консольний інтерфейс. Для доступу до нього необхідно під час запуску системи через консоль ввести прапорець «--cli», після чого буде виведено сам інтерфейс. Вигляд команди з інтерфейсом наведено на рисунку 5.5.

```
o (.venv) PS D:\1-DZ\4_course\Diploma\Code> python main.py --cli

=====
KNEE PATHOLOGY PIPELINE
=====
1 -- Convert DICOM files from datasets
2 -- Examine NumPy files
3 -- Split train data
4 -- Load dataset
5 -- Start NPY model pipeline Stage 1
6 -- Start NPY model pipeline Stage 2
7 -- Start PNG model pipeline Stage 1
8 -- Start PNG model pipeline Stage 2
0 -- Exit
-----
Choice: █
```

Рисунок 5.5 – Вигляд консольного інтерфейсу, разом з командою для запуску

Як і в графічній версії, кожна операція в CLI-режимі супроводжується детальним логуванням. Усі результати діагностики, технічні повідомлення та помилки автоматично записуються в журнал подій через модуль логування.

Висновки до розділу 5

У п'ятому розділі було описано і продемонстровано за допомогою рисунків створені інтерфейси для розробленої системи класифікації травм. Було продемонстровано процес роботи графічного інтерфейсу, з яким повинен працювати лікар-радіолог: від завантаження файлу з відображенням зрізів завантаженого знімка до результатів роботи як бінарного, так і диференціального аналізів. Було наведено інструкцію для запуску CLI-інтерфейсу для розробника і показано зовнішній вигляд списку команд, що виводиться відразу після запуску та є доступними для виконання.

ВИСНОВКИ

У даній дипломній роботі було успішно розроблено автоматизовану систему класифікації травм на основі машинного навчання. Створена система працює з МРТ-знімками колінних суглобів: вона виявляє і класифікує травми на знімках та має можливість модифікації для класифікації інших об'єктів або частин тіла.

Проведено аналіз доступних систем для класифікації медичних зображень на ринку. Було виявлено, що комерційні рішення розповсюджуються за платними підписками, ціни на які публічно не розголошуються через політику компаній-дистриб'юторів. Альтернативи з вільним доступом, на противагу комерційним системам, не виявлено.

Спроектовано дві моделі машинного навчання для роботи з різними типами даних: PNG-зрізами МРТ-знімків та тривимірними масивами.

Проведений аналіз виявив потенціал для подальшого покращення розроблених моделей. Основними напрямками покращення є розширення навчального набору даних для усунення дисбалансу класів, впровадження методів багатоміткової класифікації для одночасного виявлення кількох супутніх травм у пацієнта, а також залучення потужнішого обчислювального обладнання, що дозволить відмовитися від сильного стиснення вхідних даних і зберегти дрібні анатомічні ознаки.

Розроблено два види інтерфейсів для системи: графічний інтерфейс – для класифікації МРТ-знімків у реальному часі, та консольний інтерфейс – для тренування нових моделей машинного навчання на місці розгортання системи.

Навіть за умов дефіциту даних розроблені нейромережі можуть успішно використовуватися як інструменти для первинної діагностики. Впровадження подібних асистентів у клінічну практику дозволяє створити систему «другої думки» для верифікації складних випадків і зменшити робоче навантаження на працівників медичної сфери.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ramon W., Ejaz U., Olaoye G. Big Data in Healthcare: Revolutionizing Diagnostics with AI and Machine Learning. 2025. URL: https://www.researchgate.net/publication/388036100_Big_Data_in_Healthcare_Revolutionizing_Diagnostics_with_AI_and_Machine_Learning (date of access: 07.06.2026)
2. Sarvamangala D. R., Kulkarni R. V. Convolutional neural networks in medical image understanding: a survey. *Evol Intell.* 2022. Vol. 15, no. 1. P. 1–22. DOI: <https://doi.org/10.1007/s12065-020-00540-3>. (date of access: 07.06.2026)
3. Захист інформації та робота з персональними даними у закладі охорони здоров'я. Міністерство здоров'я України. URL: <https://moz.gov.ua/uk/osnovi-kiberbezpeki> (дата звернення: 25.05.2026).
4. Caffery L. Metadata and DICOM for medical photography. *Photography in clinical medicine* : Дослідницька книга. Хам, 2020. P. 545–556.
5. MedGemma. Google DeepMind. URL: <https://deepmind.google/models/gemma/medgemma/> (date of access: 26.05.2026).
6. AI-Rad Companion. Siemens Healthineers | Corporate Home. URL: <https://www.siemens-healthineers.com/digital-health-solutions/ai-rad-companion> (date of access: 26.05.2026).
7. Merge. Medical imaging solutions. Merative. URL: <https://www.merative.com/merge-imaging> (date of access: 26.05.2026).
8. Pydicom. PyPI. URL: <https://pypi.org/project/pydicom/> (date of access: 26.05.2026).
9. Oliphant T. E. Guide to NumPy. 2nd ed. CreateSpace Independent Publishing Platform, 2015. 364 p.
10. Willman J. Modern PyQt : Навч. посіб. Hampton : Apress, 2021. 322 p.
11. Logging – logging facility for python. Python documentation. URL: <https://docs.python.org/3.12/library/logging.html> (date of access: 26.05.2026).

12. Stevens E., Antiga L., Viehmann T. Deep Learning with PyTorch. Shelter Island : Manning Publications Co., 2020. 522p. URL: https://isip.piconepress.com/courses/temple/ece_4822/resources/books/Deep-Learning-with-PyTorch.pdf (date of access: 27.05.2026).
13. Scikit-learn: machine learning in Python – scikit-learn 1.8.0 documentation. scikit-learn. URL: <https://scikit-learn.org/stable/index.html> (date of access: 27.05.2026).
14. Yan J. Application of CNN in computer vision. Applied and Computational Engineering. 2024. Vol. 30, no. 1. P. 104–110. URL: https://www.researchgate.net/publication/377826786_Application_of_CNN_in_computer_vision (date of access: 27.05.2026).
15. Deep Learning-Based Networks for Detecting Anomalies in Chest X-Rays / M. B. Alazzam et al. BioMed Research International. 2022. Vol. 2022, no. 1. P. 1–10. URL: https://www.researchgate.net/publication/362220253_Deep_Learning-Based_Networks_for_Detecting_Anomalies_in_Chest_X-Rays (date of access: 27.05.2026).
16. Ben Baoues E., Jegham I., El Yacoubi M., Ben Khalifa A. Soft-Attention Based Person Re-Identification in Real-world Settings using Variational AutoEncoders. 2024. DOI: <https://doi.org/10.13140/RG.2.2.16154.04807>.

ДОДАТОК А Лістинг окремих файлів розробленої системи

image_converter.py:

```
import os
import struct
import pydicom
from pydicom.pixels.processing import apply_rescale
import numpy as np
import torch
import torch.nn.functional as F
from PIL import Image
from pathlib import Path
from concurrent.futures import ProcessPoolExecutor
from .image_preprocessing import wavelet_denoising_3d, resample_3d, get_knee_bbox, resize_3d_tensor
from ..logger_module.logger import CustomLogger

# Файл, що реалізує головний клас модуля обробки зображень. Відповідає за
# конвертацію DICOM-файлів у PNG та NPY, анонімізацію даних пацієнтів, попередню обробку та розподіл
# даних на набори для навчання.

logger = CustomLogger("Imaging_log")

class DICOMProcessor:
    """
    Main engine for medical image conversion and standardization.

    Responsible for transforming raw DICOM archives into standardized <br>
    Hounsfield Unit (HU) volumes compatible with neural network inputs.
    """
    def __init__(self):
        self.ds = None
        self._pixels_hu = None

    def load_file(self, dicom_path):
        """
        Reads a DICOM file and validates the presence of pixel data. <br>
        Returns True if the file is a valid image ready for processing.
        """
        if not os.path.exists(dicom_path):
            logger.error(f"Path does not exist: {dicom_path}")
            return False

        try:
            self.ds = pydicom.dcmread(dicom_path, force=True)

            pixel_tags = ['PixelData', 'FloatPixelData', 'DoubleFloatPixelData']
            if not any(tag in self.ds for tag in pixel_tags):
                logger.warning(f"Skipping non-image DICOM (Report/SR): {os.path.basename(dicom_path)}")
                return False

            _ = self.ds.pixel_array

            self._pixels_hu = None
            return True
        except AttributeError:
            logger.warning(f"File has no pixel attribute: {os.path.basename(dicom_path)}")
            return False
        except (OSError, struct.error) as e:
            # Handled "No tag to read" and "unpack requires a buffer"
```

```

        logger.warning(f"Corrupted structure in {os.path.basename(dicom_path)}: {e}")
        return False
    except Exception as e:
        logger.error(f"Unexpected error in {os.path.basename(dicom_path)}: {e}")
        return False

@property
def pixels_hu(self):
    """
    Lazy property that rescales raw pixel intensities into Hounsfield Units (HU)
    using the Rescale Slope and Intercept from DICOM metadata.
    """
    if self.ds is None:
        raise ValueError("No DICOM file loaded. Call load_file() first.")

    if self._pixels_hu is None:
        try:
            raw_pixels = self.ds.pixel_array.astype(np.float32)
            self._pixels_hu = apply_rescale(raw_pixels, self.ds)
        except AttributeError as e:
            logger.warning(f"Rescale tags missing, using raw pixel data: {e}")
            self._pixels_hu = self.ds.pixel_array.astype(np.float32)
    return self._pixels_hu

@property
def spacing(self):
    """
    Extracts physical voxel dimensions (Z, H, W) from DICOM headers,
    supporting both standard and functional group sequences.
    """
    if self.ds is None:
        return (1.0, 1.0, 1.0)

    z_spacing = float(getattr(self.ds, 'SpacingBetweenSlices', 1.0))

    pixel_spacing = None

    try:
        if hasattr(self.ds, 'SharedFunctionalGroupsSequence'):
            shared_groups = self.ds.SharedFunctionalGroupsSequence[0]
            if hasattr(shared_groups, 'PixelMeasuresSequence'):
                pixel_spacing = shared_groups.PixelMeasuresSequence[0].PixelSpacing

        if pixel_spacing is None and hasattr(self.ds, 'PerFrameFunctionalGroupsSequence'):
            frame_groups = self.ds.PerFrameFunctionalGroupsSequence[0]
            if hasattr(frame_groups, 'PixelMeasuresSequence'):
                pixel_spacing = frame_groups.PixelMeasuresSequence[0].PixelSpacing
    except Exception as e:
        logger.warning(f"Error traversing functional groups: {e}")

    if pixel_spacing:
        h_spacing = float(pixel_spacing[0])
        w_spacing = float(pixel_spacing[1])
    else:
        logger.warning("PixelSpacing NOT found in functional groups. Using default 1.0mm")
        h_spacing, w_spacing = 1.0, 1.0

    return (z_spacing, h_spacing, w_spacing)

```

```

def get_normalized(self, data, target_range=(0, 1), clamping_percentile=(1, 99)):
    """
    Clamps outliers using percentiles and scales the intensity values
    to a specific range (e.g., 0.0-1.0 or 0-255).
    """
    data = data.astype(np.float32)

    lower_bound = np.percentile(data, clamping_percentile[0])
    upper_bound = np.percentile(data, clamping_percentile[1])

    data = np.clip(data, lower_bound, upper_bound)
    img_min, img_max = data.min(), data.max()

    if img_max - img_min == 0:
        return np.zeros_like(data)

    normalized = (data - img_min) / (img_max - img_min)

    if target_range == (0, 255):
        return (normalized * 255).astype(np.uint8)

    return normalized.astype(np.float16)

def get_processed_volume(
    self, target_range: tuple[int, int] = (0, 1),
    target_spacing: tuple[float, float, float] = (1.0, 1.0, 1.0),
    target_shape: tuple[int, int, int] = (32, 128, 128)):
    """
    Executes the full clinical pipeline: Wavelet denoising, Voxel resampling, <br>
    Knee-specific cropping (BBox), and 3D interpolation to the target shape.
    """
    try:
        data = self.pixels_hu.astype(np.float32)

        data = wavelet_denoising_3d(data)

        logger.info(f"Resampling from {self.spacing} to {target_spacing}...")
        data = resample_3d(data, self.spacing, target_spacing=target_spacing)
        data = get_knee_bbox(data, threshold=0.01)

        tensor = torch.from_numpy(data).float().unsqueeze(0).unsqueeze(0)

        current_depth = tensor.shape[2]
        tensor = F.interpolate(tensor, size=(current_depth, 224, 224), mode='trilinear')

        tensor = resize_3d_tensor(tensor.squeeze(0), target_shape)
        final_data_f32 = tensor.squeeze(0).numpy()

        normalized_data_f16 = self.get_normalized(final_data_f32, target_range=target_range)
        return normalized_data_f16
    except Exception as e:
        logger.error(f"Pipeline processing failed: {e}")
    return None

def save_as_png(self, data, output_path):
    """
    Converts a 3D volume into a series of 2D PNG slices, <br>
    maintaining spatial order for visualization or 2D training.
    """
    data = np.squeeze(data)

```

```

if data.ndim == 3:
    base_name = os.path.splitext(os.path.basename(output_path))[0]
    dicom_dir = os.path.join(os.path.dirname(output_path), base_name)
    os.makedirs(dicom_dir, exist_ok=True)
    for i, slice_data in enumerate(data):
        Image.fromarray(slice_data).save(os.path.join(dicom_dir, f"slice{i:03d}.png"))
    return True
elif data.ndim == 2:
    Image.fromarray(data).save(output_path)
    return True
return False

def batch_conversion(
    self, input_dir, output_png_dir=None, output_npy_dir=None,
    start_idx: int = 1, target_shape: tuple[int, int, int] = (32, 128, 128),
    target_spacing: tuple[float, float, float]=(1.0, 1.0, 1.0)):
    """
    Iterates through a directory to convert all valid DICOM slices
    into unified .npy volumes and .png images.
    """
    files = [f for f in os.listdir(input_dir) if f.endswith('.dcm')]
    current_idx = start_idx

    for filename in files:
        file_path = os.path.join(input_dir, filename)

        if self.load_file(file_path):
            z_spacing, _, _ = self.spacing
            if z_spacing > 10.0:
                logger.warning(f"Skipping survey/scout scan: {filename}")
                continue

            processed_hu = self.get_processed_volume(target_range=(0, 1), target_shape=target_shape,
            target_spacing=target_spacing)
            if processed_hu is None:
                continue

            base_name = f"{current_idx:04d}"

            if output_npy_dir:
                npy_path = os.path.join(output_npy_dir, f"{base_name}.npy")
                np.save(npy_path, processed_hu)

            if output_png_dir:
                png_path = os.path.join(output_png_dir, f"{base_name}.png")
                png_data = (processed_hu * 255).astype(np.uint8)
                self.save_as_png(png_data, png_path)

            current_idx += 1

    return current_idx

@staticmethod
def extract_identity(folder_name: str):
    """
    Parses clinical folder names to extract patient identifiers and
    detected body side (Left/Right) based on naming conventions.
    """
    parts = folder_name.upper().split('T')
    raw_name = parts[0].strip()

```

```

side = ""
if folder_name.upper().endswith('L'):
    side = "_L"
elif folder_name.upper().endswith('R'):
    side = "_R"

return raw_name, side

@staticmethod
def process_single_patient(patient_path, dataset_name, condition_name, patient_label, output_base_png,
output_base_npy, target_shape, target_spacing):
    """
    Independent worker function designed for parallel processing <br>
    of an individual patient's full imaging record.
    """
    processor = DICOMProcessor()
    sub_path = os.path.join(dataset_name, condition_name or "", patient_label)

    target_png = Path(output_base_png) / sub_path
    target_npy = Path(output_base_npy) / sub_path
    target_png.mkdir(parents=True, exist_ok=True)
    target_npy.mkdir(parents=True, exist_ok=True)

    file_counter = 1
    for current_root, _, files in os.walk(patient_path):
        dcm_files = [f for f in files if f.endswith('.dcm')]
        if dcm_files:
            file_counter = processor.batch_conversion(
                current_root,
                output_png_dir=target_png,
                output_npy_dir=target_npy,
                start_idx=file_counter,
                target_shape=target_shape,
                target_spacing=target_spacing
            )
    return file_counter - 1

def process_all_conditions(
    self, conditions_root_dir: str | Path, knee_root_dir: str | Path, output_base_png: str | Path,
    output_base_npy: str | Path, target_shape: tuple[int, int, int] = (32, 128, 128),
    target_spacing: tuple[float, float, float] = (1.0, 1.0, 1.0)
):
    """
    High-level orchestrator that uses multiprocessing (ProcessPoolExecutor) <br>
    to convert an entire multi-class dataset into an anonymized, structured format.
    """
    patient_name_to_id = {}
    next_id = 1

    all_folders_to_process = []

    def collect_folders(root, dataset_tag, condition=None):
        nonlocal next_id
        if not Path(root).exists(): return

        for p_path in sorted(Path(root).iterdir()):
            if p_path.is_dir():
                name_key, side = self.extract_identity(p_path.name)

```

```

if name_key not in patient_name_to_id:
    patient_name_to_id[name_key] = next_id
    next_id += 1

assigned_id = patient_name_to_id[name_key]
all_folders_to_process.append({
    "path": p_path,
    "dataset": dataset_tag,
    "condition": condition,
    "id": assigned_id,
    "side": side
})

c_root = Path(conditions_root_dir)
if c_root.exists():
    for cond_path in [d for d in c_root.iterdir() if d.is_dir()]:
        collect_folders(cond_path, "conditions_dataset", cond_path.name)

collect_folders(knee_root_dir, "knee_dataset", "healthy")

with ProcessPoolExecutor() as executor:
    all_futures = []
    for item in all_folders_to_process:
        patient_label = f"patient#{item['id']} {item['side']}"

        all_futures.append(
            executor.submit(
                self.process_single_patient,
                item["path"], item["dataset"],
                item["condition"], patient_label,
                output_base_png, output_base_npy,
                target_shape, target_spacing
            )
        )

total_files = sum(f.result() for f in all_futures)
logger.info(f"Обробка завершена. Анонімізовано {next_id-1} пацієнтів. Файлів: {total_files}")

```

ml_npy_model.py

```

from pathlib import Path
from typing import Any, cast
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torch.nn.modules.loss import _Loss
from torch.optim import Optimizer
from torch.optim.lr_scheduler import LRScheduler
from torch.amp.autocast_mode import autocast
from torch.amp.grad_scaler import GradScaler
import torch.backends.cudnn as cudnn
from ..logger_module.logger import CustomLogger
from .data_loader import load_dataset
from sklearn.metrics import classification_report

```

Файл коду, що відповідає за навчання моделі, яка працює з тривимірними масивами.
У ньому присутні функції для тренування створеної моделі, її оцінки та збереження результатів.

```

logger = CustomLogger("ML_NPY_model_log")
class DetailBlock3D(nn.Module):
    """
    Implementation of a basic 3D Residual Block.

    - Includes 3D convolution, batch normalization, and LeakyReLU activation.
    - Implements a 'shortcut connection' mechanism to prevent gradient fading.
    - If the input and output dimensions do not match, a 1x1x1 projection convolution is used.
    """
    def __init__(self, in_c, out_c, stride=1):
        super().__init__()
        self.conv = nn.Conv3d(in_c, out_c, kernel_size=3, padding=1, stride=stride, bias=False)
        self.bn = nn.BatchNorm3d(out_c)
        self.relu = nn.LeakyReLU(0.1, inplace=True)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_c != out_c:
            self.shortcut = nn.Sequential(
                nn.Conv3d(in_c, out_c, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm3d(out_c)
            )

    def forward(self, x):
        return self.relu(self.bn(self.conv(x)) + self.shortcut(x))

class KneeResidualAttentionNet(nn.Module):
    """
    Custom neural network architecture for analyzing 3D MRI volumes of the knee joint.

    - Uses DetailBlock3D for feature extraction.
    - Implements a depth-slice Attention mechanism that allows the model to adaptively weight the importance of each MRI slice.
    - Includes a classification head with BatchNorm and Dropout to stabilize training.
    """
    def __init__(self, num_classes: int, dropout_p: float = 0.5):
        super().__init__()

        self.features = nn.Sequential(
            DetailBlock3D(1, 16, stride=1),
            DetailBlock3D(16, 32, stride=2),
            DetailBlock3D(32, 64, stride=2),
            DetailBlock3D(64, 256, stride=2),
        )

        self.attn_layer = nn.Sequential(
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

        self.classifier = nn.Sequential(
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.LeakyReLU(0.1),
            nn.Dropout(p=dropout_p),
            nn.Linear(512, num_classes)
        )

    def forward(self, x: torch.Tensor):

```

```

x = x.float()
x = self.features(x)

# Attention across depth slices
# [B, 256, D, H, W] -> [B, 256, D]
x_pool = F.adaptive_avg_pool3d(x, (x.shape[2], 1, 1)).flatten(2)
x_pool = x_pool.transpose(1, 2) # [B, D, 256]

weights = self.attn_layer(x_pool) # [B, D, 1]
weights = F.softmax(weights, dim=1)

combined_features = torch.sum(x_pool * weights, dim=1) # [B, 256]
return self.classifier(combined_features)

class FocalLoss(nn.Module):
    """
    Implementation of Focal Loss to solve the problem of class imbalance in medical data.

    Formula:
    $
    \text{FL}(p_t) = -(1 - p_t)^\gamma * \log(p_t)
    $

    Where:
    - gamma ( $\gamma$ ): focusing parameter that reduces the contribution of easy-to-classify examples.
    - label_smoothing: regularization technique to prevent overfitting.
    """
    def __init__(self, weight=None, gamma=2.0, label_smoothing=0.0):
        super().__init__()
        self.weight = weight
        self.gamma = gamma
        self.label_smoothing = label_smoothing

    def forward(self, inputs, targets):
        ce_loss = F.cross_entropy(inputs, targets, weight=self.weight, reduction='none',
label_smoothing=self.label_smoothing)
        pt = torch.exp(-ce_loss)
        focal_loss = ((1 - pt) ** self.gamma * ce_loss)
        return focal_loss.mean()

class EarlyStopping:
    """
    Early stopping monitor to prevent overfitting during the training process.

    Criteria for stopping:
    1. Validation loss does not improve for a specified number of epochs (patience).
    2. The accuracy gap between training and validation sets exceeds a defined threshold (max_gap).
    """
    def __init__(self, patience: int = 12, min_delta: float = 0.005, max_gap: float = 20.0):
        self.patience, self.min_delta, self.max_gap = patience, min_delta, max_gap
        self.counter, self.best_loss, self.early_stop = 0, None, False

    def __call__(self, val_loss, train_acc, val_acc):
        gap = train_acc - val_acc
        if gap > self.max_gap:
            logger.warning(f"!!! Overfitting: Acc gap {gap:.2f}%")
            self.early_stop = True
            return
        if self.best_loss is None: self.best_loss = val_loss
        elif val_loss > self.best_loss - self.min_delta:

```

```

        self.counter += 1
        logger.info(f"EarlyStopping counter: {self.counter}/{self.patience}")
        if self.counter >= self.patience: self.early_stop = True
    else:
        self.best_loss, self.counter = val_loss, 0

def train_model(
    model: Any, train_loader: Any, val_loader: Any, criterion: Any,
    optimizer: Any, scheduler: Any, device: Any, epochs: int, stage_config: Any
):
    """
    Main training loop of a model with support for Mixed Precision.

    - Trains on the training set and validates after each epoch.
    - Uses GradScaler to optimize memory on the GPU and EarlyStopping for quality control.
    - Returns the model with the best weights for validation accuracy.
    """
    scaler = GradScaler("cuda")
    best_val_acc = 0.0
    best_model_state = None
    early_stopping = EarlyStopping(patience=stage_config['patience'], max_gap=stage_config['max_gap'])

    logger.info(f"Start training 3D Stage {stage_config['stage_num']} on {device}")

    for epoch in range(epochs):
        model.train()
        train_loss, train_correct, train_total = 0.0, 0, 0
        for images, labels in train_loader:
            images: torch.Tensor
            labels: torch.Tensor

            images, labels = images.to(device, non_blocking=True), labels.to(device, non_blocking=True)
            optimizer.zero_grad(set_to_none=True)

            with autocast(device_type="cuda"):
                outputs = model(images)
                loss = criterion(outputs, labels)

            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()

            train_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            train_total += labels.size(0)
            train_correct += (predicted == labels).sum().item()

        model.eval()
        val_loss, val_correct, val_total = 0.0, 0, 0
        with torch.no_grad():
            with autocast(device_type="cuda"):
                for images, labels in val_loader:
                    images, labels = images.to(device), labels.to(device)
                    outputs = model(images)
                    loss = criterion(outputs, labels)
                    val_loss += loss.item()
                    _, predicted = torch.max(outputs.data, 1)
                    val_total += labels.size(0)
                    val_correct += (predicted == labels).sum().item()

```

```

train_acc, val_acc = 100 * train_correct / train_total, 100 * val_correct / val_total
avg_val_loss = val_loss / len(val_loader)
scheduler.step()

logger.info(f"Epoch [{epoch+1}/{epochs}] | Loss: {avg_val_loss:.4f} Acc: {val_acc:.2f}% | Train: {train_acc:.2f}%")

if val_acc > best_val_acc:
    best_val_acc = val_acc
    best_model_state = model.state_dict().copy()
    logger.info(f"New best model: {val_acc:.2f}%")

early_stopping(avg_val_loss, train_acc, val_acc)
if early_stopping.early_stop:
    break

if best_model_state:
    model.load_state_dict(best_model_state)
return model

def evaluate_model(
    model: KneeResidualAttentionNet,
    test_loader: DataLoader,
    device: torch.device,
    class_names: list[str]
):
    """
    Performs a comprehensive evaluation of the model on the held-out test set.

    Generates a classification report including Precision, Recall, and F1-score <br>
    for each pathology class, providing insights into model performance per condition.
    """
    logger.info("Starting model evaluation")
    model.eval()
    all_predictions, all_labels = [], []

    with torch.no_grad():
        with autocast(device_type="cuda"):
            for images, labels in test_loader:
                images: torch.Tensor
                labels: torch.Tensor

                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                _, predicted = torch.max(outputs, 1)

                all_predictions.extend(predicted.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

    report = classification_report(
        all_labels,
        all_predictions,
        target_names=class_names,
        zero_division=0
    )
    logger.info(f"\n{report}")

    accuracy: float = (np.array(all_predictions) == np.array(all_labels)).mean() * 100
    logger.info(f"Overall Test Accuracy: {accuracy:.2f}%")

```

```

logger.info("Evaluation complete.")

def start_npy_model_pipeline(
    base_data_path: str | Path = "data/prepared_data",
    epochs: int = 40, batch_size: int = 4,
    mode: str = 'npz', save_file_name: str = "knee_3d_binary_model",
    cache_in_ram: bool = False
):
    """
    Complete first stage training pipeline (Stage 1: Binary Classification).

    - Task: Determining the presence of pathology (Normal vs Pathology).
    - Process: Loading data (.npz), initializing the architecture, training and saving weights.
    """
    cudnn.benchmark = True
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    train_loader, val_loader, test_loader, classes = load_dataset(base_data_path, batch_size, mode, stage=1,
cache_in_ram=cache_in_ram)

    # Stage 1: Binary (Healthy vs Pathology)
    model = KneeResidualAttentionNet(num_classes=2, dropout_p=0.4).to(device)

    optimizer = optim.AdamW(model.parameters(), lr=5e-5, weight_decay=0.1)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

    # [Healthy, Pathology]
    weights = torch.tensor([1.0, 1.0]).to(device)
    criterion = FocalLoss(weight=weights, gamma=2.0, label_smoothing=0.1)

    config = {'stage_num': 1, 'patience': 15, 'max_gap': 25.0}

    try:
        model = train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, device, epochs, config)
        evaluate_model(model, test_loader, device, classes)
        torch.save(model.state_dict(), f"models/{save_file_name}.pth")
        logger.info(f"3D Binary model saved.")
    except Exception as e:
        logger.error(f"3D Pipeline failed: {e}")

def start_stage2_npy_pipeline(
    base_data_path: str | Path = "data/prepared_data",
    binary_model_path: str | Path = "models/knee_3d_binary_model.pth",
    epochs: int = 60,
    batch_size: int = 4,
    save_file_name: str = "knee_3d_stage2_6classes",
    cache_in_ram: bool = False
):
    """
    Stage 2: Differential Diagnosis training pipeline.

    - Task: Classification of a specific type of pathology among 6 classes.
    - Feature: Implements 'Model Surgery' — loads the weights of the binary model, <br>
replaces the last linear layer with a multi-class one and performs fine-tuning <br>
of only the classifier, freezing the feature extraction blocks. <br>
    """
    cudnn.benchmark = True
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```

```

train_loader, val_loader, test_loader, classes = load_dataset(base_data_path, batch_size, mode="numpy", stage=2,
cache_in_ram=cache_in_ram)

model = KneeResidualAttentionNet(num_classes=2, dropout_p=0.6).to(device)

# Binary weights load
try:
    model.load_state_dict(torch.load(binary_model_path))
    logger.info(f"3D Base weights loaded.")
except Exception as e:
    logger.error(f"Failed to load binary 3D model: {e}"); return

# Surgery on classifier
classifier_seq = cast(nn.Sequential, model.classifier)
in_features = cast(nn.Linear, classifier_seq[4]).in_features
classifier_seq[4] = nn.Linear(in_features, len(classes))
model.to("cuda")

# Optimizer
for param in model.features.parameters(): param.requires_grad = False
for param in model.classifier.parameters(): param.requires_grad = True

optimizer = optim.AdamW(model.classifier.parameters(), lr=1e-4, weight_decay=0.2)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

weights = torch.tensor([1.5, 2.0, 1.0, 1.5, 1.0, 1.8]).to(device)
criterion = FocalLoss(weight=weights, gamma=2.0, label_smoothing=0.1)

config = {'stage_num': 2, 'patience': 20, 'max_gap': 25.0}

try:
    model = train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, device, epochs, config)
    evaluate_model(model, test_loader, device, classes)
    torch.save(model.state_dict(), f"models/{save_file_name}.pth")
    logger.info(f"3D Stage 2 complete.")
except Exception as e:
    logger.error(f"3D Stage 2 failed: {e}")

```



Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

АВТОМАТИЗОВАНА СИСТЕМА КЛАСИФІКАЦІЇ ТРАВМ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Виконав: студент IV курсу, групи
ТВ-23

Лисенко Олександр Сергійович

Керівник: старший викладач
Дацюк Оксана Антонівна

Київ 2026р.



АКТУАЛЬНІСТЬ ЗАДАЧІ

1. Зменшення навантаження на медичних працівників
2. Відсутність на ринку фінансово доступних альтернатив
3. Технологія «другої думки» для лікарів
4. Практична користь навіть з невеликим набором даних для навчання



Designed by Magnific



МЕТА, ОБ'ЄКТ, ПРЕДМЕТ І ЗАВДАННЯ РОБОТИ

Мета: Розробити автоматизовану систему класифікації травм колінних суглобів з використанням машинного навчання

Об'єкт: Процес автоматизованого аналізу та класифікації медичних зображень колінних суглобів

Предмет: Методи та моделі машинного навчання для автоматичного виявлення та класифікації патологій колінного суглоба

Завдання:

1. Проаналізувати вже доступні на ринку системи для медичної класифікації
2. Розробити програмні модулі для підготовки навчальних даних, навчання моделей, графічного інтерфейсу та логування процесу роботи системи
3. Провести оцінку роботи розробленої моделі машинного навчання
4. Розробити графічний інтерфейс для користування системою



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

3/16

АНАЛІЗ ДОСТУПНИХ РІШЕНЬ



Працює зі знімками мозку, грудної клітини, простати.

Може планувати променеву терапію внутрішніх органів.

Необхідне підключення до серверів компанії.

Ціна підписки публічно не відома



Merge

Система централізованого управління медичними даними.

3D рендеринг, сегментація, автоматичне управління даними.

Необхідне підключення до серверів компанії.

Ціна підписки публічно не відома

MedGemma

Відкритий набір медичних моделей для роботи із зображеннями й текстом.

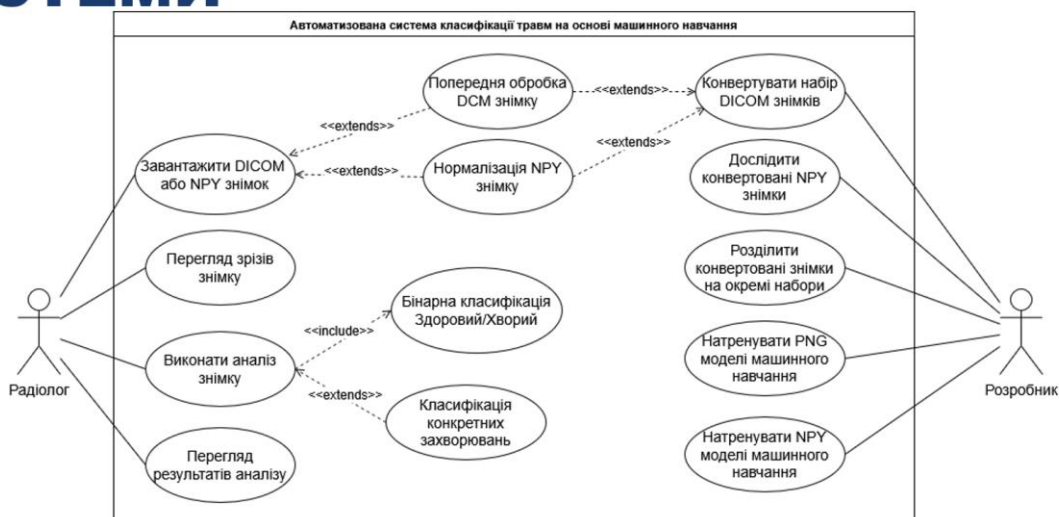
Лише моделі. Для користування необхідно розробити ПЗ.



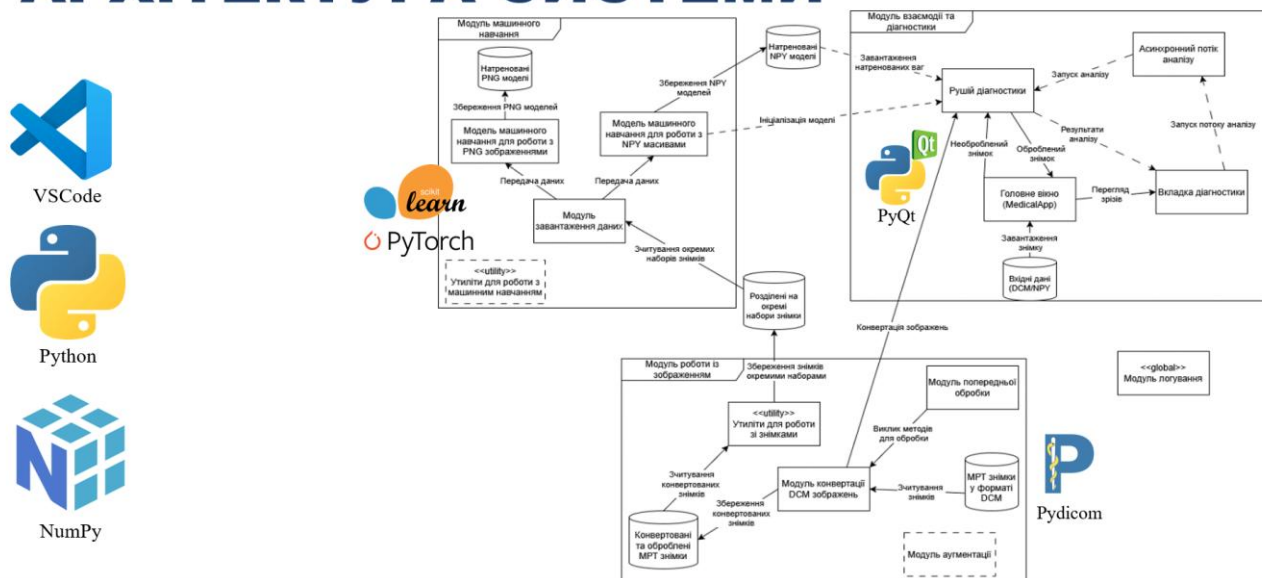
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

4/16

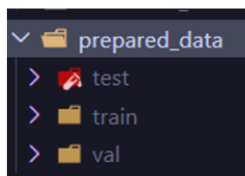
ВАРІАНТИ ВИКОРИСТАННЯ СИСТЕМИ



АРХІТЕКТУРА СИСТЕМИ

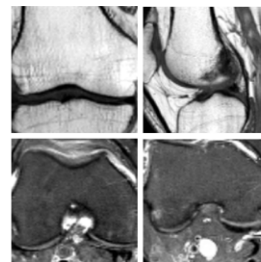


ПІДГОТОВКА ДАНИХ ДЛЯ НАВЧАННЯ



Набори для навчання

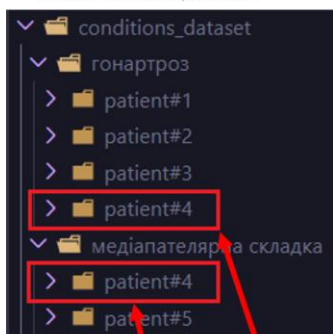
№	Етап
1	Анонімізація шляхом видалення метаданих
2	Пропуск порожніх або пошкоджених знімків
3	Попередня обробка (вейвлет фільтрація, виділення області інтересу, обрізання)
4	Приведення до однакового розміру
5	Конвертація в PNG та 3D NPY масиви
6	Розділення на набори для навчання



Приклади зрізів МРТ-знімків

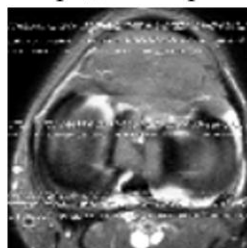
ПРОБЛЕМА ОБРОБКИ ДАНИХ

Дублювання знімків пацієнтів



Декілька хвороб у одного пацієнта

Артефакти після попередньої обробки



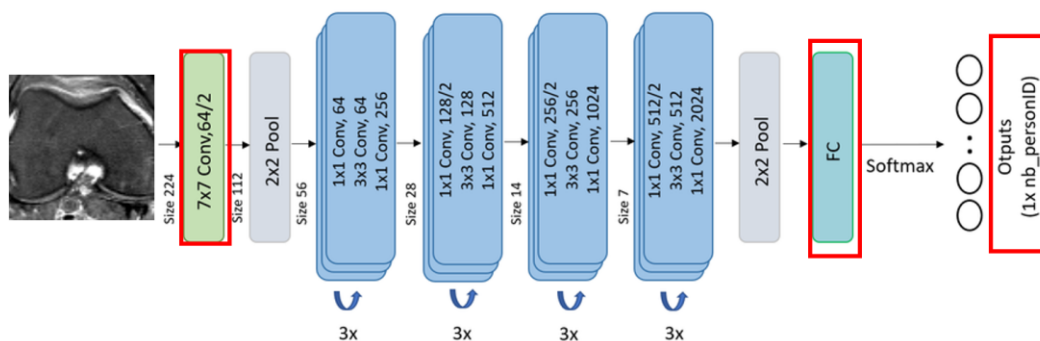
Попередження про порожні знімки

```
WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm
WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm
WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm
WARNING: Skipping survey/scout scan: IMG-0001-00001.dcm
```

Попередження про пошкодженні знімки

```
WARNING: Corrupted structure in IMG-0002-00001.dcm: No tag to read at file position 1004098
WARNING: Skipping non-image DICOM (Report/SR): IMG-0005-00001.dcm
```

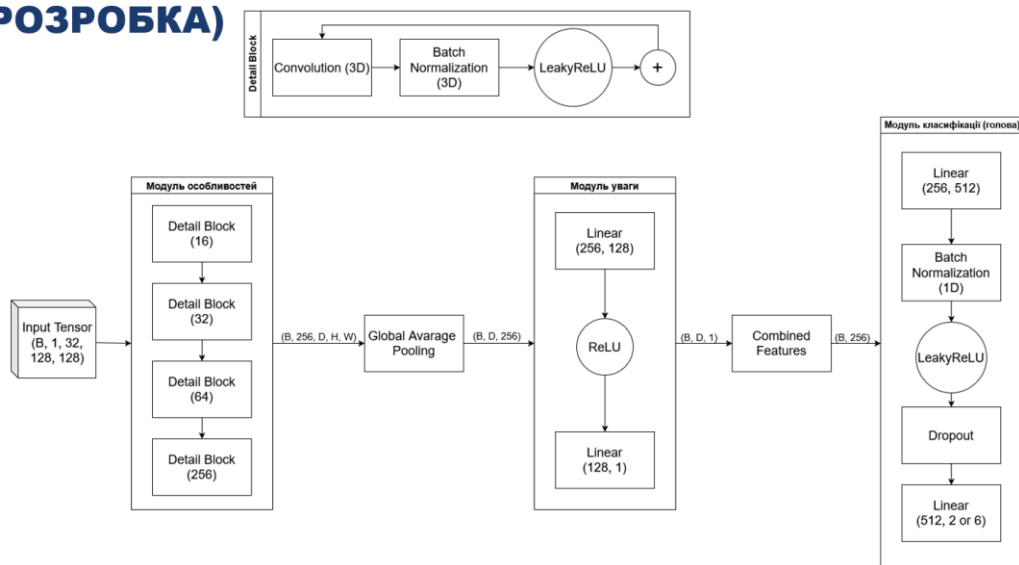
АРХІТЕКТУРА PNG-МЕРЕЖІ (ПОЧАТКОВА РОЗРОБКА)



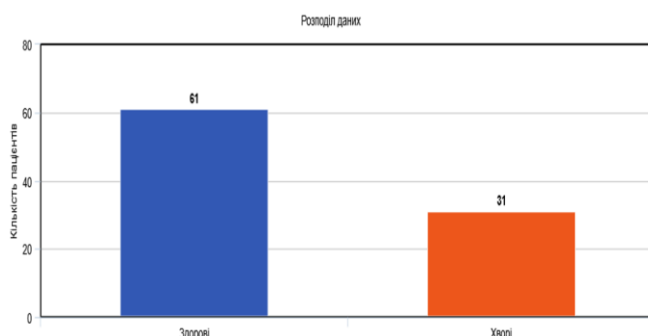
ResNet-18

Ben Baoues E., Jegham I., El Yacoubi M., Ben Khalifa A. Soft-Attention Based Person Re-Identification in Real-world Settings using Variational AutoEncoders. 2024. DOI: <https://doi.org/10.13140/RG.2.2.16154.04807>.

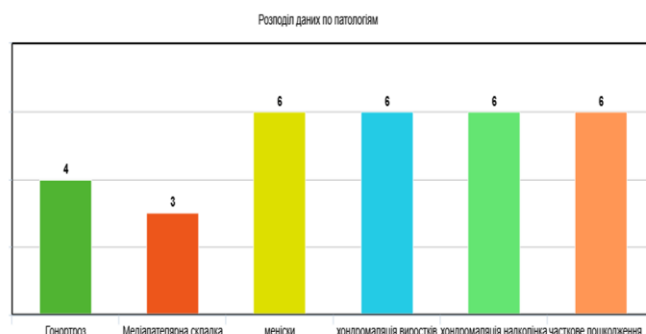
АРХІТЕКТУРА NPY-МЕРЕЖІ (ВЛАСНА РОЗРОБКА)



ПРОБЛЕМА НЕСТАЧІ ДАНИХ



Загальний розподіл



Розподіл патологій



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

11/16

КІНЦЕВА ТОЧНІСТЬ

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Здорові	0.81	0.96	0.88	49
Патологія	0.91	0.65	0.75	31
Загальна точність	—	—	0.84	80
Середнє макро	0.86	0.80	0.82	80
Середнє зважене	0.85	0.84	0.83	80

Класифікація:
здоровий / хворий

Класифікація окремих патологій

Метрика	Точність	Повнота	F1-міра	Кількість
1	2	3	4	5
Гонартроз	0.60	0.60	0.60	5
Хондромалія виросків	0.10	0.20	0.13	5
Хондромалія надколінка	0.20	0.20	0.20	5
Меніски	0.00	0.00	0.00	5
Часткове пошкодження пхз	0.33	0.20	0.25	5
Медіапатеярна складка	0.29	0.33	0.31	6
Загальна точність	—	—	0.26	31
Середнє макро	0.25	0.26	0.25	31
Середнє зважене	0.25	0.26	0.25	31

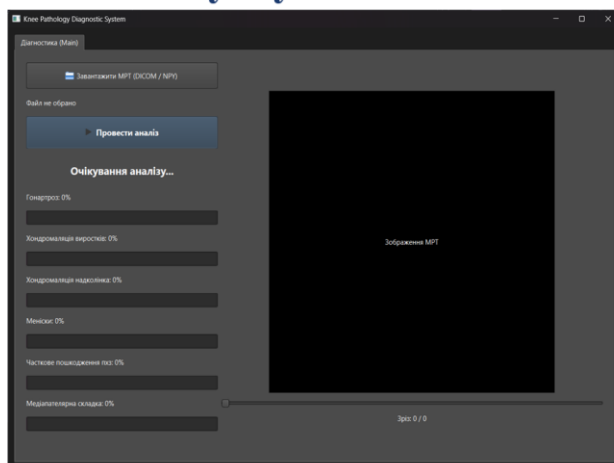


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

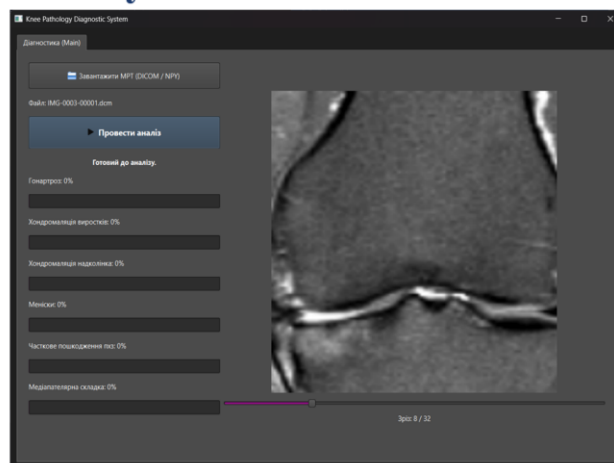
12/16

ГРАФІЧНИЙ ІНТЕРФЕЙС

Після запуску та завантаження знімку



Початок роботи



Після завантаження знімку

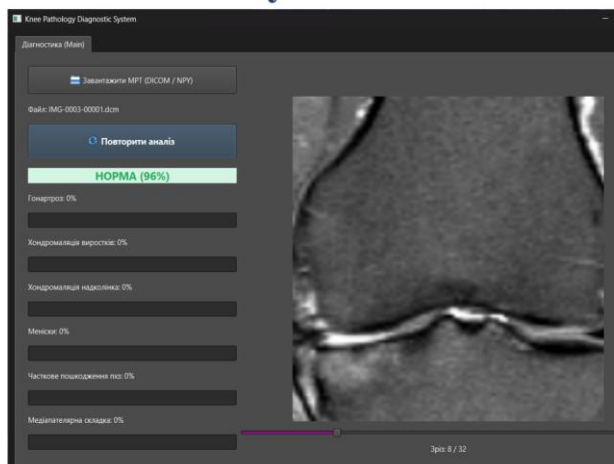


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

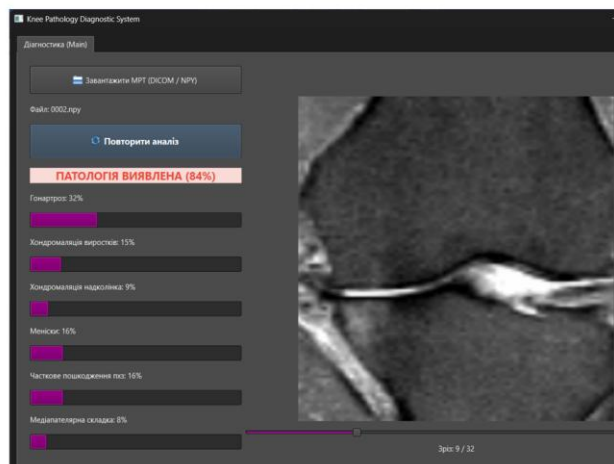
13/16

ГРАФІЧНИЙ ІНТЕРФЕЙС

Після аналізу



Класифікація здорового коліна



Класифікація хворого коліна



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

14/16

ІНТЕРФЕЙС МОДИФІКАЦІЇ СИСТЕМИ

Дозволяє натренувати власні моделі машинного навчання для розробленого графічного інтерфейсу на місці розгортання системи.

Включає в себе:

1. Підготовку даних для навчання
2. Перевірки для навчальних даних
3. Тренування моделей.

```
(.venv) PS D:\1-DZ\4_course\Diploma\Code> python main.py --cli

=====
KNEE PATHOLOGY PIPELINE
=====
1 -- Convert DICOM files from datasets
2 -- Examine NumPy files
3 -- Split train data
4 -- Load dataset
5 -- Start NPY model pipeline Stage 1
6 -- Start NPY model pipeline Stage 2
7 -- Start PNG model pipeline Stage 1
8 -- Start PNG model pipeline Stage 2
0 -- Exit
-----
Choice: █
```

Консольний інтерфейс



ВИСНОВКИ

1. Проведено аналіз вже доступних систем для класифікації медичних зображень на ринку. Виявлено, що комерційні рішення розповсюджуються за платною підпискою, ціна на яку публічно не оголошується, а також вони потребують постійного зв'язку із серверами компаній-дистриб'юторів. Альтернативні системи у вільному доступі – відсутні.
2. Спроектовано систему з модульною архітектурою, яка забезпечує слабку пов'язаність програмних модулів між собою та простоту в масштабуванні.
3. Спроектовано 2 моделі машинного навчання для роботи з різними типами даних: PNG-знімками та 3-вимірними масивами знімків.
4. Розроблено 2 види інтерфейсів для системи:
 - графічний інтерфейс – для класифікації знімків у реальному часі,
 - консольний інтерфейс – для тренування власних моделей машинного навчання на місці розгортання системи.

