Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Освітня програма *Комп'ютерні науки*
Назва освітньої програми

Виконав: *студент 2 курсу, група КНм-23-1* (Курс, група виконавця) _____ (Підпис) *Максим ДІХТЯР* (Ім'я, прізвище)

Керівник: *док.філ., ст. викл. каф. КН* (Науковий ступінь, посада) _____ (Підпис) *Павло РАДЮК* (Ім'я, прізвище)

Нормоконтроль: *к.т.н., доцент кафедри КН* (Науковий ступінь, посада) _____ (Підпис) *Руслан БАГРІЙ* (Ім'я, прізвище)

До захисту допускаю:
Зав. кафедри КН, д.т.н., професор _____ (Підпис) *Олександр БАРМАК* (Ім'я, прізвище)

16 грудня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *інформаційних технологій*
Кафедра *комп'ютерних наук*
Освітній ступінь *магістр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук*

_____
(підпис)
д.т.н., професор Олександр БАРМАК
«*02*» *вересня* *2024* року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

1. Тема кваліфікаційної роботи магістра: «Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ»

2. Завдання видано студенту *Максиму ДІХТЯРУ*
Ім'я, прізвище

3. Керівник роботи *старший викладач кафедри КН Павло РАДЮК*
Ім'я, прізвище

4. Затверджені наказом університету від «*26*» *серпня* *2024* р. № *60*.

5. Дата видачі завдання студенту: «*02*» *вересня* *2024* р.

6. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Метою кваліфікаційної роботи магістра є підвищення рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання. Досягнення мети роботи передбачає виконання таких задач: провести аналіз моделей, методів та технологій глибокого навчання для оброблення та аналізу зображень МРТ; удосконалити метод виявлення візуальних патологій серця за зображенням МРТ з використанням згорткової нейронної мережі; спроєктувати метод інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих від згорткової нейронної мережі; реалізувати метод інтерпретування у вигляді вебзастосунку; провести експериментальне тестування реалізованого модуля за еталонними наборами даних.

Реферат

Кваліфікаційна робота магістра присвячена підвищенню рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання, через удосконалення методу інтерпретування результатів виявлення патологій серця у вигляді вебзастосунку.

Актуальність теми. Актуальність дослідження зумовлена зростанням обсягів медичних МРТ-зображень серця, які створюють виклики для їхнього своєчасного та точного аналізу. Хоча методи глибокого навчання демонструють високу ефективність у виявленні патологій, недостатня прозорість цих моделей обмежує їхнє широке використання в медичній практиці. З огляду на це, актуальним є проєктування методів інтерпретації результатів виявлення патологій серця на МРТ-зображеннях, який би поєднував точність нейронних мереж з можливістю пояснення їхніх рішень.

Об'єкт дослідження. Процес інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.

Предмет дослідження. Моделі, методи та технології глибокого навчання для аналізу зображень МРТ.

Мета і задачі роботи. Мета роботи полягає у підвищенні рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.

Досягнення мети роботи передбачає виконання таких задач:

1. Провести аналіз моделей, методів та технологій глибокого навчання для оброблення та аналізу зображень МРТ.

2. Вдосконалити метод виявлення візуальних патологій серця за зображенням МРТ з використанням згорткової нейронної мережі.

3. Спроєктувати метод інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих від згорткової нейронної мережі.

4. Реалізувати метод інтерпретування у вигляді вебзастосунку.

5. Провести експериментальне тестування реалізованого модуля за еталонними наборами даних.

Методи дослідження. У роботі використано методи аналізу та синтезу для вивчення сучасних архітектур згорткових нейронних мереж, Grad-CAM, SHAP та методів візуалізації медичних даних. Для навчання моделі застосовано методи оброблення зображень, оптимізації функції втрат та аугментації даних.

Наукова новизна одержаних результатів. Удосконалено метод інтерпретування результатів виявлення патологій серця за зображеннями МРТ, який, на відміну від існуючих, поєднує використання архітектури U-Net для точного сегментування анатомічних структур серця з ResNet50 для класифікації патологій, а також застосування методів Grad-CAM та SHAP для генерації візуальних і текстових пояснень до рішень моделі, що забезпечує їхню інтерпретованість, що дає можливість інтерпретації рішень, сприяючи підвищенню довіри медичних фахівців до систем штучного інтелекту.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на науково-практичній конференції – XVI Всеукраїнська науково-практична конференція “Актуальні проблеми комп’ютерних наук (АПКН – 20243)”, м. Хмельницький, ХНУ, 15–16 листопада 2024 р. (Діхтяр М. О., Радюк П. М., Скрипник Т. К. Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ. Актуальні проблеми комп’ютерних наук АПКН-2024 : матеріали XVI Всеукр. науково-практ. конф., м. Хмельницький, 15–16 листоп. 2024 р. Хмельницький, 2024. Хмельницький : ХНУ, 2024. С. 189–191. URL: https://elar.khmnu.edu.ua/handle/123456789/17156).

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 41 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 117 сторінок, з поміж яких 81 сторінка основного тексту та 36 сторінок додатків. У роботі наведено 44 рисунки, 11 формул та 8 таблиць.

Ключові слова: МРТ, виявлення патологій серця, згорткові нейронні мережі, пояснюваний штучний інтелект, Grad-CAM, SHAP.

Зміст

Перелік скорочень

Скорочення, термін, позначення	**Пояснення**
МРТ	Магнітно-резонансна томографія
ШІ	Штучний інтелект
ЗНМ	Згорткова нейронна мережа
XAI	Explainable Artificial Intelligence
Grad-CAM	Gradient-weighted Class Activation Mapping
LIME	Local Interpretable Model-agnostic Explanations
SHAP	SHapley Additive exPlanations
IoU	Intersection over Union
API	Application Programming Interface
ReLU	Rectified Linear Unit
BCE	Binary Cross-Entropy
FLOPS	Floating Point Operations Per Second
VGG	Visual Geometry Group
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
DICOM	Digital Imaging and Communications in Medicine
КТ	Комп’ютерна томографія
LRP	Layer-wise Relevance Propagation

Вступ

Актуальність теми. Актуальність дослідження зумовлена зростанням обсягів медичних зображень серця магнітно-резонансної томографії, які створюють виклики для їхнього своєчасного та точного аналізу. Хоча методи глибокого навчання демонструють високу ефективність у виявленні патологій, недостатня прозорість цих моделей обмежує їхнє широке використання в медичній практиці. З огляду на це, актуальним є проєктування методів інтерпретації результатів виявлення патологій серця на МРТ-зображеннях, який би поєднував точність нейронних мереж з можливістю пояснення їхніх рішень.

Об’єкт дослідження. Процес інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.

Предмет дослідження. Моделі, методи та технології глибокого навчання для аналізу зображень МРТ.

Мета і задачі роботи. Мета роботи полягає у підвищенні рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.

Досягнення мети роботи передбачає виконання таких задач:

1. Провести аналіз моделей, методів та технологій глибокого навчання для оброблення та аналізу зображень МРТ.
2. Вдосконалити метод виявлення візуальних патологій серця за зображенням МРТ з використанням згорткової нейронної мережі.
3. Спроєктувати метод інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих від згорткової нейронної мережі.
4. Реалізувати метод інтерпретування у вигляді вебзастосунку.
5. Провести експериментальне тестування реалізованого модуля за еталонними наборами даних.

Методи дослідження. У роботі використано методи аналізу та синтезу для вивчення сучасних архітектур згорткових нейронних мереж, Grad-CAM, SHAP та

методів візуалізації медичних даних. Для навчання моделі застосовано методи оброблення зображень, оптимізації функції втрат та аугментації даних.

Наукова новизна одержаних результатів. Удосконалено метод інтерпретування результатів виявлення патологій серця за зображеннями МРТ, який, на відміну від існуючих, поєднує використання архітектури U-Net для точного сегментування анатомічних структур серця з ResNet50 для класифікації патологій, а також застосування методів Grad-CAM та SHAP для генерації візуальних і текстових пояснень до рішень моделі, що забезпечує їхню інтерпретованість, що дає можливість інтерпретації рішень, сприяючи підвищенню довіри медичних фахівців до систем штучного інтелекту.

Апробація результатів кваліфікаційної роботи магістра та публікації. Основні наукові та практичні результати пройшли апробацію на науково-практичній конференції – XVI Всеукраїнська науково-практична конференція “Актуальні проблеми комп’ютерних наук (АПКН – 20243)”, м. Хмельницький, ХНУ, 15–16 листопада 2024 р. [1].

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із завдання, реферату, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань з 41 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 117 сторінок, з поміж яких 81 сторінка основного тексту та 36 сторінок додатків. У роботі наведено 44 рисунки, 11 формул та 8 таблиць.

РОЗДІЛ 1. Дослідження методів і технологій штучного інтелекту для аналізу зображень магнітно-резонансної томографії

1.1 Аналіз проблеми оброблення та пояснення результатів класифікації МРТ-зображень

Наразі МРТ є одним із найбільш важливих інструментів сучасної медицини для отримання детальних зображень внутрішніх органів і тканин без використання іонізуючого випромінювання. Цей метод широко застосовують для діагностування різних захворювань, зокрема неврологічних, кардіологічних та онкологічних патологій. Водночас аналіз та інтерпретація МРТ-зображень можуть виявлятися складними й трудомісткими, особливо за наявності великих обсягів даних [1].

Розвиток технологій штучного інтелекту (ШІ) та глибинного навчання відкриває нові можливості для автоматизації оброблення та аналізу МРТ-зображень. Зокрема, згорткові нейронні мережі (ЗНМ) демонструють високі показники у задачах класифікації та сегментації зображень, оскільки здатні автоматично виокремлювати складні просторові та контекстуальні ознаки, що робить їх корисними у медичній діагностиці [2].

Незважаючи на високу точність класифікації, яку забезпечують сучасні моделі глибинного навчання, вони нерідко постають як так звані «чорні скриньки». Це означає, що внутрішні механізми ухвалення рішень залишаються непрозорими для кінцевого користувача. У контексті медичної діагностування така непрозорість породжує низку проблем:

– недовіра медичних фахівців: лікарі можуть не довіряти системам ШІ, якщо не розуміють обґрунтування конкретних результатів;

– юридична відповідальність: відсутність пояснень ускладнює визначення відповідальної сторони у разі хибних діагностичних висновків;

– етичні міркування: пацієнти мають право знати, як було сформульовано діагноз, пов’язаний з їхнім здоров’ям [3].

Для вирішення окреслених проблем розвивається напрямок пояснюваного штучного інтелекту (рисунок 1.1). Його метою є проєктування методів та моделей, що забезпечують високу точність та надають зрозумілі пояснення своїх рішень.

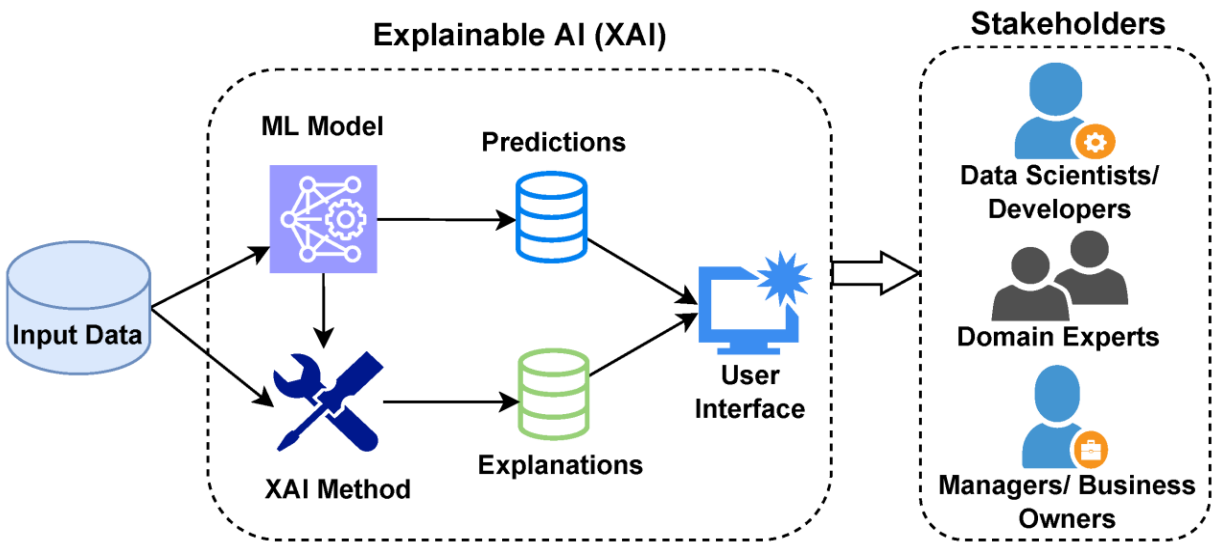


Рисунок 1.1 – Процес пояснюваного штучного інтелекту [4]

Пояснення результатів класифікації МРТ-зображень має низку особливостей та викликів:

– висока розмірність даних. МРТ-зображення можуть мати три- або чотиривимірну структуру (враховуючи часовий компонент), що ускладнює їхній аналіз та візуалізацію;

– складність патологій. Деякі патологічні зміни можуть бути мінімальними за розміром або маскуватися під нормальні тканини, що вимагає високої точності та чутливості моделей;

– інтерпретація для медичних фахівців. Пояснення слід подавати у формі, зрозумілій лікарям, які можуть не мати глибоких знань у галузі ШІ [5].

Існує кілька методів, розроблених для пояснення рішень моделей глибокого навчання в контексті аналізу зображень:

Grad-CAM (Gradient-weighted Class Activation Mapping) дає змогу візуалізувати, які області зображення найбільше вплинули на рішення моделі. Цей

метод обчислює градієнти вихідного класу відносно активацій певного шару, створюючи теплову карту важливості пікселів [6].

Приклад застосування Grad-CAM до МРТ-зображень головного мозку наведено на рисунку 1.2 [7]. Теплова карта накладається на оригінальне зображення, позначаючи області, що вплинули на класифікацію.

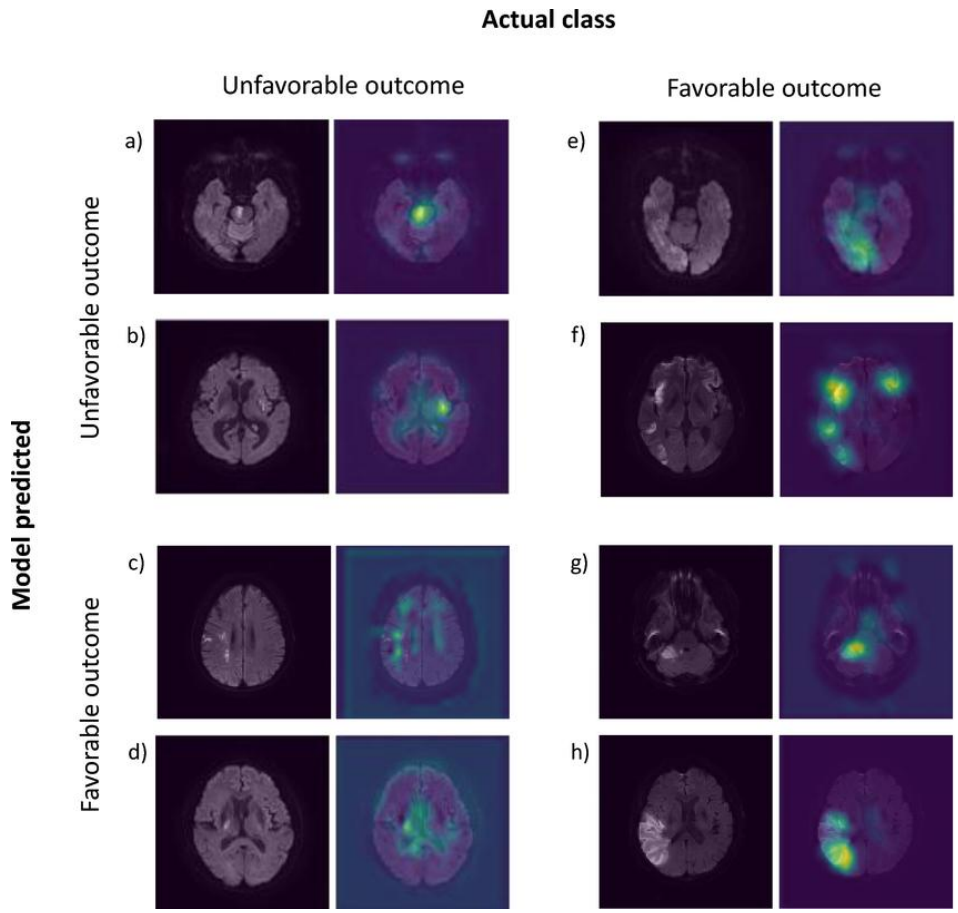


Рисунок 1.2 – Приклад застосування Grad-CAM [7]

LIME (Local Interpretable Model-agnostic Explanations) створює просту лінійну модель, яка апроксимує поведінку складної моделі в локальній околиці простору ознак як зображено на рисунку 1.3. Це дає можливість отримати інтерпретовані пояснення для окремих прикладів.

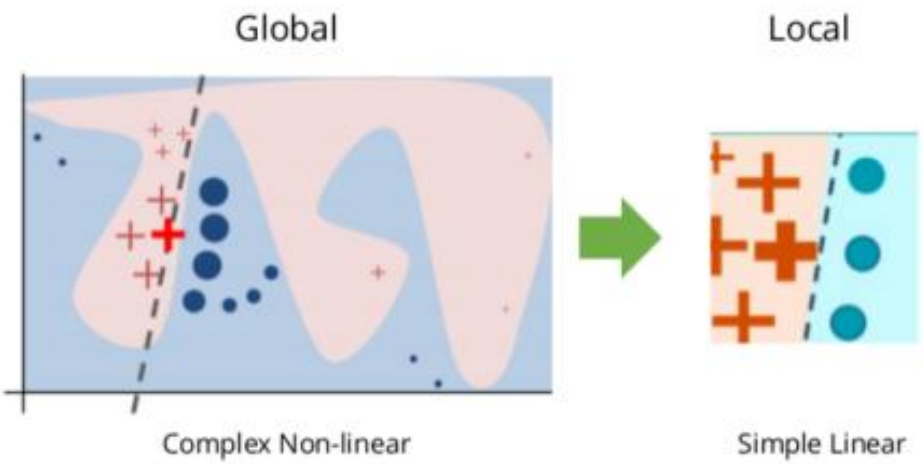


Рисунок 1.3 – Приклад застосування LIME [8]

SHAP (SHapley Additive exPlanations), оснований на теорії ігор, оцінює внесок кожної ознаки (або групи ознак) у прогноз моделі (рисунок 1.4). Цей метод надає змогу отримувати як глобальні, так і локальні пояснення.

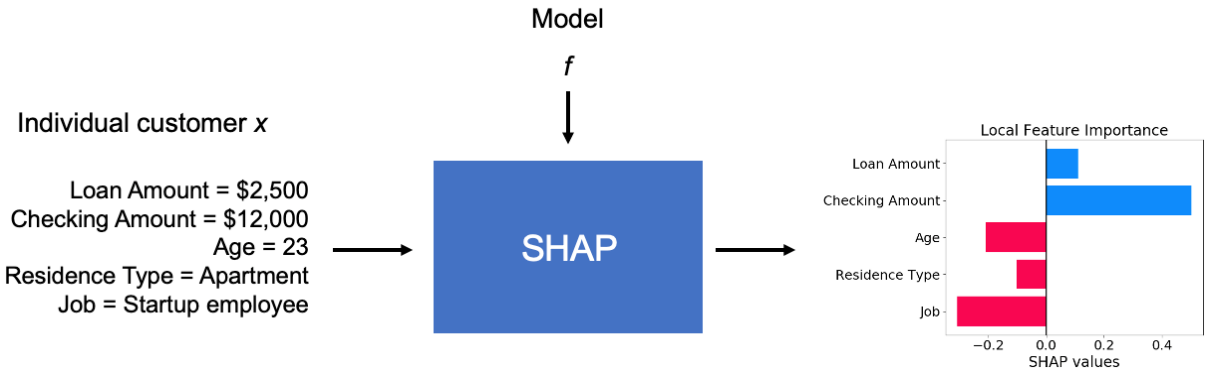


Рисунок 1.4 – Приклад роботи SHAP [9]

Кожен з цих методів має свої переваги та недоліки (таблиця 1.1).

У медичній сфері пояснення рішень ШІ є критично важливими з огляду на такі аспекти:

– підвищення довіри лікарів. Коли фахівці розуміють, на чому ґрунтується рішення моделі, вони охочіше впроваджують її у свою практику;

– покращення якості діагностування. Пояснення можуть указати на додаткові ділянки аналізу, що підвищує імовірність виявлення патологій;

– освітній аспект. Візуалізації та пояснення можуть бути використані для навчання студентів і молодих фахівців [10].

Таблиця 1.1 – Переваги та недоліки методів пояснення результатів

Назва методу:	Переваги:	Недоліки:
Grad-CAM	Візуалізація на рівні пікселів, що особливо корисно для медичних зображень; сумісність з різними архітектурами ЗНМ.	Залежить від обраного шару в мережі; може бути неточним при глибоких шарах.
LIME	Модель-незалежність; простота реалізації та інтерпретації.	Пояснення є локальними і можуть не відображати глобальну поведінку моделі; може бути нестабільним.
SHAP	Теоретично обґрунтований; надає як локальні, так і глобальні пояснення; сумісний з різними типами моделей.	Висока обчислювальна складність, особливо для глибоких нейронних мереж; потребує оптимізації для практичного застосування.

Приклади застосування:

– діагностика пухлин мозку: Використання Grad-CAM для виявлення та пояснення області пухлини на МРТ-зображенні. Це допомагає нейрохірургам при плануванні операції;

– виявлення серцевих захворювань: Застосування SHAP для пояснення прогнозів моделі щодо наявності ішемічної хвороби серця на основі МРТ серця.

Незважаючи на прогрес, залишається низка проблем;

– валідація пояснень: Необхідні стандартизовані методи для оцінки якості та точності пояснень;

– обчислювальні витрати: Методи XAI можуть бути ресурсомісткими, що обмежує їх застосування в реальному часі;

– інтеграція в клінічні робочі процеси: Потрібні зручні інструменти, які легко інтегруються в існуючі системи медичної інформації [11].

Перспективи розвитку

– проєктування більш ефективних методів XAI: Зменшення обчислювальної складності без втрати точності;

– створення спеціалізованих інтерфейсів: Інструменти, адаптовані до потреб медичних фахівців, з інтуїтивно зрозумілим відображенням пояснень;

– мультимодальний аналіз: Поєднання МРТ з іншими видами зображень (КТ, ПЕТ) та даними (клінічні показники) для більш повного пояснення.

Отже, проведений аналіз проблеми оброблення та пояснення результатів класифікації МРТ-зображень свідчить про необхідність проєктування підходів, які б поєднували високу точність з підвищеною інтерпретованістю. Пояснюваний штучний інтелект вважається ключовим чинником, що сприятиме ширшому впровадженню цих технологій у медичну практику, підвищенню довіри з боку медичних працівників і, зрештою, покращенню якості діагностування та лікування пацієнтів [12].

1.2 Аналіз практичних рішень для пояснення результатів класифікації МРТ-зображень

Завдяки розвитку методів штучного інтелекту та глибинного навчання стало можливим не лише класифікувати та сегментувати МРТ-зображення з високою точністю, а й пояснювати отримані результати. Це особливо важливо у медичній практиці, де прозорість та інтерпретованість моделей має вирішальне значення. Нижче розглянуто наявні програмні рішення та інструменти для пояснення

результатів класифікації МРТ-зображень, а також наведено їхні особливості, переваги та недоліки [13].

Captum – це бібліотека для пояснення моделей глибокого навчання, розроблена для PyTorch (рисунок 1.5). Вона підтримує різноманітні методи інтерпретації, серед яких Gradient Shap, DeepLIFT, Integrated Gradients, а також Grad-CAM. Captum дає змогу інтегрувати методи пояснення до наявних моделей та забезпечує зрозумілу для користувачів візуалізацію [14].

Основні характеристики Captum:

– підтримка різних методів інтерпретації (градієнтні та неградієнтні);

– тісна інтеграція з PyTorch для спрощення застосування;

– наявність інструментів для візуалізації атрибуцій, корисна для медичних зображень.

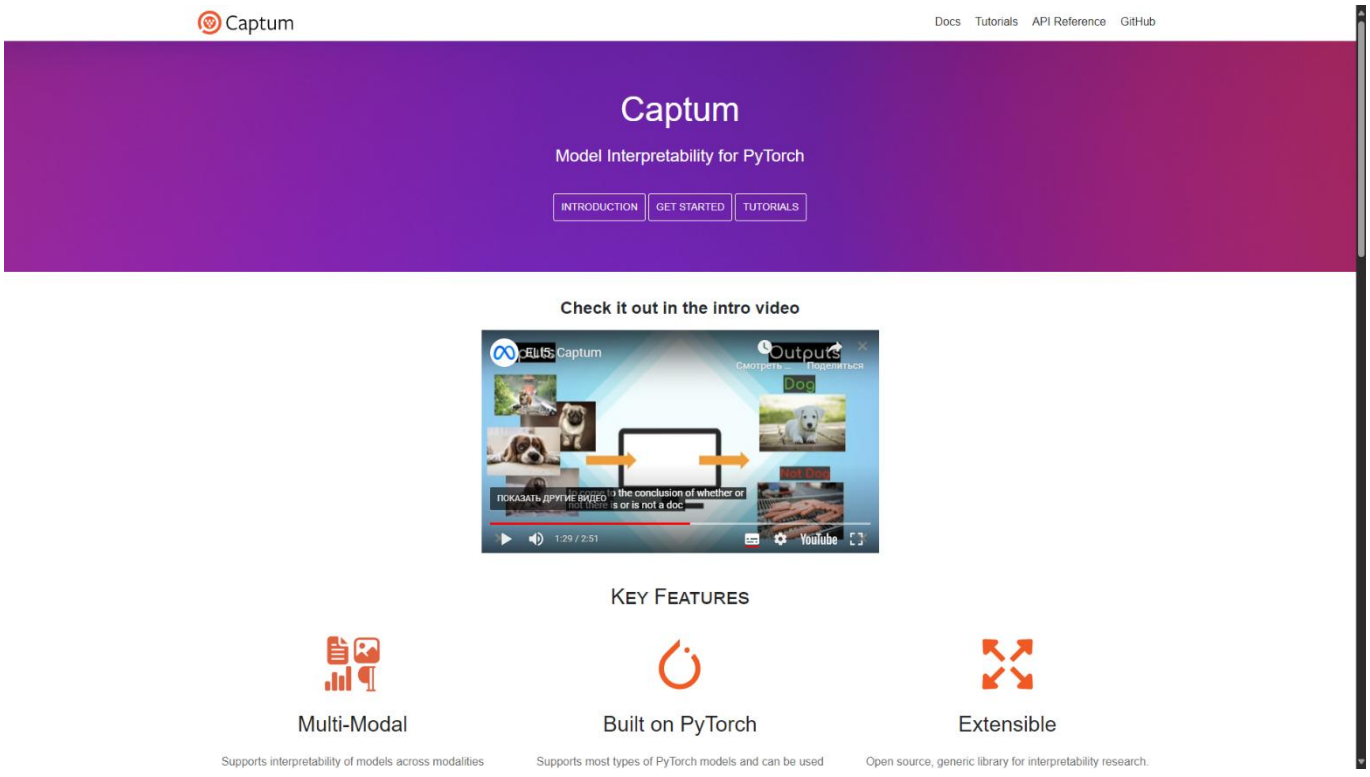


Рисунок 1.5 – Головна сторінка Captum [15]

TF-Explain – бібліотека для пояснення моделей, розроблених на TensorFlow і Keras (рисунок 1.6). Вона реалізує методи Grad-CAM, Occlusion, SmoothGrad та інші. TF-Explain орієнтована на швидку інтеграцію в існуючі процеси [16].

Основні характеристики TF-Explain:

– підтримка TensorFlow та Keras;

– простота використання без необхідності глибоких змін у коді;

– наявність інструментів для створення теплових карт і візуалізацій.

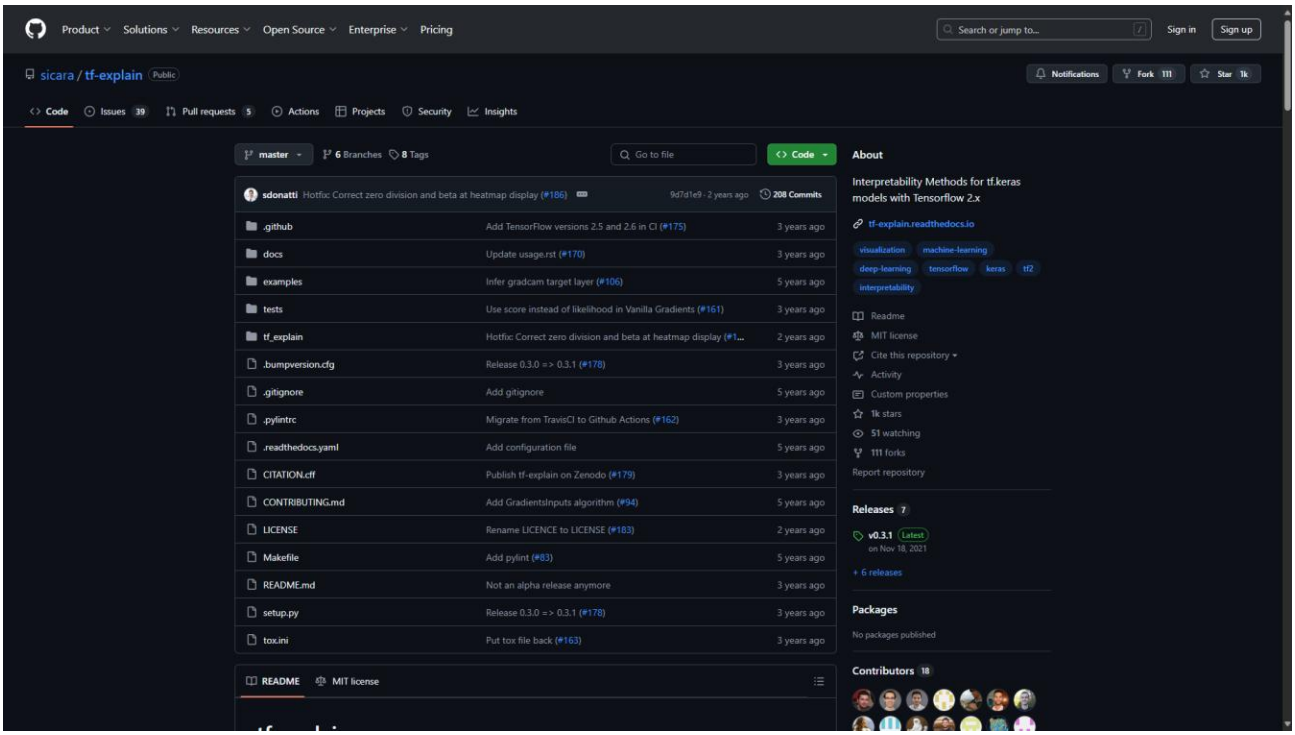


Рисунок 1.6 – Головна сторінка GitHub TF-Explain [17]

LIME (Local Interpretable Model-agnostic Explanations) – модель-незалежний інструмент для локальної інтерпретації моделей (рисуноку 1.7). Він апроксимує поведінку складної моделі простою, полегшуючи розуміння ухвалених рішень у локальному просторі ознак [18].

Основні характеристики LIME:

– універсальність (підтримка різних моделей, включаючи нейронні мережі та дерева рішень);

– простота інтеграції без змін у моделі;

– здатність працювати з різними типами даних (текст, зображення, табличні дані).

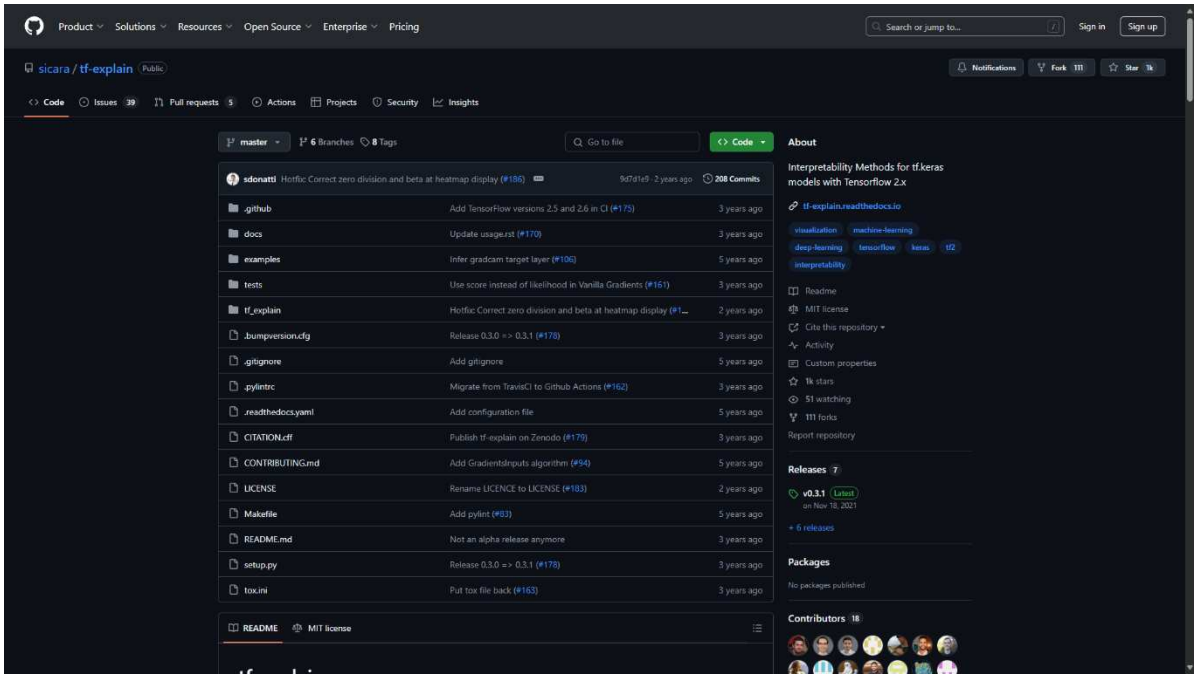


Рисунок 1.7 – сторінка GitHub Lime [19]

SHAP (SHapley Additive exPlanations) – бібліотека для пояснення моделей машинного навчання, заснована на теорії вартості Шеплі (рисунок 1.8). Вона обчислює внесок кожної ознаки у прогноз моделі, надаючи як глобальні, так і локальні пояснення [20].

Основні характеристики SHAP:

– строге теоретичне обґрунтування;

– сумісність із широким спектром моделей і типів даних;

– можливість створення різноманітних графіків та теплових карт.

ELI5 – інструмент для спрощення розуміння моделей машинного навчання та їхніх прогнозів (рисунок 1.9). Він підтримує різні моделі та надає пояснення у зрозумілому форматі [22].

Основні характеристики ELI5:

– сумісність із scikit-learn, XGBoost, LightGBM та іншими моделями;

– можливість інтеграції з LIME та SHAP;

– наявність зрозумілого веб-інтерфейсу.

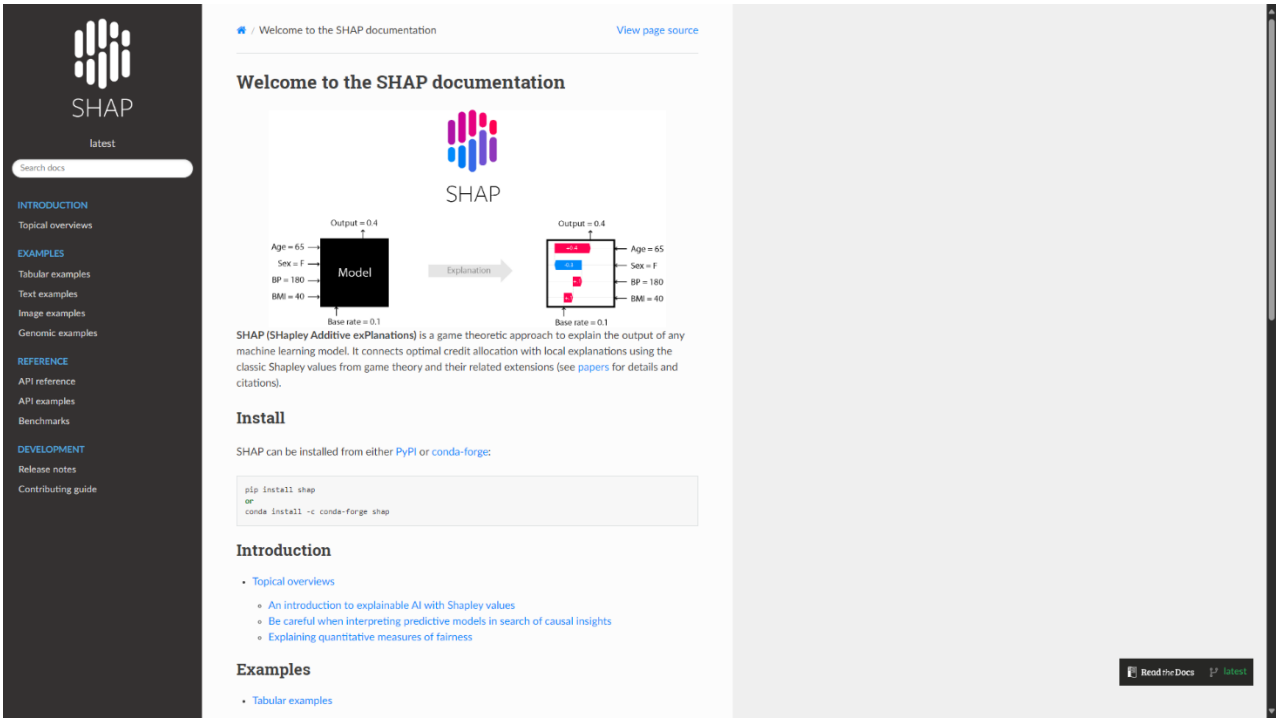


Рисунок 1.8 – Головна сторінка SHAP [21]

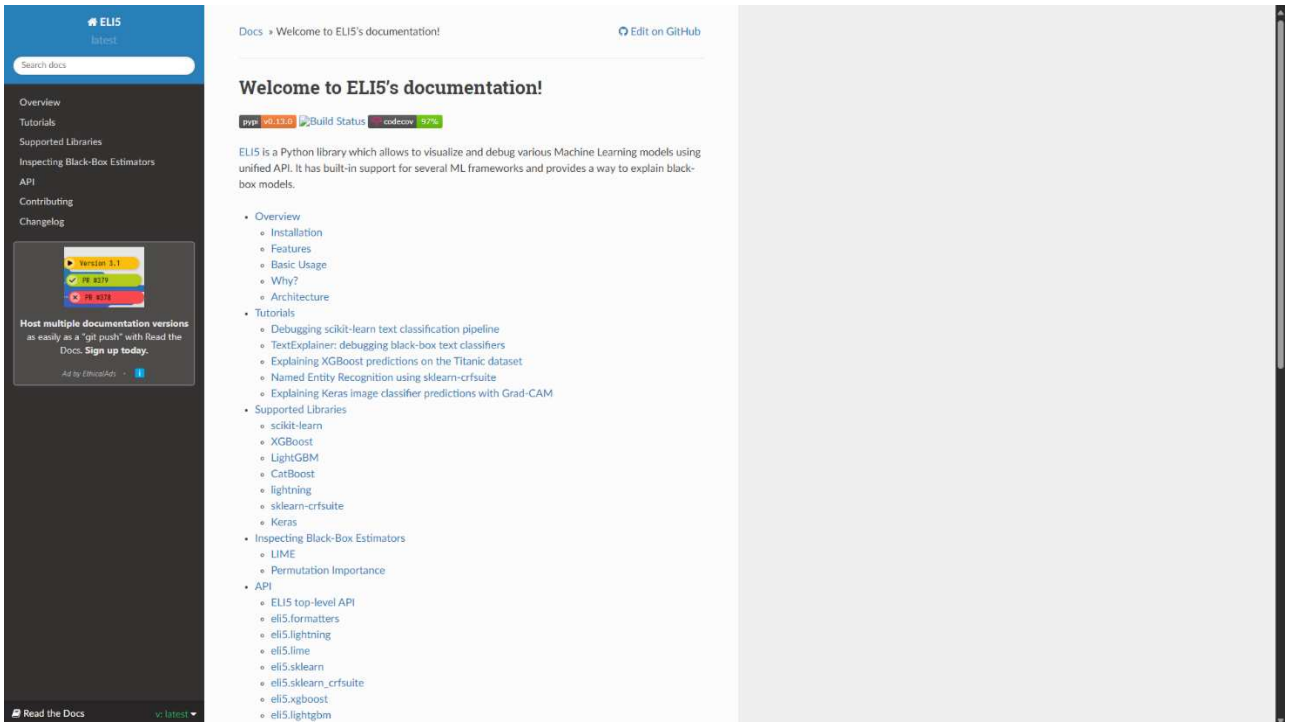


Рисунок 1.9 – Головна сторінка ELI5 [23]

Innvestigate – бібліотека для інтерпретації моделей глибокого навчання, що реалізує такі методи, як LRP (Layer-wise Relevance Propagation), Deep Taylor Decomposition та інші [24] (рисунок 1.10).

Основні характеристики Innvestigate:

– орієнтація на глибокі нейронні мережі;

– різноманітність методів інтерпретації;

– сумісність із моделями на базі Keras.

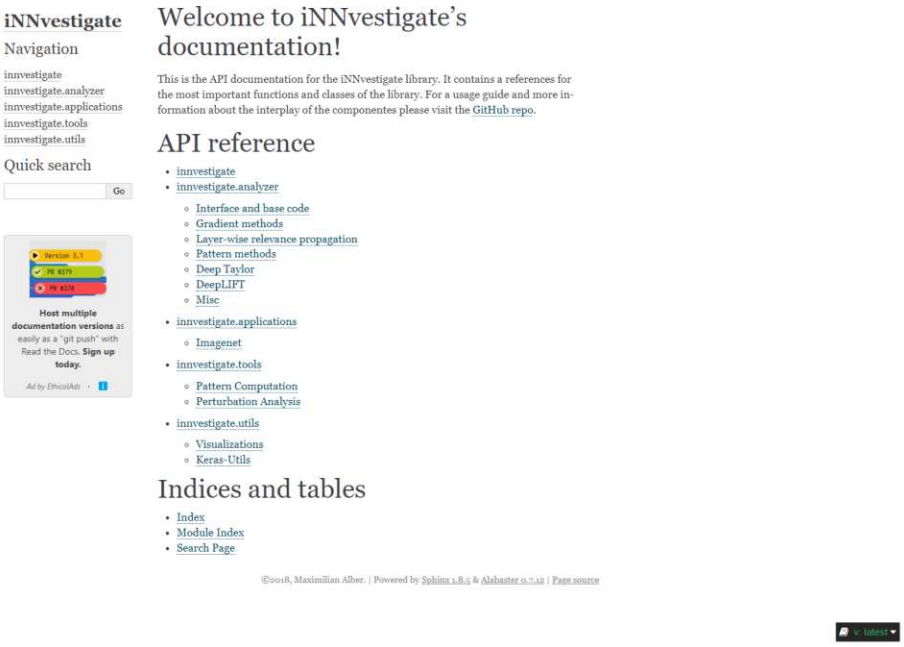


Рисунок 1.10 – Головна сторінка iNNvestigate [25]

У таблиці 1.2 наведено порівняння розглянутих бібліотек за основними критеріями.

Зазначені бібліотеки використовують для пояснення моделей у різних проєктах, зокрема:

– діагностика пухлин мозку. Застосування Captum та Grad-CAM для візуалізації областей мозку, які вплинули на рішення моделі;

– аналіз серцевих захворювань. Використання SHAP для оцінки внеску різних областей серця у прогноз моделі щодо кардіоміопатії;

– виявлення розсіяного склерозу. Залучення LIME для пояснення класифікації вогнищ на МРТ-зображеннях [26];

Таблиця 1.2 – Порівняння бібліотек для пояснення моделей

Метод	Підтримувані методи	Сумісність	Переваги	Недоліки
Captum	Grad-CAM, LRP, Integrated Gradients	PyTorch	Глибока інтеграція з PyTorch; широкий набір методів	Обмежена сумісність з іншими платформами
TF-Explain	Grad-CAM, Occlusion, SmoothGrad	TensorFlow, Keras	Простота використання; швидка інтеграція	Менший набір методів порівняно з іншими бібліотеками
LIME	LIME	Модель-незалежна	Універсальність; простота реалізації	Локальність пояснень; може бути повільною
SHAP	SHAP values	Модель-незалежна	Теоретична основа; детальні пояснення	Висока обчислювальна складність для великих моделей
ELI5	LIME, SHAP, моделі scikit-learn	scikit-learn, XGBoost	Зручний інтерфейс; підтримка різних моделей	Обмежена підтримка глибоких нейронних мереж
Innvestigate	LRP, Deep Taylor Decomposition та ін.	Keras	Глибока інтерпретація ЗНМ; різноманітність методів	Обмежена сумісність; складність реалізації
Бібліотека	Підтримувані методи	Сумісність	Переваги	Недоліки

Попри наявність широкого спектра інструментів, існує низка викликів, пов’язаних із їхнім використанням:

– обчислювальні ресурси. Деякі методи (наприклад, SHAP) вимагають значних ресурсів, що ускладнює їх застосування в реальному часі;

– складність інтерпретації. Отримані пояснення можуть бути доволі складними, особливо для користувачів без досвіду в галузі ШІ;

– відсутність стандартів. Нині немає уніфікованих критеріїв оцінки якості пояснень, що може призводити до різних інтерпретацій [27].

Для розв’язання зазначених проблем ведуться дослідження спрямовані на:

– оптимізацію методів. Робота над зниженням обчислювальної складності методів XAI без втрати точності;

– спеціалізовані інтерфейси. Створення інструментів з інтуїтивно зрозумілим інтерфейсом, адаптованим до потреб медичних працівників;

– інтеграція в клінічні системи. Впровадження інструментів XAI в електронні медичні записи та інші медичні інформаційні системи;

Отже, аналіз практичних рішень свідчить про наявність широкого спектра інструментів та бібліотек для пояснення результатів класифікації МРТ-зображень. Вони дають змогу підвищити прозорість і зрозумілість моделей глибинного навчання, що має важливе значення для медичної сфери. Водночас залишаються питання щодо обчислювальних ресурсів, інтерпретації результатів та відсутності загальноприйнятих стандартів. Подальший розвиток інструментів та методів XAI сприятиме ширшому впровадженню штучного інтелекту в медицину, удосконаленню якості діагностування та лікування.

1.3 Актуальність використання згорткових нейронних мереж для інтерпретування результатів виявлення з метою пояснення результатів класифікації МРТ

Інтерпретація результатів, одержаних за допомогою нейронних мереж під час аналізу МРТ серця, є важливим елементом процесу діагностування серцево-

судинних захворювань. МРТ належить до найбільш ефективних методів отримання високоточної візуалізації серця, що дає змогу виявляти ранні ознаки патологій, включаючи ішемічну хворобу, кардіоміопатії та аномалії клапанів. Водночас результати глибоких нейронних мереж (ЗНМ), які застосовуються для аналізу таких даних, часто важко інтерпретувати через недостатню прозорість механізмів прийняття рішень [28].

Серед основних архітектур ЗНМ, що застосовуються для інтерпретації результатів, виділяють:

1. ResNet (Residual Networks). Ця архітектура була розроблена для розв'язання проблеми згасання градієнтів у глибоких нейронних мережах. Використання залишкових блоків дає можливість передавати частину інформації безпосередньо через шари, зберігаючи релевантні ознаки навіть на великій глибині. Це істотно підвищує успішність вилучення ознак, що особливо важливо для складних зображень, зокрема МРТ серця [29].

Для інтерпретації результатів ResNet застосовують методи Grad-CAM (Gradient-weighted Class Activation Mapping), SHAP (SHapley Additive exPlanations) та LIME (Local Interpretable Model-agnostic Explanations). Grad-CAM візуалізує впливові області зображення у вигляді теплової карти, а SHAP і LIME надають кількісне оцінювання внеску окремих ознак у кінцеве рішення.

Переваги**:**

– висока точність класифікації;

– підтримка інтерпретаційних методів.

Недоліки: висока обчислювальна складність, особливо для великих наборів даних [30].

2. DenseNet (Densely Connected Convolutional Networks). DenseNet використовує густі зв'язки між шарами, завдяки чому кожен шар отримує дані від усіх попередніх, що підвищує ефективність використання параметрів. Це зменшує потребу в більшій кількості параметрів. Проте щільна структура значно ускладнює інтерпретацію рішень моделі, особливо для медичних даних високої складності [31].

DenseNet сумісна з Grad-CAM, SHAP і LIME, однак застосування цих методів може бути ускладненим через складність структури.

Переваги**:**

– менша кількість параметрів порівняно з іншими архітектурами;

– ефективне використання ресурсів.

Недоліки: складність інтерпретації через велику кількість зв’язків між шарами [32].

3. VGG (Visual Geometry Group Networks). Архітектура VGG вирізняється простою структурою та єдиними розмірами фільтрів у всіх шарах. Це спрощує інтерпретацію результатів, що є важливим у задачах, де потрібно забезпечити зрозумілість та прозорість рішень. Проте VGG має обмежений потенціал для детального аналізу складних структур, які властиві МРТ-зображенням [33].

Grad-CAM, SHAP і LIME також застосовуються для інтерпретації результатів VGG, але можливості для глибокого аналізу обмежені через відносно невелику кількість параметрів.

Переваги**:** простота і легкість інтерпретації.

Недоліки**:** обмежені можливості для глибокого аналізу через малу кількість параметрів [34].

4. Inception Networks. Архітектура Inception використовує блоки з різними розмірами фільтрів одночасно, що надає гнучкість у виявленні ознак різних масштабів. Цей підхід є ефективним для завдань візуальної класифікації, оскільки забезпечує оптимальне використання ресурсів. Проте складна багаторівнева структура ускладнює інтерпретацію результатів [35].

Grad-CAM, SHAP і LIME можуть бути застосовані для пояснення рішень Inception, проте через складність архітектури це потребує значних обчислювальних ресурсів.

Переваги**:** гнучкість та можливість паралельної оброблення ознак різних масштабів.

Недоліки**:** висока обчислювальна складність та складність інтерпретації [36].

Щодо інтерпретаційних методів варто виділити наступні:

1. Grad-CAM дає змогу візуалізувати впливові області зображення у формі теплової карти, що дає змогу лікарю зрозуміти причини ухвалення конкретного рішення моделлю. Цей метод застосовують для різних архітектур ЗНМ, включаючи ResNet, DenseNet, VGG та Inception [37].

2. SHAP, базований на теорії ігор, оцінює внесок кожної ознаки або пікселя у загальний прогноз. Він забезпечує як локальні, так і глобальні пояснення для моделей, що робить його універсальним інструментом для інтерпретації рішень ResNet, DenseNet та VGG [38].

3. LIME створює спрощені моделі для пояснення рішень нейронних мереж у локальних областях простору ознак. Цей метод добре підходить для пояснення рішень на рівні окремих прикладів, що є корисним у медичних застосуваннях [39].

Порівняння архітектур подано в таблиці 1.3.

Таблиця 1.3 – Порівняння архітектур згорткових нейронних мереж

Метод	Переваги	Недоліки
ResNet	Висока точність, підтримка Grad-CAM, SHAP, LIME	Висока обчислювальна складність
DenseNet	Менша кількість параметрів, ефективне використання ресурсів	Складна інтерпретація
VGG	Простота, легкість інтерпретації	Обмежені можливості для глибокого аналізу
Inception	Гнучкість, ефективне використання ресурсів	Висока складність архітектури, складна інтерпретація

Результати досліджень свідчать про те, що архітектура ResNet виявляється найбільш придатною для інтерпретації результатів класифікації МРТ-зображень, поєднуючи високу точність з наявністю інструментів для інтерпретації (Grad-CAM,

SHAP, LIME). Утім, кожна з розглянутих архітектур має власні переваги та недоліки, які необхідно враховувати під час вибору моделі для конкретних завдань [40].

1.4 Постановка задачі

Аналіз медичних зображень, зокрема МРТ серця, є складним і трудомістким процесом, що вимагає високої кваліфікації фахівців. Зростання обсягів таких даних, а також їхня складна структура, роблять автоматизацію процесу аналізу актуальною задачею. Сучасні методи глибокого навчання, такі як ЗНМ, дають можливість автоматизувати виявлення патологій серця, проте їхня недостатня прозорість обмежує їхнє широке застосування в клінічній практиці. Виникає потреба у проєктуванні методів інтерпретації результатів, що забезпечують як високу точність класифікації, так і зрозумілість рішень для медичних фахівців.

Метою даної роботи є підвищення рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання. Досягнення мети роботи передбачає виконання таких задач:

1. Провести аналіз моделей, методів та технологій глибокого навчання для оброблення та аналізу зображень МРТ.
2. Вдосконалити метод виявлення візуальних патологій серця за зображенням МРТ з використанням згорткової нейронної мережі.
3. Спроєктувати метод інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих від згорткової нейронної мережі.
4. Реалізувати метод інтерпретування у вигляді вебзастосунку.
5. Провести експериментальне тестування реалізованого модуля за еталонними наборами даних.

РОЗДІЛ 2. Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ

2.1 Проєктування методу інтерпретування результатів виявлення патологій серця за зображенням МРТ

Метод інтерпретації результатів виявлення патологій серця є ключовим етапом автоматизованої діагностування, що не лише виявляє патологічні зміни, але й надає обґрунтовані пояснення щодо рішень моделі, забезпечуючи прозорість її роботи. Застосування алгоритмів глибинного навчання та технік інтерпретації дає змогу отримувати результати, які підвищують довіру медичних фахівців до автоматизованих систем.

На рисунку 2.1 представлено схему методу інтерпретації результатів.

Рисунок 2.1 – Схема покращеного методу інтерпретації результатів виявлення патологій на зображеннях МРТ

Поданий на рисунку 2.1 метод включає низку послідовних етапів, кожен з яких спрямований на забезпечення точного й зрозумілого аналізу МРТ-зображень серця. Вхідними даними є МРТ-зображення серця та попередні результати від нейронної мережі, що виконує первинне виявлення аномалій.

Вхідні дані:

– МРТ-зображення серця у форматах JPEG, PNG або DICOM;

– результати первинного аналізу нейронної мережі: сегментовані області серця та попередні оцінки можливих патологій.

Метод реалізується у таких кроках:

Крок 1: Завантаження та попередня обробка зображення:

На цьому етапі система приймає вхідне МРТ-зображення та проводить його обробку для забезпечення сумісності з моделлю:

– завантаження файлу: Вебінтерфейс Gradio дає змогу завантажувати файли формату JPEG, PNG або DICOM. Перевіряється коректність формату, і в разі помилки користувач отримує відповідне повідомлення;.

– фільтрація шумів: Застосовується алгоритм згладжування, наприклад, Gaussian Blur, для зменшення артефактів.

Формула згладжування (2.1):

$$I'(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} * I(x,y), \quad (2.1)$$

де σ – параметр, що визначає інтенсивність згладжування.

– нормалізація інтенсивності: Приведення піксельних значень до стандартного діапазону для відповідності вхідним даним моделі (2.2):

$$I_{norm}(x,y) = \frac{I(x,y)-\mu}{\sigma}, \quad (2.2)$$

де μ – середнє значення інтенсивності, а σ – стандартне відхилення.

– масштабування: Зміна розміру зображення до стандартного розміру (наприклад, 224×224) для аналізу моделлю ResNet50.

Крок 2: Сегментація області серця:

Для автоматичного виділення серцевих зон застосовується модель U-Net. Цей крок дає змогу виділити основні структури серця (міокард, перикард), ігноруючи нерелевантні області.

Основні особливості U-Net:

– архітектура: Модель складається з енкодера (зменшує розмір зображення) і декодера (відновлює розмір із деталями);

– пропускні зв'язки (skip-connections): Передають важливу інформацію від енкодера до декодера, зберігаючи просторові деталі.

Результат: виділені області серця відображаються у вигляді сегментованих масок, які передаються на класифікацію.

Крок 3: Класифікація патологій

Модель ResNet50 аналізує сегментовані зображення та визначає наявність патологій, класифікуючи їх на три класи:

– Normal: Нормальний стан;

– Hypertrophy: Гіпертрофія міокарда;

– Infarction: Інфаркт міокарда.

Формула Softmax (2.3):

$$P(y = c \mid x) = \frac{e^{z_c}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.3}$$

де z_c – логіт для класу c, а K – кількість класів.

Вихід: Вектор ймовірностей для кожного класу, що дає змогу визначити патологію з максимальною ймовірністю.

Крок 4: Інтерпретація результатів:

Для пояснення результатів застосовуються Grad-CAM і SHAP:

– Grad-CAM: Генерує теплові карти, які демонструють, які ділянки зображення найбільше вплинули на рішення моделі.

Формула Grad-CAM (2.4):

$$L^{e}_{\mathrm{LGrad-CAM}} = \mathrm{ReLU}\left(\sum\nolimits_{k} (a_k^c) A^k\right), \tag{2.4}$$

де a_k^c – вагові коефіцієнти, що визначають вплив k-го каналу на клас c, A^k – активації в k-му каналі.

SHAP аналізує внесок кожного пікселя у прийняте рішення, створюючи текстові пояснення.

Результат: теплові карти з Grad-CAM та текстові пояснення з SHAP надають лікарям детальну інформацію про впливові області.

Крок 5: Збереження результатів

На цьому етапі забезпечується збереження результатів аналізу та інтерпретації для подальшого використання.

Основні етапи:

Формати збереження:

– DICOM: для інтеграції з медичними системами;

– JPEG/PNG: для графічного представлення результатів;

– JSON/CSV: для збереження текстових пояснень у структурованому вигляді.

Обробка перед збереженням**:**

– накладення теплових карт Grad-CAM на зображення (2.5):

$$I_{result}(x,y) = \alpha \cdot I_{\text{original}}(x,y) + (1-\alpha) \cdot I_{\text{heatmap}}(x,y); \quad (2.5)$$

– генерація текстових пояснень SHAP у вигляді JSON-структур.

Місця збереження**:**

– локальний диск або сервер зберігання;

– база даних для швидкого доступу через API.

Повідомлення користувача**:**

– інформація про успішне збереження або виявлені помилки.

Результат:

– збережені класифікації, теплові карти та текстові пояснення;

– інтеграція з медичними системами для доступу лікарів.

Таким чином, МРТ є одним із провідних діагностичних інструментів, а застосування пояснюваних методів дає змогу підвищити довіру лікарів до використання штучного інтелекту у медичній діагностиці. Інтеграція таких підходів

у клінічну практику сприяє підвищенню точності та оперативності діагностування, що позитивно впливає на якість медичної допомоги пацієнтам.

2.2 Проєктування архітектури згорткових нейронних мереж для виявлення патологій

Архітектура згорткових нейронних мереж для виявлення патологій серця на МРТ-зображеннях була розроблена з урахуванням вимог до точності, можливості інтерпретації результатів та ефективності обчислень. У межах проєкту інтегровано дві основні нейронні мережі – U-Net і ResNet50, що забезпечують повний цикл аналізу: від сегментації серцевих областей до класифікації патологій з подальшою інтерпретацією одержаних результатів.

U-Net застосовували для задачі сегментації серця, під час якої необхідно було виділити ключові анатомічні структури, зокрема міокард і перикард. Архітектура U-Net складається з енкодера та декодера. Енкодер, сформований з кількох послідовних згорткових блоків, функцій активації ReLU та шарів MaxPooling, зменшує розмірність вхідного зображення й витягує високорівневі ознаки.

Формула згортки (2.6):

$$y(i,j) = \sum_m \sum_n x(i+m, j+n) \cdot w(m,n) + b, \tag{2.6}$$

де x(i,j) – значення пікселя, $w(m,n)$ – ваги фільтра, а b – зсув.

Формула функції активації ReLU (2.7):

$$f(x) = \max(0, x). \tag{2.7}$$

Декодер відновлює розмірність зображення за допомогою транспонованих згорток. Пропускні зв’язки (skip-connections) між відповідними шарами енкодера й декодера дають можливість передавати контекстну інформацію та покращувати точність сегментації.

Формула функції втрат Dice для сегментації (2.8):

$$L_{Dice} = 1 - \frac{2|A \cap B|}{|A| + |B|}, \tag{2.8}$$

де A – передбачена моделлю маска, а B – справжня маска.

Результати сегментації, отримані від U-Net, передавалися на вхід ResNet50 для класифікації. ResNet50 була обрана через її здатність витягувати складні ознаки завдяки залишковим блокам.

Формула залишкового блоку ResNet (2.9):

$$y = F(x, \{W_i\}) + x, \tag{2.9}$$

де x – вхід, $F(x, \{W_i\})$ – операція згортки, а y – вихід.

Формула функції втрат перехресної ентропії для класифікації (2.10):

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)], \tag{2.10}$$

де y_i – справжня мітка класу, p_i – передбачена ймовірність, N – кількість зразків.

У процесі інтеграції U-Net та ResNet50 ResNet50 використовували як “backbone” для енкодера U-Net. Такий підхід забезпечив не лише точне виділення релевантних зон серця, а й класифікацію на основі складних ознак. Сегментація, реалізована U-Net, зберігала точну локалізацію патологічних ділянок, а ResNet50 відповідала за якісний аналіз ознак для визначення типу патології.

Після класифікації результатів застосовано Grad-CAM для створення теплових карт, які вказують на ділянки зображення, що найбільше вплинули на рішення моделі. Додатково використовували SHAP для формування текстових пояснень, які детально описували внесок кожного пікселя в остаточне рішення моделі. Це надало лікарям можливість не лише ознайомитися з тепловими картами, а й отримати текстову інтерпретацію, що підвищило довіру до автоматизованої системи.

Запропонована архітектура була протестована на наборі МРТ-зображень з відомими патологіями. Сегментація продемонструвала середню точність Dice на рівні 92%, а класифікація досягла значень Precision і Recall понад 85%. Інтеграція

Grad-CAM та SHAP забезпечила візуалізацію та пояснення результатів, що допомогло лікарям оперативно оцінити стан пацієнтів та ухвалювати рішення.

Таким чином, створена архітектура на основі U-Net і ResNet50 з використанням інтерпретаційних методів виявилася ефективним інструментом для автоматизованої діагностування серцевих патологій за МРТ-зображеннями, гарантувавши не лише точність виявлення патологій, а й прозорість прийнятих рішень, що є надзвичайно важливим для медичної практики.

2.3 Навчання згорткової нейронної мережі для виявлення патологій

Процес навчання згорткової нейронної мережі (ЗНМ) для виявлення патологій на МРТ-зображеннях серця базувався на використанні сегментованих зображень, де кожен піксель або ділянка були позначені як патологічні або здорові. Під час дослідження було виконано кілька ключових етапів, що охоплювали підготовку даних, побудову архітектури моделі, навчання, підбір параметрів та тестування моделі на нових даних.

Для навчання використовувався медичний набір даних Automated Cardiac Diagnosis Challenge (ACDC) [41]. Цей набір містить високоякісні МРТ-зображення серця з експертними мітками (рисунок 2.2), що гарантувало належну точність та достовірність навчання.

Набір даних ACDC містить чотири класи:

– Normal – зображення здорового серця;

– Hypertrophy – патологія гіпертрофії міокарда;

– Infarction – наслідки інфаркту міокарда;

– Other – інші патології.

Загальна кількість зображень у наборі:

– 1912 для тренування;

– 200 для валідації;

– 100 для тестування.

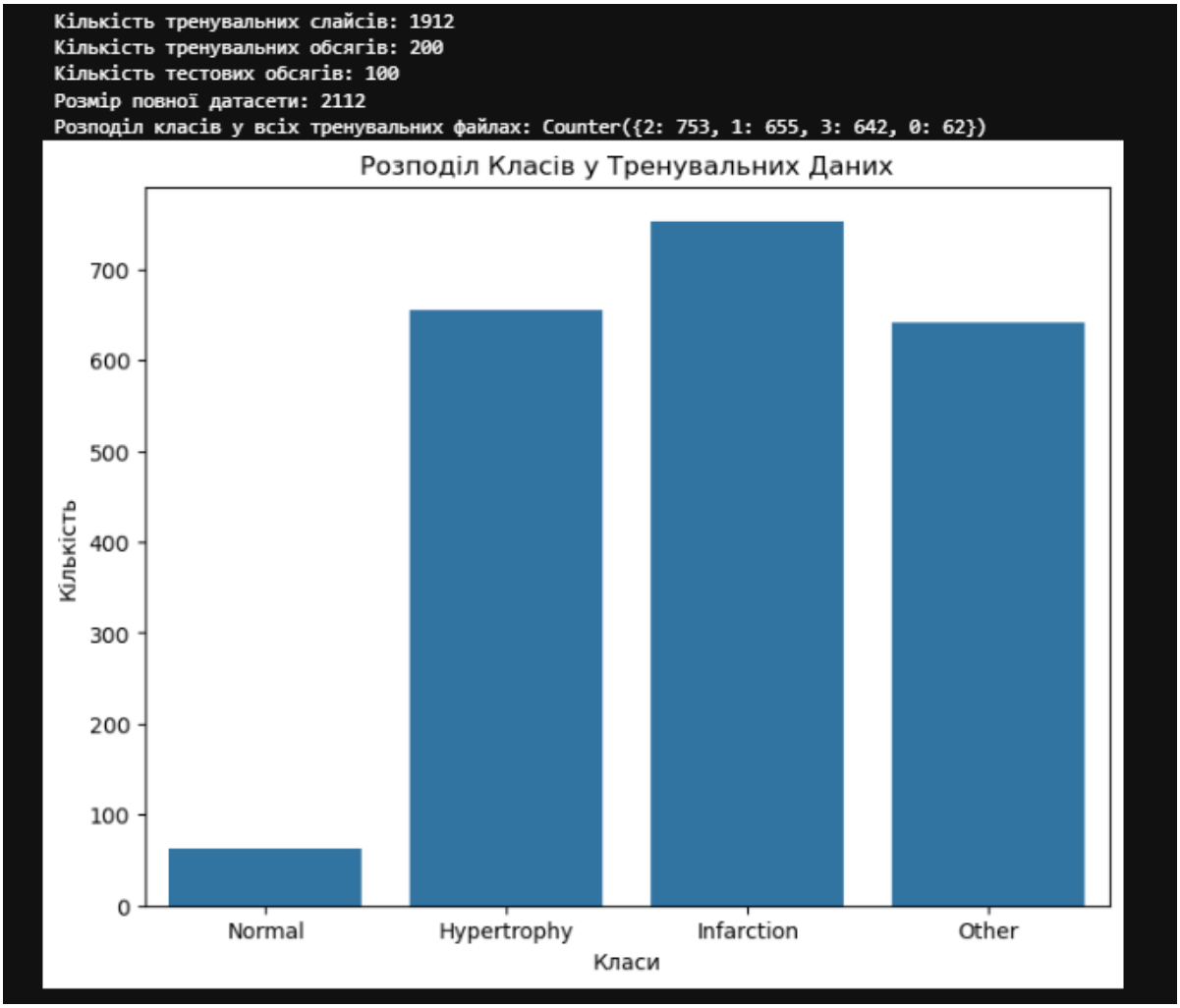


Рисунок 2.2 – Розподіл класів у тренувальних даних

Серед тренувальних даних спостерігався дисбаланс: Normal (62), Hypertrophy (655), Infarction (753), Other (642). Для корекції цього дисбалансу застосовували вагові коефіцієнти у функції втрат.

Перший етап – підготовка даних: на першому етапі було підготовлено великий набір МРТ-зображень серця, на яких попередньо було проведено сегментацію ділянок серця з використанням архітектури U-Net. Крім того, було зібрано дані про патології, включаючи маски, що вказують на патологічні зони, а також здорові ділянки для правильного порівняння. Маски були точно позначені медичними експертами для забезпечення високої точності.

Під час підготовки даних було виконано наступні дії:

– аугментація даних: через обмежену кількість медичних даних було використано методи аугментації (обертання, зміна масштабу, відбиття), що дало змогу збільшити розмір вибірки та покращити узагальнюючі властивості моделі;

– нормалізація зображень: для усунення різниць у контрасті та яскравості зображень було застосовано нормалізацію інтенсивності пікселів, що дало змогу зробити дані більш однорідними та полегшити процес навчання мережі.

Другий етап – побудова архітектури згорткової нейронної мережі. Для виявлення патологій було побудовано згорткову нейронну мережу, яка базувалася на архітектурі ResNet. Ця модель використовувалася як енкодер для витягування ключових ознак із сегментованих зображень.

Архітектура складається з:

– згорткових шарів (Conv2D): використовуються для витягування ознак зображення. Застосовуються фільтри для розпізнавання різних характеристик зображення на різних рівнях абстракції;

– пулінг-шарів (MaxPooling): зменшують розмір ознак і видаляють незначні деталі, що дає змогу моделі зосередитись на важливих характеристиках;

– повнозв’язаних шарів: використовуються для класифікації. Ці шари допомагають моделі приймати рішення на основі витягнутих ознак – чи присутня патологія на зображенні чи ні;

Для кінцевої класифікації було використано функцію softmax, яка дала змогу отримати ймовірність наявності або відсутності патології.

3. Третій етап – процес навчання: На цьому етапі було проведено навчання ЗНМ для розпізнавання патологічних змін у тканинах серця на основі масок та сегментованих зображень. Навчання мережі здійснювалося за допомогою методу зворотного поширення помилки та оптимізації вагових коефіцієнтів, що дало змогу поступово покращити здатність мережі до класифікації патологій.

Кроки навчання:

– функція втрат: Для навчання моделі було використано функцію бінарної перехресної ентропії для двокласової задачі (наявність або відсутність патології). Такий вибір функції втрат дав змогу точніше розпізнавати патологічні зміни на

зображеннях. У випадку багатокласової задачі застосовувалася категоріальна перехресна ентропія.

Формула для бінарної перехресної ентропії виглядає наступним чином (2.11):

$$\mathrm{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - \mathrm{y}_i)\log(1 - \mathrm{p}_i)], \quad (2.11)$$

де y_i – це істинне значення (0 або 1), p_1 – це ймовірність, передбачена моделлю для класу 1.

– оптимізація: Для оновлення ваг використовують алгоритми оптимізації, такі як Adam або SGD (стохастичний градієнтний спуск), що дають можливість мінімізувати функцію втрат і покращувати здатність мережі до виявлення патологій;

– метрики оцінки: Для оцінки ефективності роботи моделі під час навчання використовуються метрики, такі як точність, чутливість, специфічність, F1-оцінка. Це дає змогу зрозуміти, наскільки добре модель здатна виявляти патології й уникати хибнопозитивних та хибнонегативних результатів.

4. Четвертий етап – після первинного навчання було проведено оцінку якості роботи моделі на тестових даних, що не використовувалися під час тренування. Основними показниками для оцінки були точність, чутливість (sensitivity) та специфічність (specificity). Якщо результати не задовольняли вимоги точності, було проведено повторне навчання з коригуванням параметрів моделі.

Коригування включало:

– зміну архітектури(додавання додаткових згорткових шарів або використання більш глибокої моделі);

– налаштування гіперпараметрів, таких як швидкість навчання, розмір пакетів (batch size) або кількість епох.

5. П’ятий етап – застосування моделі до нових даних: після завершення навчання модель була застосована для аналізу нових МРТ-зображень. Модель провела аналіз сегментованих зображень і визначила ймовірність наявності патології на основі витягнутих ознак.

Основні етапи застосування:

– завантаження нових зображень і застосування попередньо навченого енкодера для витягування ознак;

– класифікація з використанням повнозв'язаних шарів і отримання результатів виявлення патології (наявність або відсутність);

– виведення результатів у вигляді ймовірності або теплових карт (наприклад, за допомогою Grad-CAM), що дає змогу візуалізувати зони патологій;

– навчання здійснювалося з використанням 5-кратна крос-валідація, що забезпечило всебічне оцінювання моделі. На кожному фолді модель тренувалася протягом 30 епох (Рисунок 2.3 – 2.12).

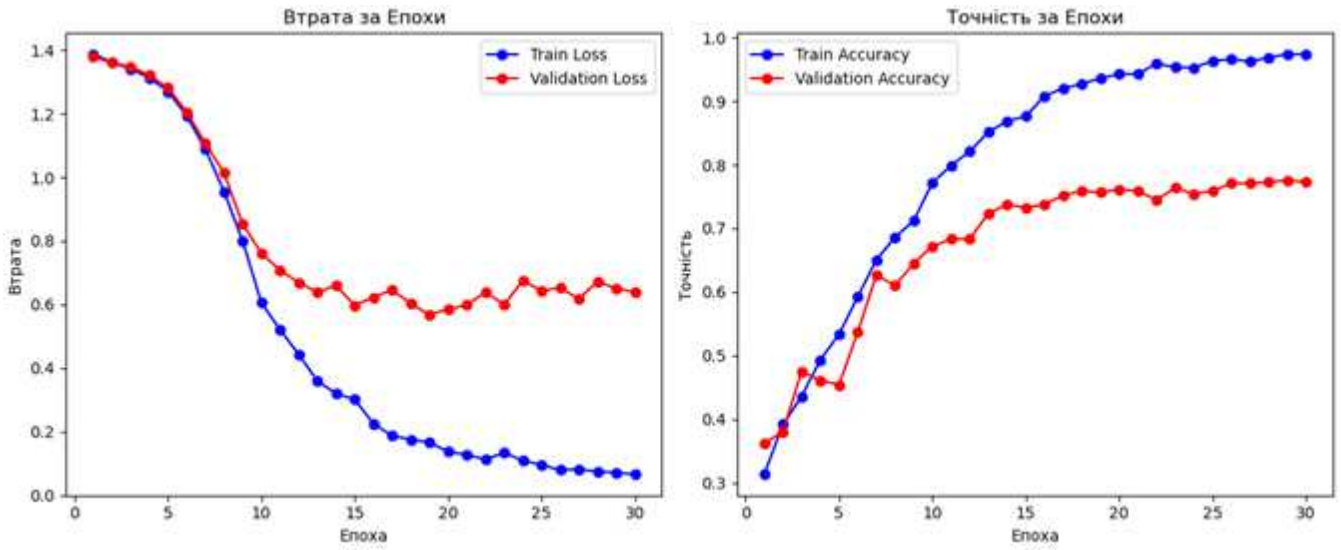


Рисунок 2.3 – Графік тренування моделі на 1 підмножині

При навчанні на першій підмножині модель демонструє стабільне покращення результатів із кожною наступною епохою протягом усіх 30 епох.

– втрати: Значення тренувальних втрат постійно зменшуються, досягаючи мінімуму до кінця навчання. Валідаційні втрати стабілізуються після 15 епох, що свідчить про відсутність перенавчання;

– точність: Тренувальна точність поступово зростає, досягаючи ~90% до останньої епохи. Валідаційна точність стабілізується на рівні ~72%, що вказує на хорошу узагальнюючу здатність моделі;

– Precision і Recall: Ці метрики поступово покращуються протягом навчання, демонструючи, що модель ефективно знаходить баланс між виявленням патологій і уникненням хибнопозитивних результатів;

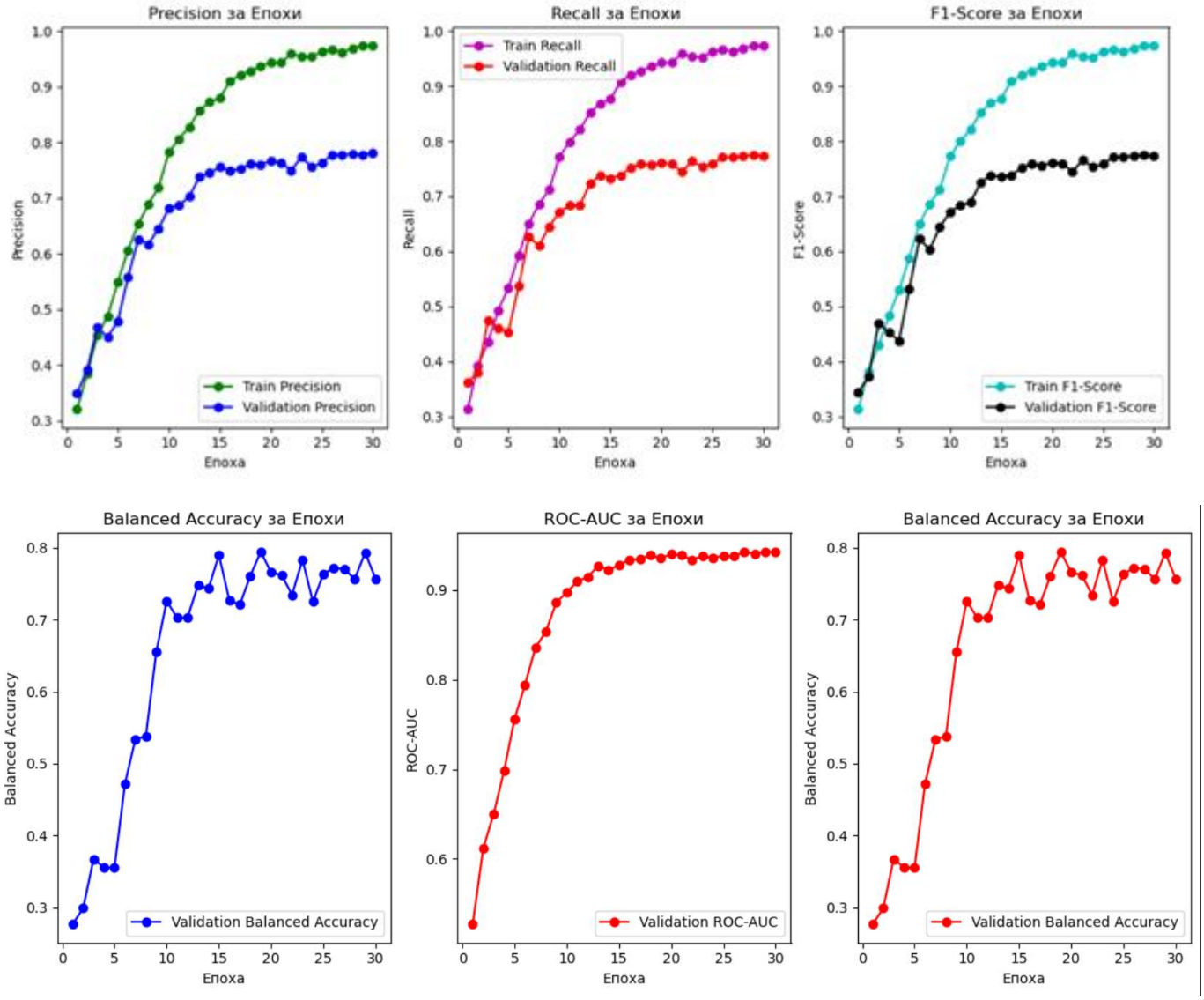


Рисунок 2.4 – Графіки метрик оцінювання моделі на 1 підмножині

– ROC-AUC: На цій підмножині площа під кривою ROC-AUC перевищує 0.93, що свідчить про високий рівень дискримінації між класами.

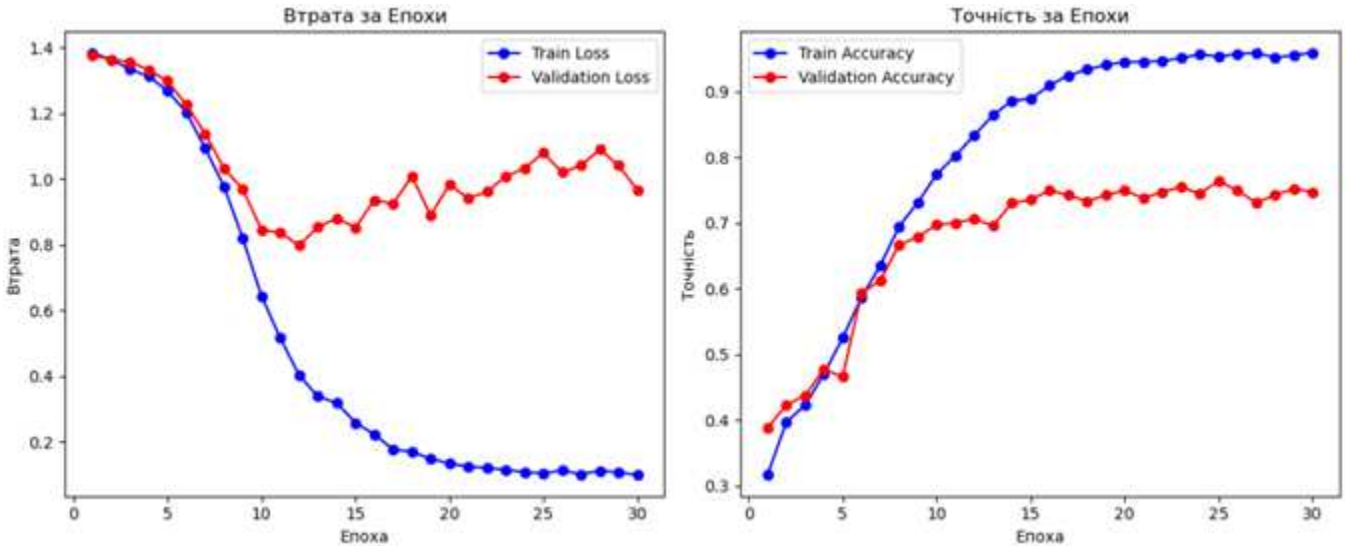


Рисунок 2.5 – Графік тренування моделі на 2 підмножині

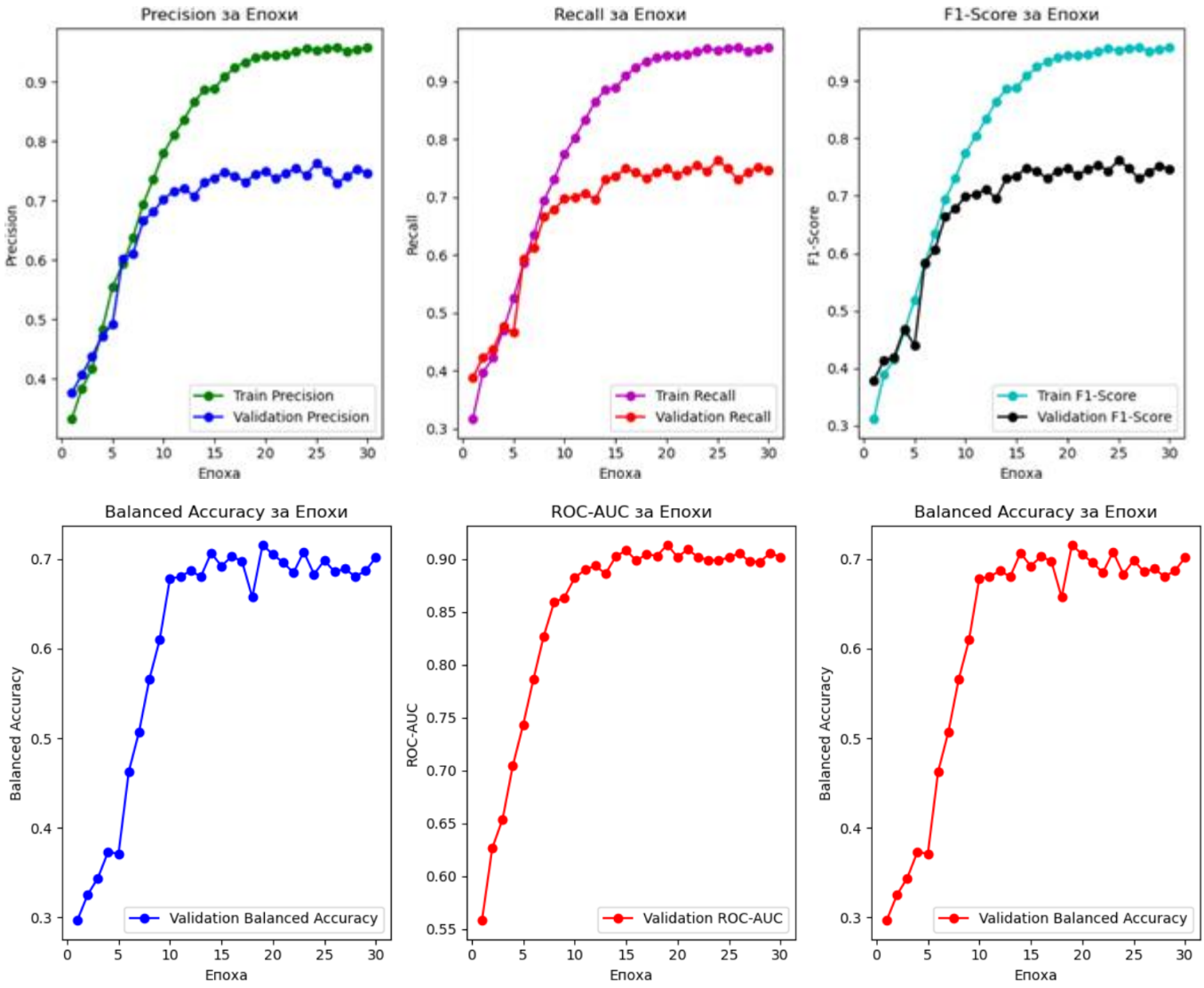


Рисунок 2.6 – Графіки метрик оцінювання моделі на 2 підмножині

Навчання моделі у другій підмножині також демонструє стабільний прогрес протягом 30 епох.

– врати: Зменшення тренувальних втрат є плавним і без значних коливань, що свідчить про поступове навчання моделі. Валідаційні втрати стабілізуються приблизно після 10-ї епохи, вказуючи на оптимальний процес навчання;

– точність: Тренувальна точність досягає ~92%, тоді як валідаційна стабілізується на ~70%;

– Balanced Accuracy: Модель демонструє збалансовану точність на рівні ~73%, що підтверджує її здатність працювати з різними класами;

– ROC-AUC: Показники ROC-AUC перевищують 0.91 до кінця навчання, що свідчить про високу якість класифікації навіть на складних прикладах.

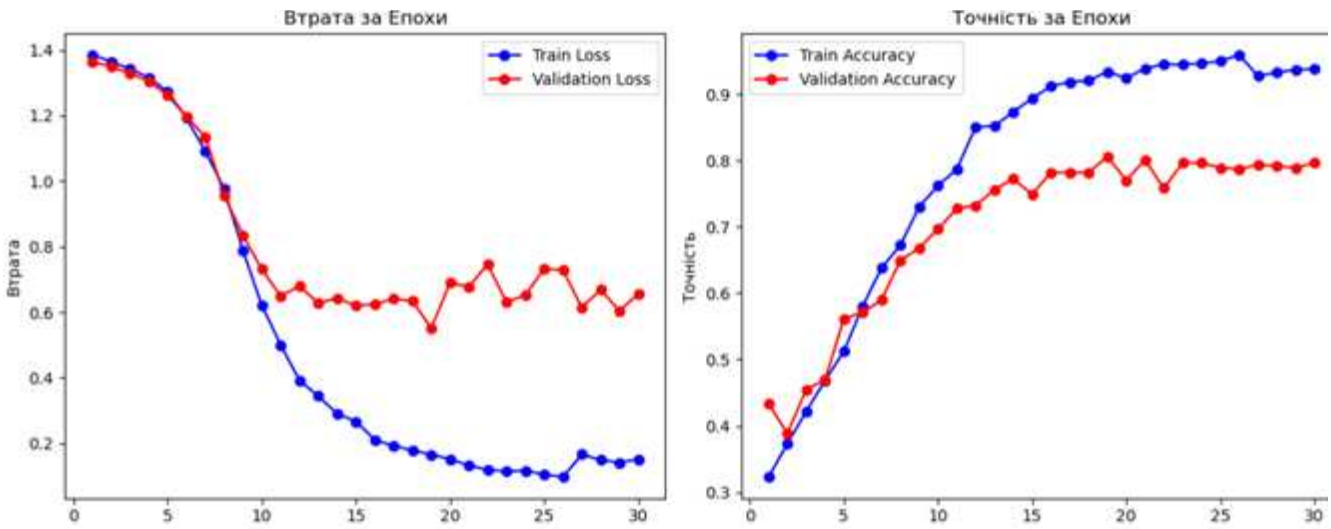


Рисунок 2.7 – Графік тренування моделі на 3 підмножині

У третій підмножині спостерігається один із найкращих результатів серед усіх підмножин, особливо щодо збалансованої точності (Balanced Accuracy).

– втрати: Тренувальні втрати зменшуються до мінімуму до 30-ї епохи, а валідаційні втрати стабілізуються після 12-15 епох, вказуючи на успішне навчання;

– точність: Валідаційна точність стабілізується на рівні ~73%, тренувальна – понад 91%;

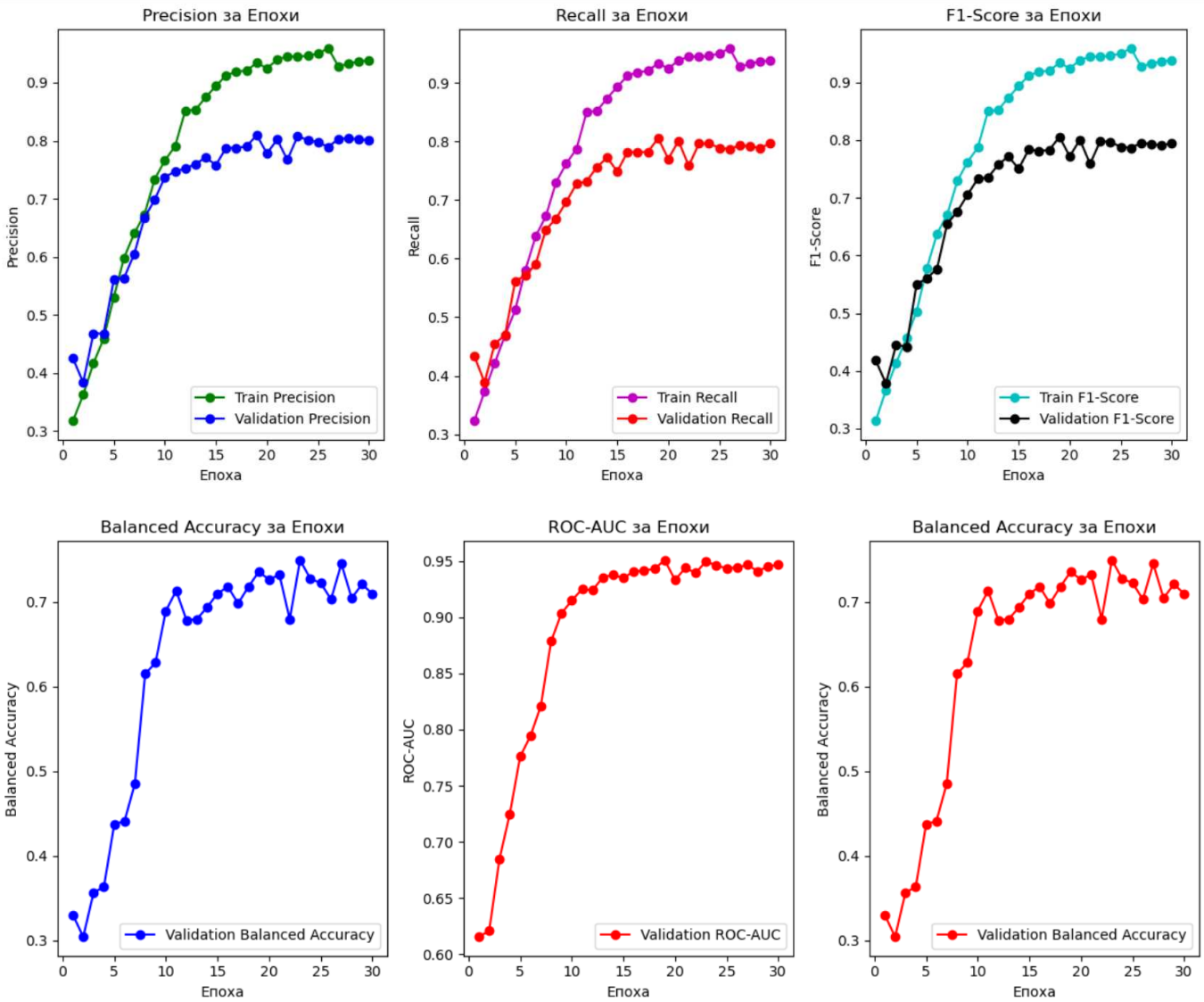


Рисунок 2.8 – Графіки метрик оцінювання моделі на 3 підмножині

– F1-Score: Валідаційний F1-Score досягає ~0.73, підтверджуючи, що модель ефективно збалансовує Precision і Recall;

– ROC-AUC: Значення ROC-AUC перевищує 0.92, що підтверджує відмінну здатність моделі відрізняти патологічні стани від нормальних.

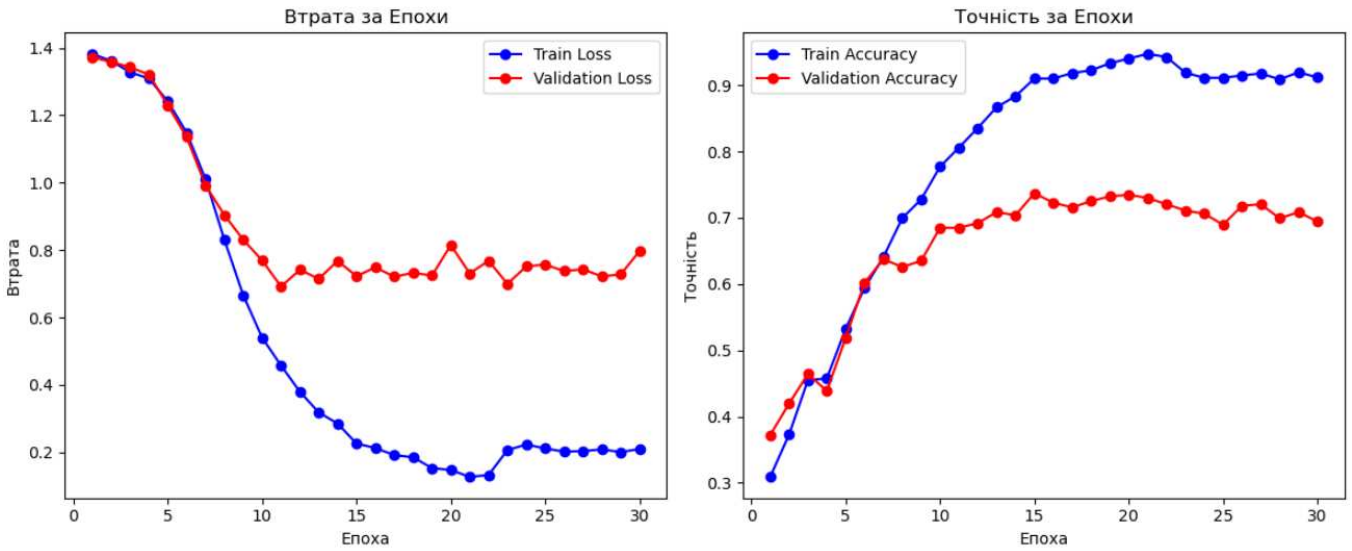


Рисунок 2.9 – Графік тренування моделі на 4 підмножині

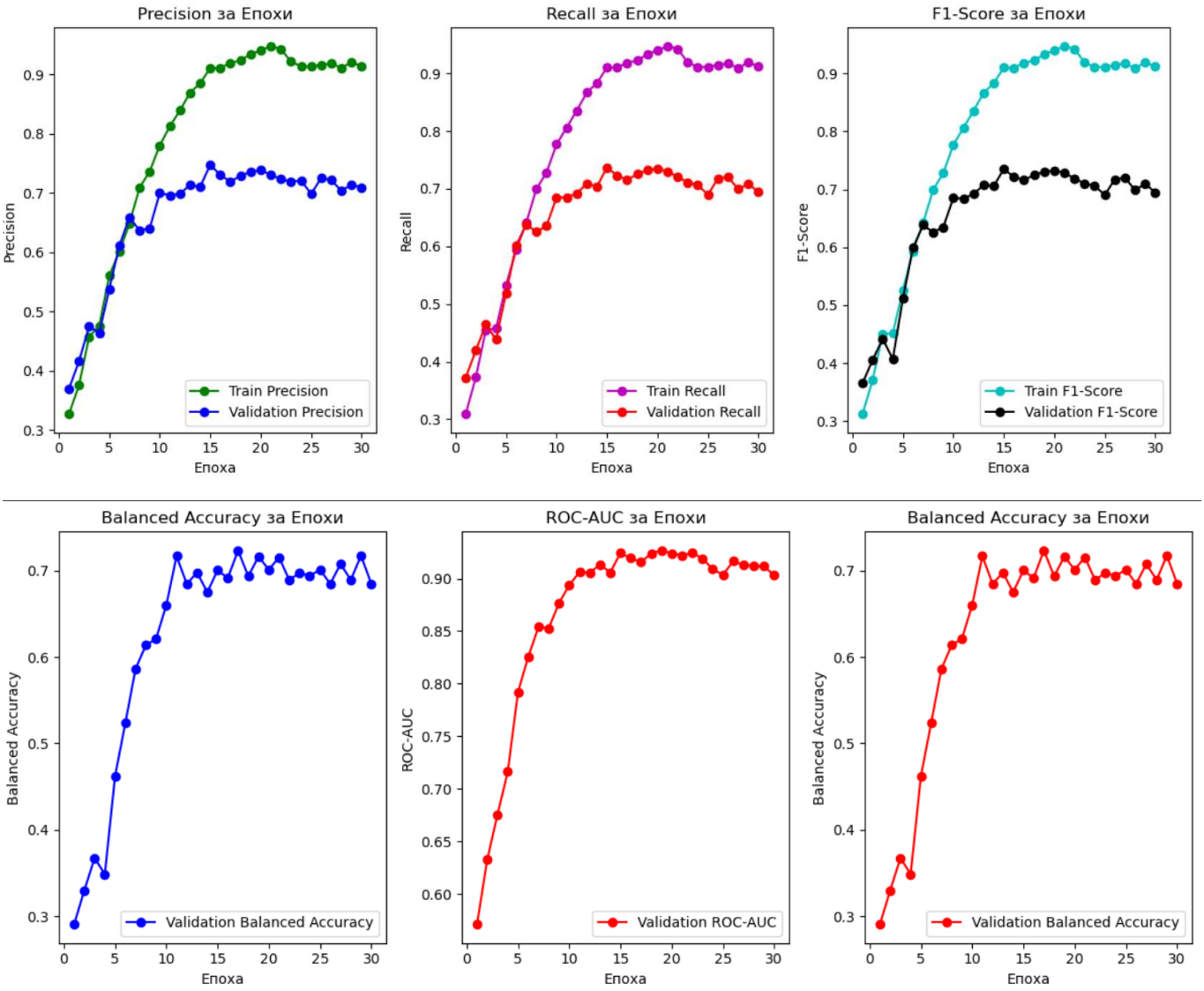


Рисунок 2.10 – Графіки метрик оцінювання моделі на 4 підмножині

У четвертій підмножині модель демонструє стабільні результати, хоча валідаційні втрати трохи вищі, ніж на інших фолдах.

– втрати: Тренувальні втрати стабільно зменшуються, тоді як валідаційні втрати показують легкі коливання після 15-ї епохи, що може свідчити про невеликі складності у валідаційній вибірці;

– точність: Тренувальна точність досягає ~92%, а валідаційна стабілізується на рівні ~72%;

– Balanced Accuracy: Модель демонструє результати на рівні ~72%, зберігаючи стабільність;

– ROC-AUC: Значення ROC-AUC стабілізується на рівні ~0.90, що вказує на гарну якість класифікації.

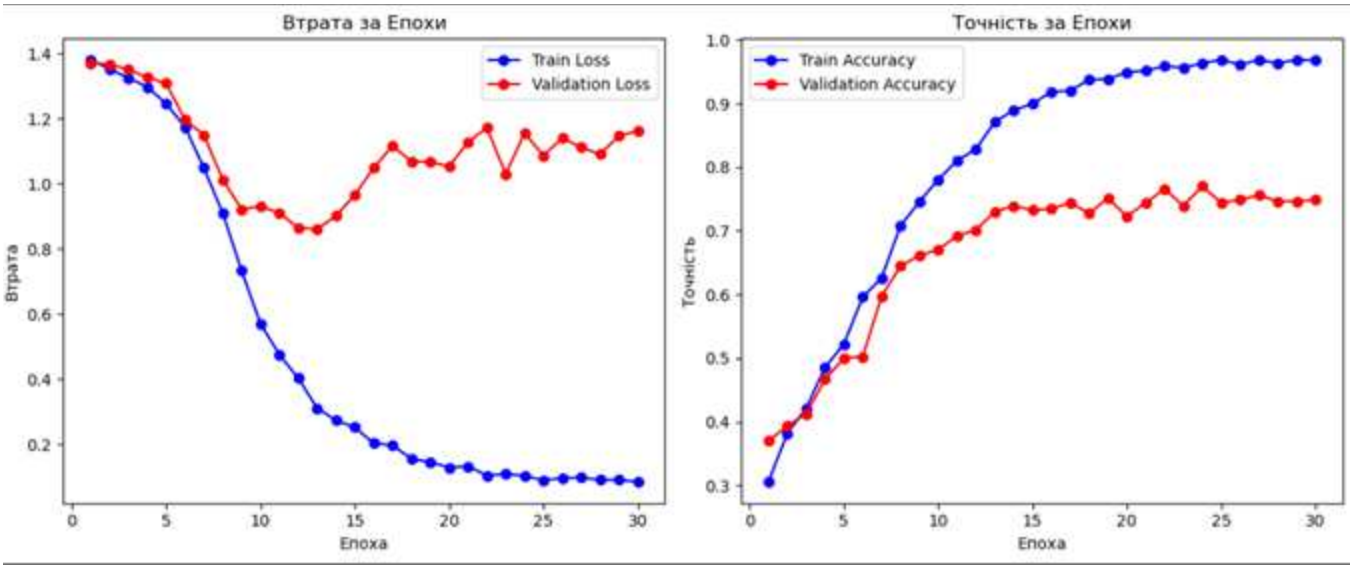


Рисунок 2.11 – Графік тренування моделі на 5 підмножині

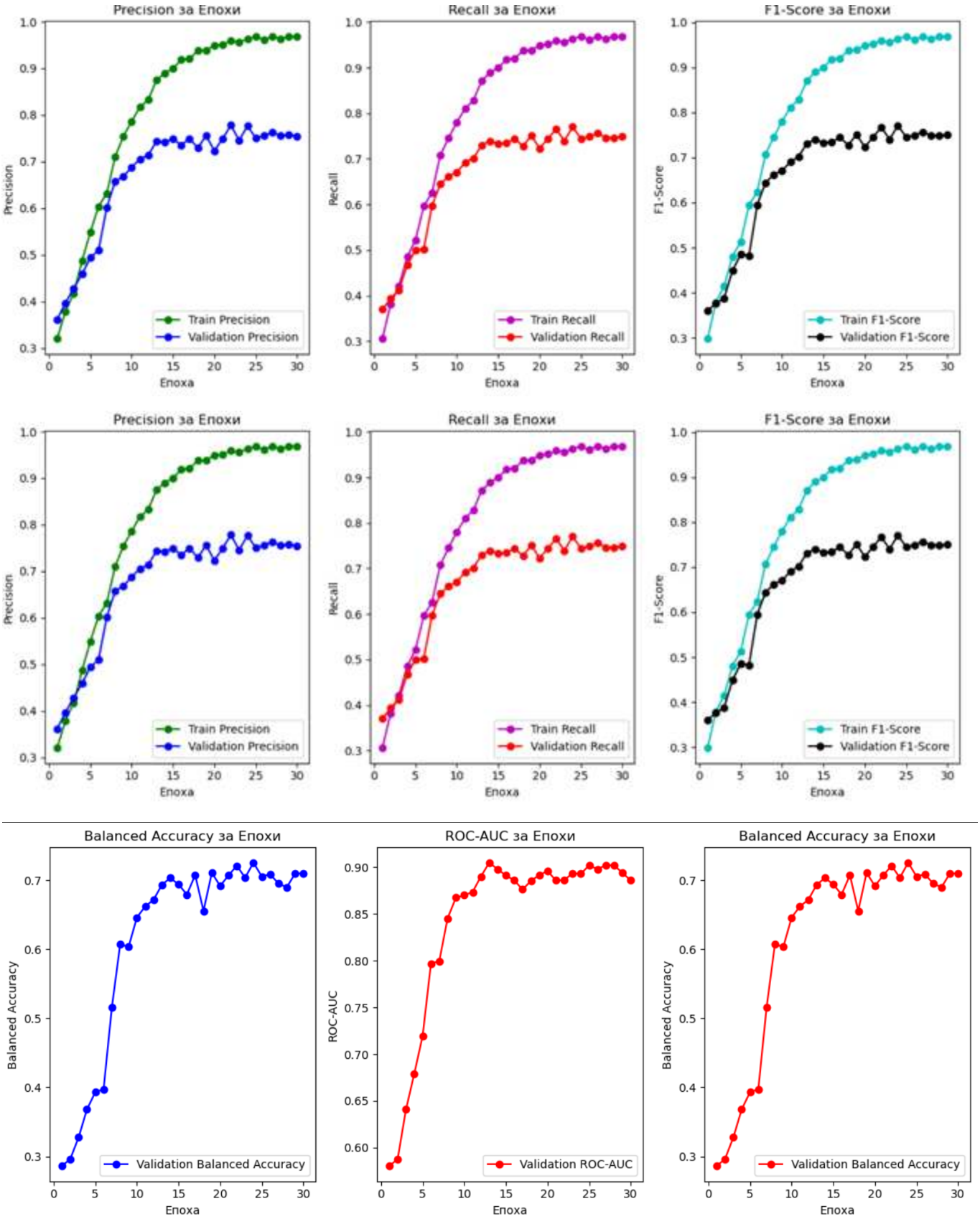


Рисунок 2.12 – Графіки метрик оцінювання моделі на 5 підмножині

П’ята підмножина демонструє стабільний процес навчання з покращенням показників протягом усіх 30 епох.

– втрати: Зменшення тренувальних втрат свідчить про стабільний процес оптимізації. Валідаційні втрати стабілізуються після 10-ї епохи;

– точність: Валідаційна точність досягає ~71%, тренувальна – ~91%;

– Precision і Recall: Ці метрики стабільно покращуються, демонструючи хорошу здатність моделі знаходити патології;

– ROC-AUC: Значення перевищують 0.91, що підтверджує високу якість роботи моделі.

На всіх підмножинах модель демонструє стабільний прогрес у навчанні з кожною епохою, що підтверджується зменшенням втрат і покращенням метрик. Найкращі результати досягаються на третьому фолді, де спостерігається найвища збалансована точність і ROC-AUC. Незначні коливання валідаційних втрат свідчать про можливість вдосконалення моделі через додаткові техніки, такі як регуляризація або збалансування вибірки.

Для реалізації процесу навчання ЗНМ було використано мову програмування Python та фреймворки TensorFlow і Keras. Ці інструменти дали змогу будувати і тренувати згорткові нейронні мережі, а також інтегрувати додаткові методи інтерпретації результатів, такі як Grad-CAM, для візуалізації патологічних зон.

Таким чином, навчання ЗНМ для виявлення патологій на МРТ-зображеннях серця охоплювало підготовку даних, проєктування архітектури, налаштування й оптимізацію моделі, її оцінювання на тестових даних та застосування в умовах реальних досліджень. Це забезпечило високий рівень точності та надійності результатів, що є важливим чинником у медичній практиці.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи проведено дослідження методів виявлення області патологій на зображеннях МРТ. Розроблено ієрархічну структуру методу, що включає етапи завантаження та підготовки зображень, сегментації

області серця за допомогою U-Net, класифікації патологій ResNet50 та інтерпретації результатів з застосуванням Grad-CAM і SHAP. Запропоновано алгоритм дій на кожному етапі, а також детально описано архітектуру нейронних мереж, які використовуються.

Крім того, було обґрунтовано вибір саме таких методів для аналізу МРТ-зображень серця. Метод U-Net забезпечує точне виділення анатомічних структур, а ResNet50 – ефективно виявляє патології, враховуючи особливості структури серця на зображеннях. Інтеграція Grad-CAM і SHAP дає змогу лікарям отримати пояснення рішень моделі, сприяючи її прозорості та надійності. Наведено повний опис процесу підготовки даних, навчання та тестування, що враховує важливість правильного підбору гіперпараметрів та налаштування процесу навчання.

Сформовано ґрунтовну базу для подальшої програмної реалізації створеного методу інтерпретування та проведена перевірка працездатності запропонованих методів на тестових даних. Спроєктовано метод інтерпретації результатів виявлення патологій серця на зображеннях МРТ. Загалом, другий розділ закладає міцний фундамент для створення вебзастосунку за поданим методом інтерпретування з високим ступенем точності та надійності.

РОЗДІЛ 3. Програмна реалізація методу інтерпретування результатів виявлення патологій серця за зображенням МРТ у вигляді вебзастосунку

3.1 Функціональна структура та функціональні процеси програмної реалізації

При розробці автоматизованої системи інтерпретації результатів виявлення патологій серця на основі МРТ-зображень із використанням згорткових нейронних мереж (ЗНМ) необхідно автоматизувати кілька ключових функціональних процесів. Ці процеси мають забезпечити не лише виявлення патологій, але й надання пояснень результатів, щоб лікарі могли розуміти, на основі яких даних і ознак модель приймає рішення.

Основні функціональні процеси такої системи включають:

– функціональний процес «Робота з введенням зображення»;

– функціональний процес «Підготовка зображення»;

– функціональний процес «Вибір області серця для інтерпретації»;

– функціональний процес «Аналіз патологій за допомогою ЗНМ»;

– функціональний процес «Інтерпретація результатів»;

– функціональний процес «Виведення результатів»;

– функціональний процес «Збереження результатів».

Функціональний процес «Робота з введенням зображення»:

Цей процес відповідає за початкове введення МРТ-зображення для аналізу та інтерпретації. Його функції включають (рисунок 3.1):

– отримання вхідних даних від користувача (зображення МРТ);

– перевірка формату та якості зображення для забезпечення відповідності вимогам системи;

– завантаження зображення у систему;

– відображення попереднього перегляду зображення;

– повідомлення користувача про успішне завантаження або виявлені проблеми.

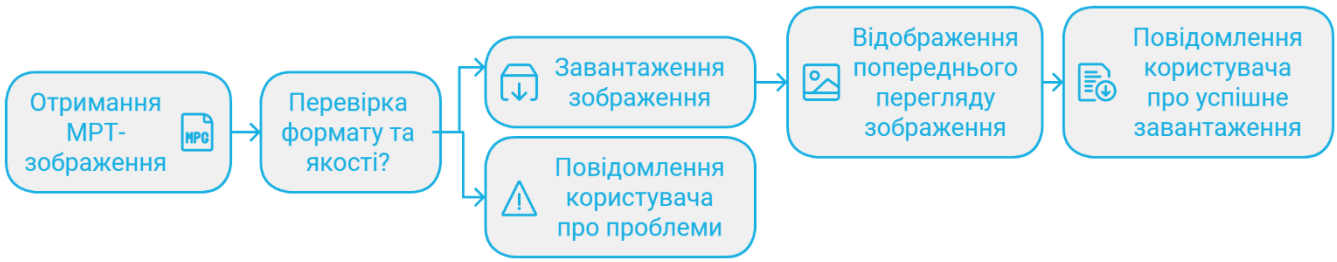


Рисунок 3.1 – Діаграма дій функціонального процесу «Робота з введенням зображення»

Функціональний процес «Підготовка зображення»:

Цей етап відповідає за підготовку зображення для подальшої інтерпретації та аналізу за допомогою ЗНМ. Він включає наступні дії (рисунок 3.2):

– фільтрація зображення для зменшення шумів і покращення якості;

– конвертація зображення до необхідного формату для оброблення моделлю;

– масштабування зображення до необхідного розміру, щоб воно відповідало вимогам ЗНМ;

– відображення попереднього перегляду підготовленого зображення;

– повідомлення користувача про успішну підготовку зображення або виявлені проблеми.

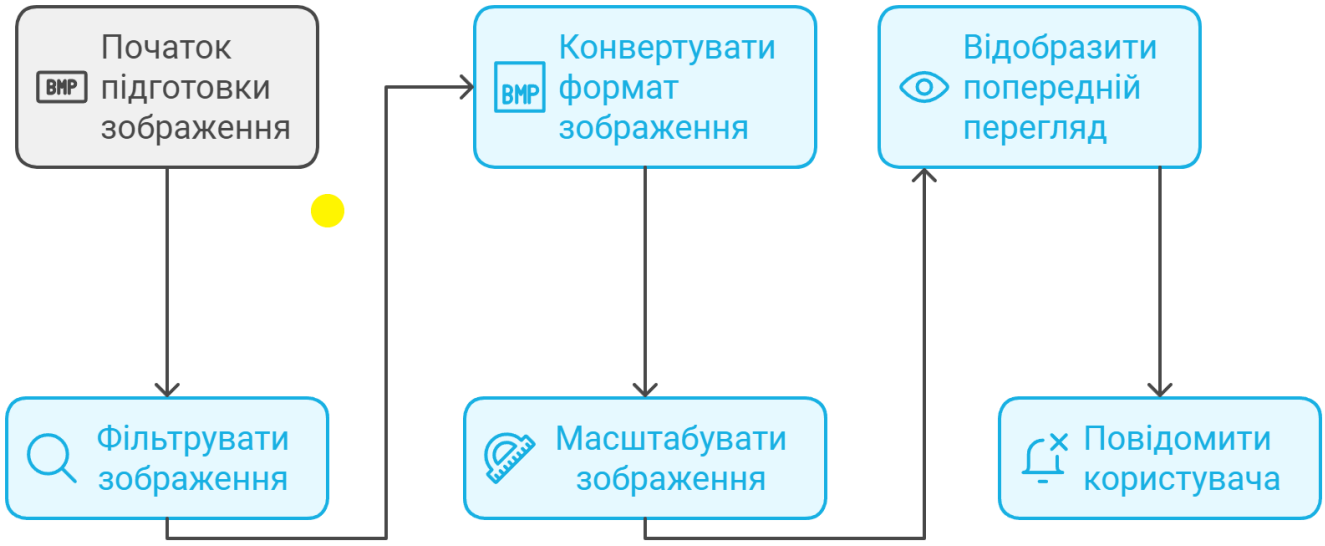


Рисунок 3.2 – «Алгоритм підготовки зображення з основними етапами: фільтрація, конвертація, масштабування та попередній перегляд»

Функціональний процес «Вибір області серця для інтерпретації»: Цей етап передбачає вибір області серця, яку потрібно аналізувати. Для цього можливі ручний або автоматичний вибір області (рисунок 3.3). Основні дії цього процесу:

– навігація по зображенню для вибору потрібної області серця;

– використання алгоритму автоматичного визначення області серця або можливість ручного виділення;

– відображення попереднього перегляду обраної області;

– повідомлення користувача про успішний вибір або проблеми з виділенням.

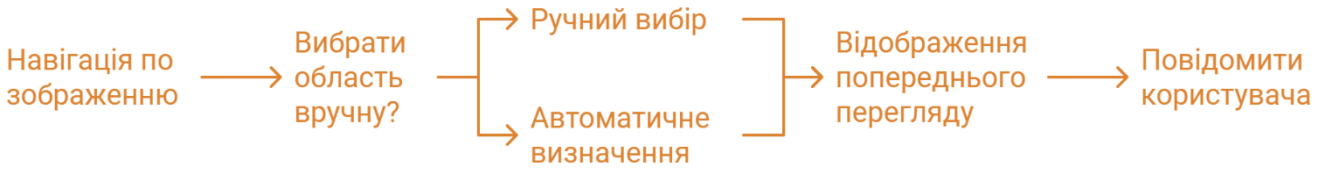


Рисунок 3.3 – Діаграма дій функціонального процесу «Вибір області серця для інтерпретації»

Функціональний процес «Аналіз патологій за допомогою ЗНМ»: На цьому етапі система застосовує навчений ЗНМ для аналізу зображення та виявлення можливих патологій у вибраній області серця (рисунок 3.4). Процес включає:

– використання алгоритмів нейронної мережі для аналізу зображення та виявлення патологій;

– відображення попереднього перегляду результатів аналізу;

– можливість користувачу переглянути та редагувати результати, якщо це необхідно;

– повідомлення користувача про успішне завершення аналізу або виявлені проблеми.

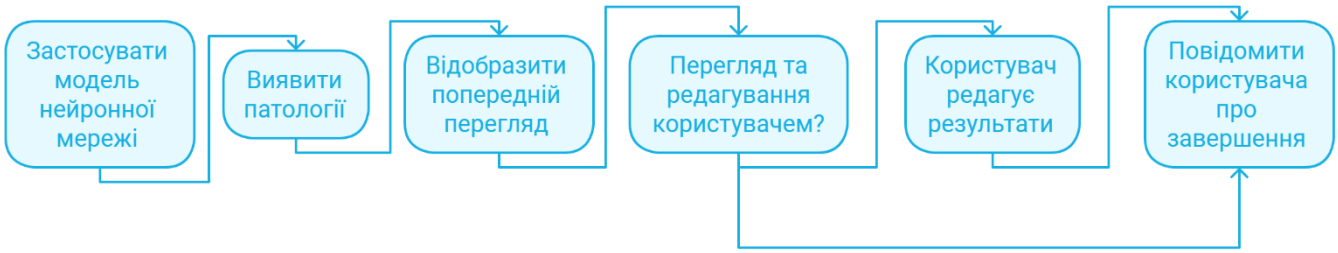


Рисунок 3.4 – Діаграма дій функціонального процесу «Аналіз патологій за допомогою ЗНМ»

Функціональний процес «Інтерпретація результатів»:

Цей етап є ключовим для пояснення результатів, які були отримані в процесі аналізу. Він надає візуальні або текстові пояснення, які дають можливість зрозуміти, чому модель визнала певну область патологічною (рисунок 3.5). Функції включають:

– автоматичне створення теплових карт (наприклад, за допомогою Grad-CAM) для візуалізації, які області вплинули на рішення моделі;

– генерація текстових пояснень, що пояснюють, на яких особливостях модель базувала своє рішення;

– відображення попереднього перегляду візуальних та текстових пояснень;

– повідомлення користувача про успішне завершення інтерпретації або про проблеми з нею.

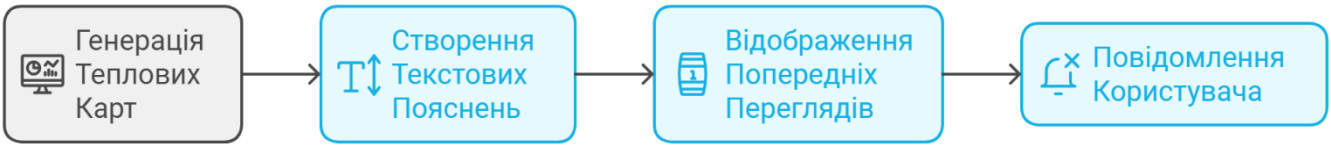


Рисунок 3.5 – Діаграма дій функціонального процесу «Інтерпретація результатів»

Функціональний процес «Виведення результатів»:

Цей процес відповідає за подання результатів аналізу та інтерпретації користувачеві (рисунок 3.6). Його функції:

– відображення результатів аналізу з розподілом на сегменти серця та поясненнями;

– можливість навігації по різним сегментам зображення та перегляд візуальних пояснень (теплових карт);

– виведення статистики по результатам аналізу (наприклад, вказання зон з високим ризиком);

– надання користувачу можливості зберегти результати для подальшого використання або дослідження.

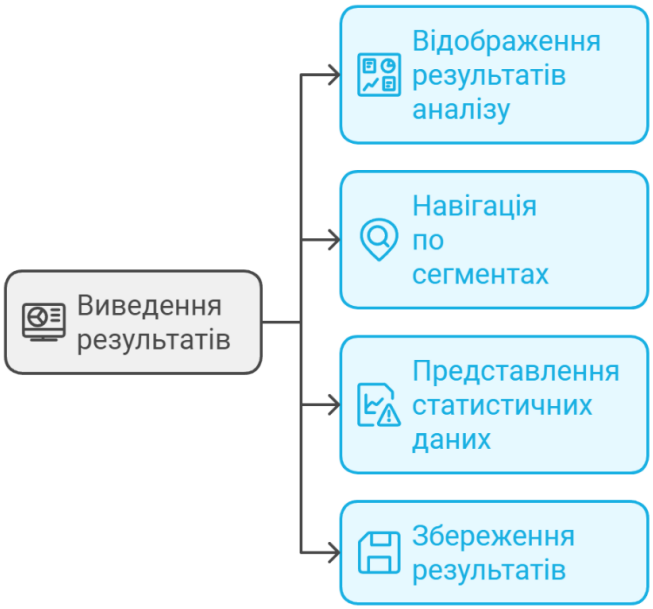


Рисунок 3.6 – Діаграма дій функціонального процесу «Аналіз патологій за допомогою ЗНМ»

Функціональний процес «Збереження результатів»:

Цей процес забезпечує можливість збереження результатів аналізу та інтерпретації (рисунок 3.7). Його можливості включають:

– вибір формату для збереження результатів (наприклад, DICOM або PNG);

– збереження результатів у вибраному форматі та місці на диску;

– запис інформації про аналіз у базу даних для подальшого доступу;

– повідомлення користувача про успішне збереження результатів або про проблеми збереження.

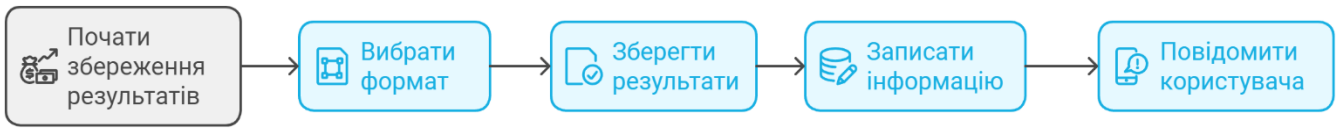


Рисунок 3.7 – Діаграма дій функціонального процесу «Збереження результатів»

Усі зазначені процеси утворюють функціональну структуру автоматизованої системи, яка забезпечує не тільки аналіз і виявлення патологій, але й надання медичним фахівцям пояснень, що підвищують прозорість прийняття рішень на основі МРТ-зображень. Це дає змогу створити більш довірливу та надійну систему підтримки прийняття рішень у медицині.

3.2 Структура та функціональне призначення складових вебзастосунку

Програмна реалізація методу призначена для автоматизованої інтерпретації результатів аналізу патологій серця на основі зображень МРТ. Система використовує модель ResNet50, а також методи Grad-CAM і SHAP, що забезпечують прозору інтерпретацію результатів. Для інтерактивної взаємодії з користувачем застосовано платформу Gradio. Архітектура системи відображена на рисунках 3.8 і 3.9, які демонструють послідовність етапів аналізу: від початкової оброблення зображення до кінцевого збереження результатів.

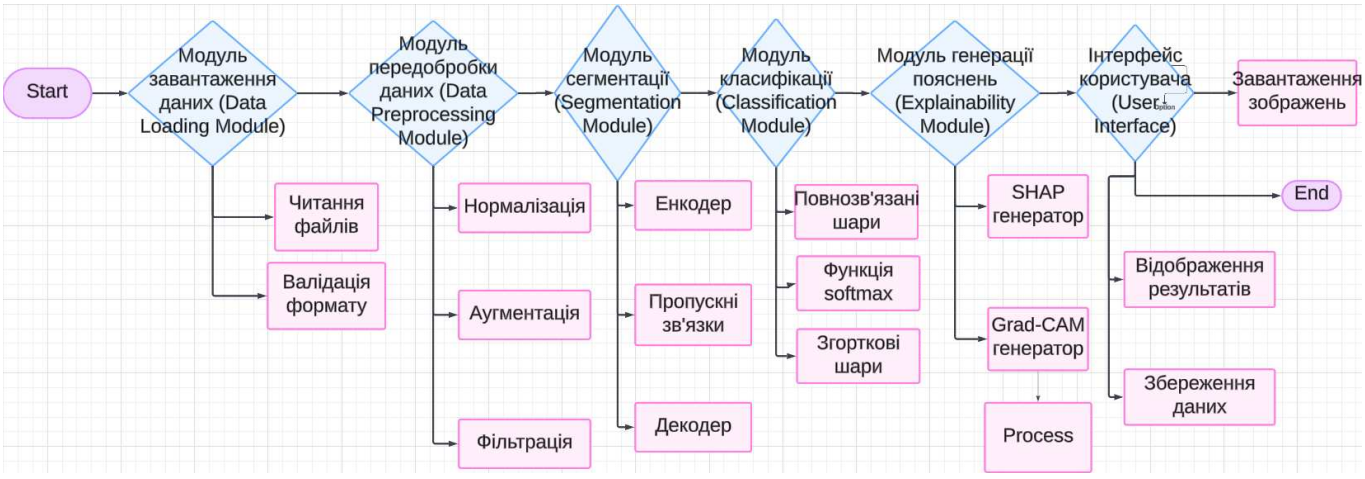


Рисунок 3.8 – Функціональна структура системи

Стартовим етапом є завантаження вхідного зображення за допомогою веб-інтерфейсу. Користувач може завантажувати файли у форматах JPEG, PNG або інших підтримуваних.

Модуль перевірки формату:

– перевірка зображення на відповідність підтримуваним форматам;

– за невідповідності система інформує користувача про помилку.

Модуль передоброблення даних:

– усунення шумів та артефактів за допомогою фільтрації;

– нормалізація інтенсивності пікселів на основі середнього та стандартного відхилення, використаних під час навчання моделі;

– аугментація даних шляхом випадкових трансформацій (обертання, зміна яскравості та контрастності) для підвищення узагальнювальної здатності моделі.

Сегментація області серця:

– модель U-Net сегментує серце на зображенні, що дає змогу зосередити подальший аналіз лише на релевантних ділянках;

– сегментоване зображення надсилається для подальшої класифікації.

Модуль класифікації патології:

– модель ResNet50 класифікує сегментоване зображення за трьома станами: Normal (нормальний), Hypertrophy (гіпертрофія міокарда), Infarction (інфаркт міокарда);

– функція Softmax генерує вектор ймовірностей для кожного класу, що дає змогу визначити найімовірніший стан;

– результати класифікації містять тип патології та ймовірність передбачення.

Генерація пояснень:

Grad-CAM:

– використовує градієнти, обчислені ResNet50, для створення теплової карти, що показує впливові зони зображення;

– карта візуалізує ключові області зображення, що вплинули на рішення моделі;

– отримана теплова карта накладається на початкове зображення для покращення сприйняття.

SHAP:

– оцінює вплив кожного пікселя або регіону зображення на передбачення моделі;

– формує текстові пояснення, що деталізують внесок кожної частини зображення в остаточний результат.

Інтерпретація результатів:

– користувач отримує зображення з накладеними тепловими картами Grad-CAM і SHAP;

– формується текстове пояснення, яке описує виявлену патологію (наприклад, «Виявлено гіпертрофію серцевого м’яза»);

– ці результати дають можливість лікарю або користувачеві отримати деталізовану інформацію для прийняття рішень.

Збереження результатів:

– усі результати зберігаються у базі даних або у вигляді окремих файлів.

Збережені дані можуть містити:

– початкове зображення;

– накладені Grad-CAM і SHAP теплові карти;

– текстові пояснення результатів класифікації.

На рисунку 3.9 наведена деталізована архітектура системи.

Завантаження зображення:

– користувач через веб-інтерфейс Gradio завантажує МРТ-зображення;

– інтерфейс забезпечує можливість перегляду та перевірки файлів перед обробкою.

Передобробка зображення:

– фільтрація шумів: Виконується первинна обробка для видалення небажаних артефактів;

– нормалізація інтенсивності: Приведення піксельних значень до стандартного діапазону;

– аугментація: Для покращення узагальнюючої здатності моделі додаються зміни, як-от зміна масштабу та обертання.

Рисунок 3.9 – Деталізація функціональних етапів

Сегментація області серця:

– застосовується U-Net для виділення серцевих областей на зображенні. Це дає змогу уникнути аналізу зайвих областей зображення.

Класифікація патології:

– ResNet50: На основі сегментованих даних визначає ймовірність трьох станів серця;

– результати класифікації представлені у вигляді текстового опису, що включає ймовірність для кожного стану.

Генерація пояснень:

Grad-CAM:

– відображає впливові області зображення через теплову карту.

SHAP:

– аналізує внесок кожної зони зображення;

– доповнює Grad-CAM, надаючи текстові пояснення.

Відображення результатів:

– користувач отримує текстове пояснення, результати Grad-CAM і SHAP, що дає змогу детально проаналізувати рішення моделі.

Збереження даних:

– всі отримані результати можуть бути збережені для подальшого аналізу. Це включає вихідні зображення, теплові карти та текстові пояснення.

Таким чином, розроблена система поєднує всі необхідні етапи аналізу МРТ-зображень серця – від завантаження даних та їх первинної оброблення до формування інтерпретованих результатів і збереження даних. Такий підхід забезпечує ефективний аналіз патологій, підвищуючи точність, прозорість та зручність використання у практичній медицині.

3.3 Особливості реалізації програмних складових вебзастосунку

Розроблений вебзастосунок мусить слугувати для виявлення патологій серця за МРТ-зображеннями та базується на застосуванні моделі ResNet50, а також методів Grad-CAM і SHAP для прозорої інтерпретації результатів. Система підтримує етапи завантаження зображень, їх попередньої оброблення, класифікації патологій та подання результатів із поясненнями.

На рисунку 3.10 представлено класову структуру системи. Зокрема на рисунку 3.11 та 3.12 зображено послідовність роботи системи з оброблення зображень та архітектуру взаємодії серверної та клієнтської частин. Основні компоненти рисунка 3.10 включають:

Методи:

– display_results(results): Виводить результати класифікації та пояснень (Grad-CAM, SHAP);

– save_results(file_path, results): Зберігає результати у файли;

– взаємодія: отримує дані від DataLoader, ExplainabilityModule і ClassificationModel.

DataLoader (Завантажувач даних):

– завантажує та валідує формат зображення.

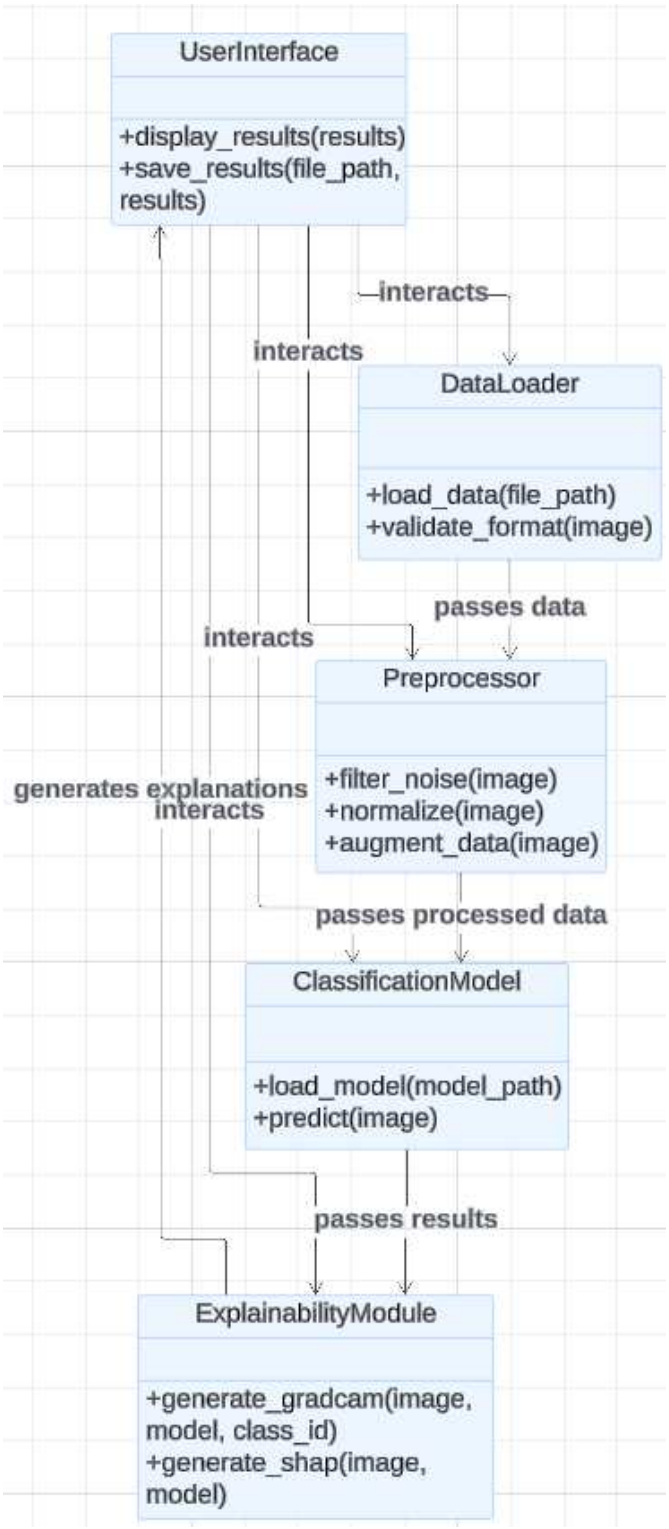


Рисунок 3.10 – Діаграма класів аналізу та пояснення медичних зображень

UserInterface (Інтерфейс користувача):

– відповідає за взаємодію з користувачем.

Методи:

– load_data(file_path): завантажує зображення для аналізу;

– validate_format(image): перевіряє формат зображення;

– передає зображення модулю Preprocessor для подальшої оброблення.

Preprocessor (Попередня обробка):

– виконує попередню обробку зображень.

Методи:

– filter_noise(image): фільтрує шуми;

– normalize(image): нормалізує інтенсивність;

– augment_data(image): застосовує аугментацію (масштабування, обертання);

– результат: підготовлене зображення передається ClassificationModel.

ClassificationModel (Модель класифікації):

– основний компонент для класифікації зображень.

Методи:

– load_model(model_path): завантажує попередньо навчений ResNet50;

– predict(image): класифікує зображення;

– передає результати модулю ExplainabilityModule.

ExplainabilityModule (Модуль пояснень):

– генерує візуалізації Grad-CAM та пояснення SHAP.

Методи:

– generate_gradcam(image, model, class_id): створює теплову карту важливих областей;

– generate_shap(image, model): оцінює вплив пікселів;

– повертає пояснення в UserInterface.

Ця схема описує порядок виконання операцій і взаємодію між модулями системи під час аналізу зображень МРТ.

Завантаження зображення:

– користувач завантажує зображення через UserInterface;

– зображення передається в DataLoader для перевірки формату.

Перевірка формату:

– DataLoader перевіряє відповідність зображення підтримуваним форматам;

– у разі успішної валідації передає зображення в Preprocessor.

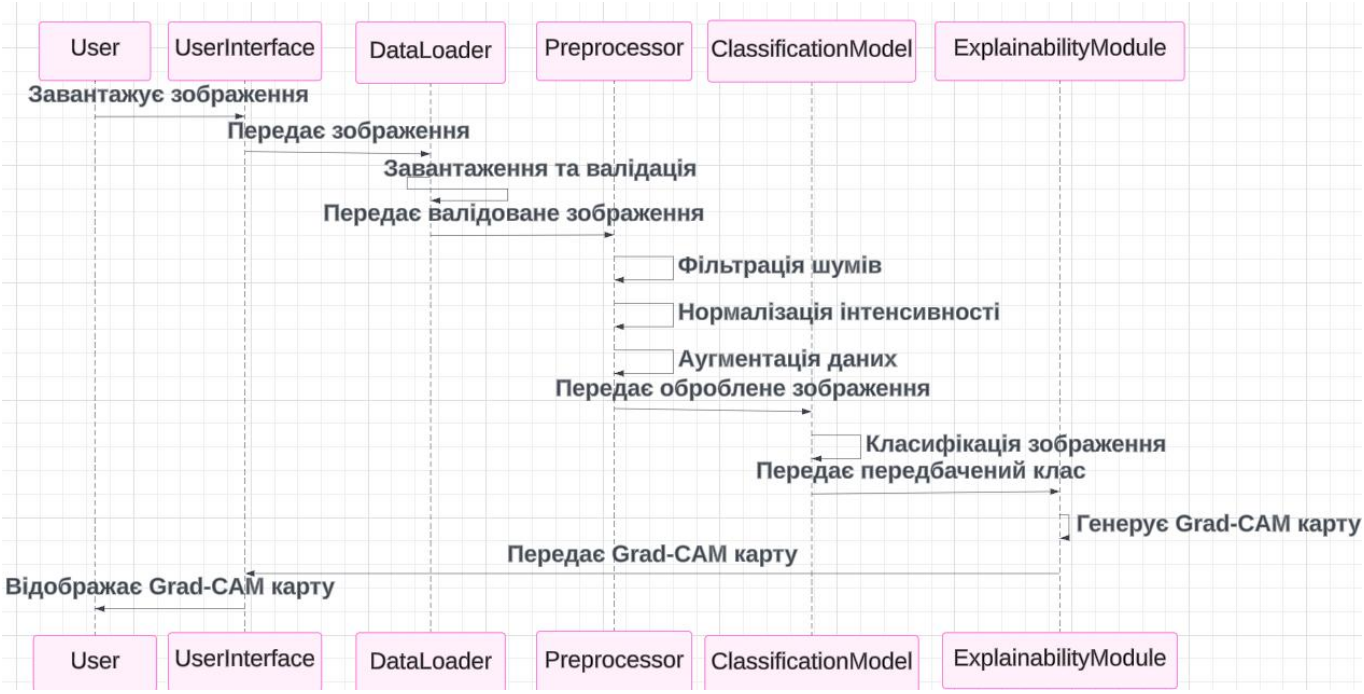


Рисунок 3.11 – Послідовність роботи системи з оброблення зображень, яка демонструє етапи взаємодії між користувачем та усіма комонентами системи

Попередня обробка зображення:

Preprocessor:

– видаляє шуми;

– нормалізує піксельні значення;

– застосовує аугментацію;

– оброблене зображення передається ClassificationModel.

Класифікація зображення:

ClassificationModel:

– використовує ResNet50 для класифікації зображення;

– генерує вектор ймовірностей для кожного класу патології;

– результати класифікації передаються в ExplainabilityModule.

Генерація пояснень:

ExplainabilityModule:

– створює теплову карту Grad-CAM для візуалізації важливих областей;

– генерує SHAP пояснення для текстової інтерпретації впливу пікселів;

– пояснення передаються в UserInterface.

Виведення результатів:

UserInterface:

– виводить результати класифікації, теплову карту Grad-CAM і текстові пояснення SHAP;

– за потреби користувач може зберегти результати у файл.

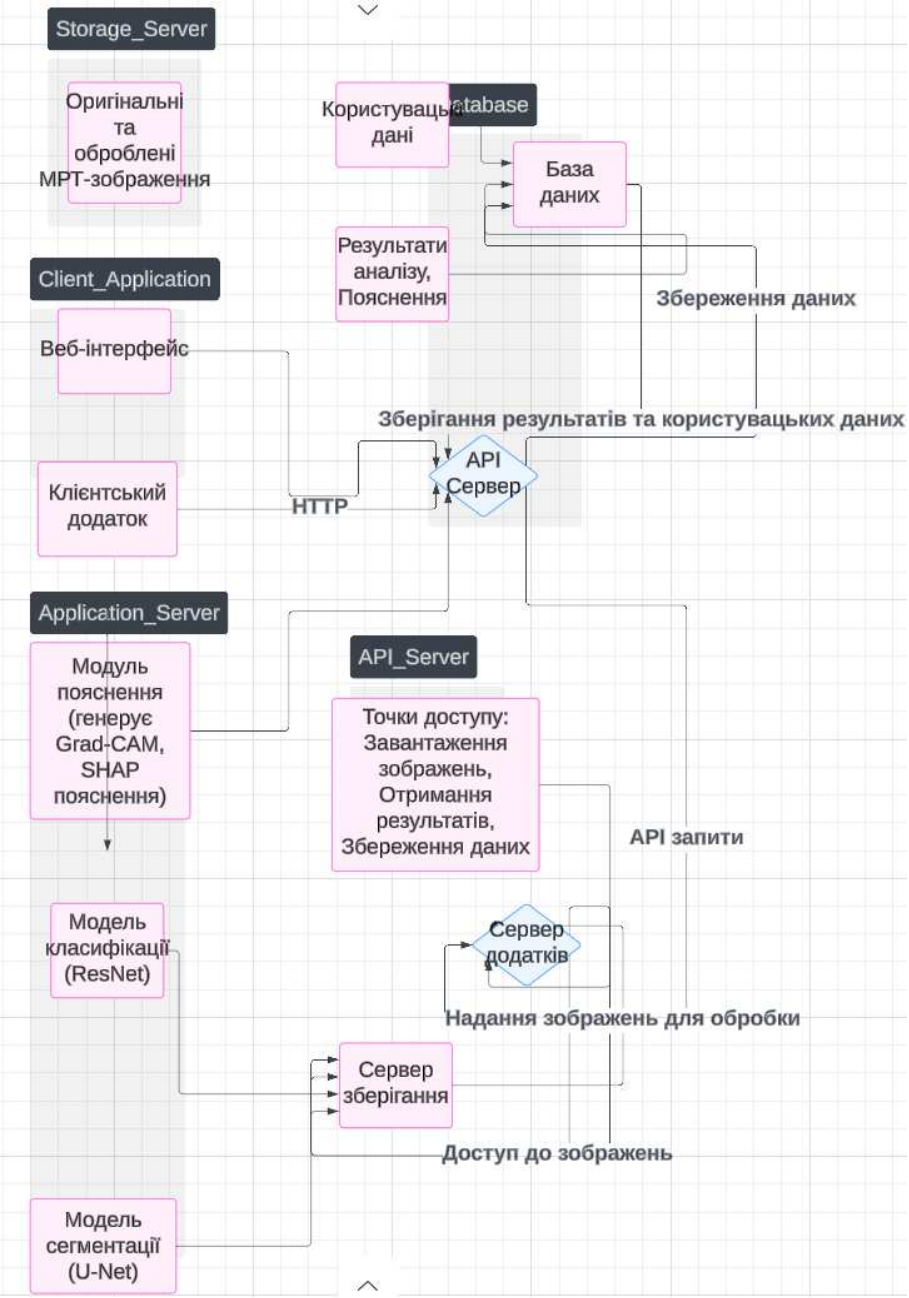


Рисунок 3.12 – Архітектура взаємодії серверної та клієнтської частин, яка відображає процес оброблення МРТ-зображень:

Остання схема демонструє серверну архітектуру, яка забезпечує обробку зображень і управління даними.

Storage_Server (Сервер зберігання):

– зберігає оригінальні та оброблені зображення МРТ;

– доступний через API_Server.

Client_Application (Клієнтський додаток):

– надсилає HTTP-запити на сервер для оброблення зображень;

– отримує результати класифікації та пояснень.

API_Server (Сервер доступу до API):

– обробляє запити від клієнтського додатка;

– забезпечує доступ до даних у базі даних і сервері зберігання.

Application_Server (Сервер додатків):

– виконує основну обробку;

– сегментацію (U-Net);

– класифікацію (ResNet50);

– генерацію пояснень (Grad-CAM, SHAP);

– повертає результати через API_Server.

Database (База даних):

– зберігає результати класифікації та пояснення SHAP для подальшого використання.

Висновки до розділу 3

Третій розділ кваліфікаційної роботи присвячений детальному вивченню функціональної структури вебзастосунку для інтерпретування результатів виявлення патологій серця на основі МРТ-зображень за спроєктованим методом у попередньому розділі. Проаналізовано всі етапи взаємодії системи, визначено їхню функціональну роль та окреслено послідовність виконання. Проєктування архітектури цього вебзастосунку стало можливим завдяки вдалому поєднанню

сучасних методів глибокого навчання та надійних технологій розроблення програмного забезпечення.

В рамках розділу було розроблено опис ключових функціональних процесів, які забезпечують автоматизоване введення зображень, їхню підготовку, виділення областей серця, аналіз патологій та їхню інтерпретацію, а також виведення результатів з можливістю їх збереження. Створено діаграми дій для кожного етапу. Розглянуто детально функціональні блоки програмних складових, а саме модулі завантаження, передобрроблення даних, сегментації, класифікації, пояснень, а також їхню взаємодію в межах всієї системи.

Серверна архітектура забезпечує інтеграцію клієнтської частини з обробкою даних, що дає змогу обробляти великі обсяги інформації у реальному часі, зберігати результати аналізу та надавати їх у зручному вигляді для користувача. Інфраструктура включає сервери зберігання, базу даних і модулі оброблення, які взаємодіють через API.

В результаті була сформована ієрархічна структура взаємодії між модулями, яка забезпечує послідовний процес обробки зображень МРТ, виявлення патологій та їхню інтерпретацію. Ця структура гарантує, що кожен етап обробки даних виконується належним чином, та інформація переходить між модулями з мінімальними затримками. Розроблена архітектура створює міцну основу для надійного функціонування вебзастосунку.

РОЗДІЛ 4. Експериментальне тестування програмної реалізації методу інтерпретування результатів виявлення патологій серця за зображенням МРТ

4.1 Дослідження результативності поданого методу інтерпретування

Дослідження результативності поданого методу інтерпретування полягало у визначенні точності методу, оцінці якості класифікації патологій та перевірці зрозумілості рішень за допомогою методів Grad-CAM і SHAP.

Для оцінювання точності виявлення патологій за МРТ-зображеннями використано матриці плутанини на різних етапах навчання моделі (епохах 1, 15 та 30), наведені на рисунках 4.1–4.3.

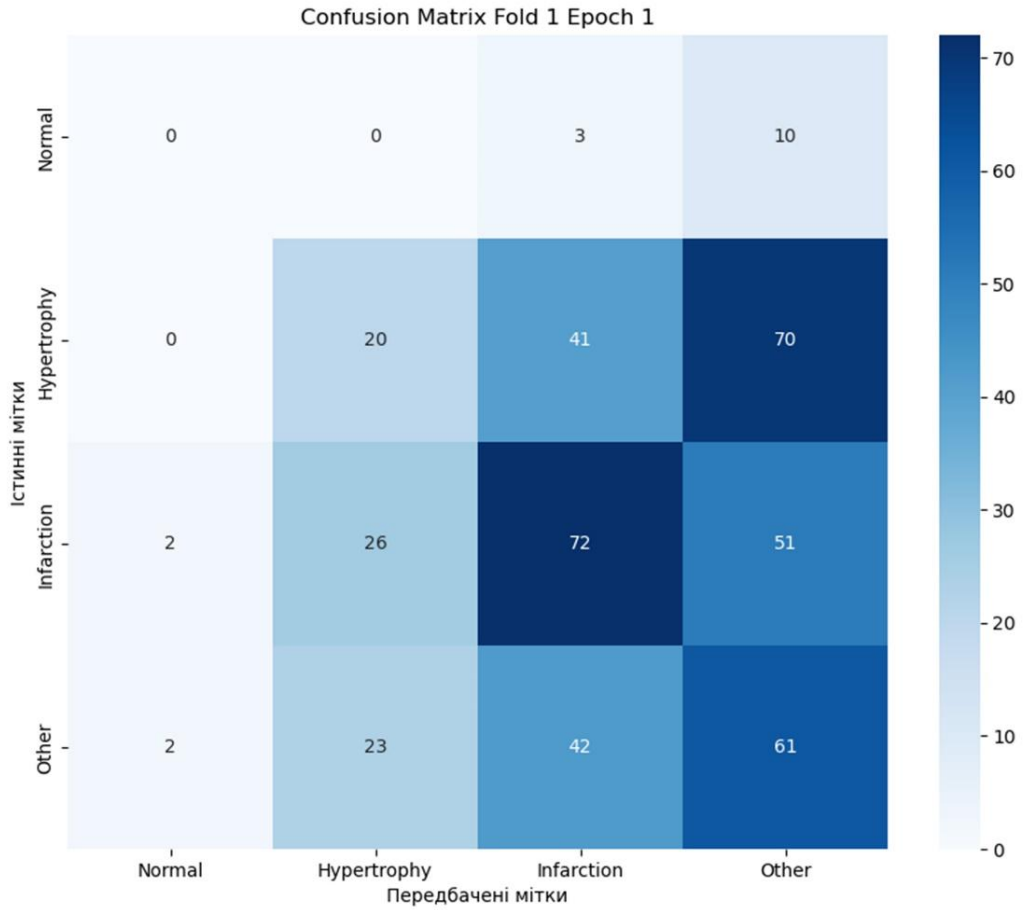


Рисунок 4.1 – Матриця помилок для першої епохи

На початковому етапі (перша епоха) модель демонструвала суттєву кількість помилкових класифікацій, зокрема більшість випадків класу «Hypertrophy» були

помилково віднесені до «Infarction». Також спостерігалася плутанина між класами «Other» та «Infarction».

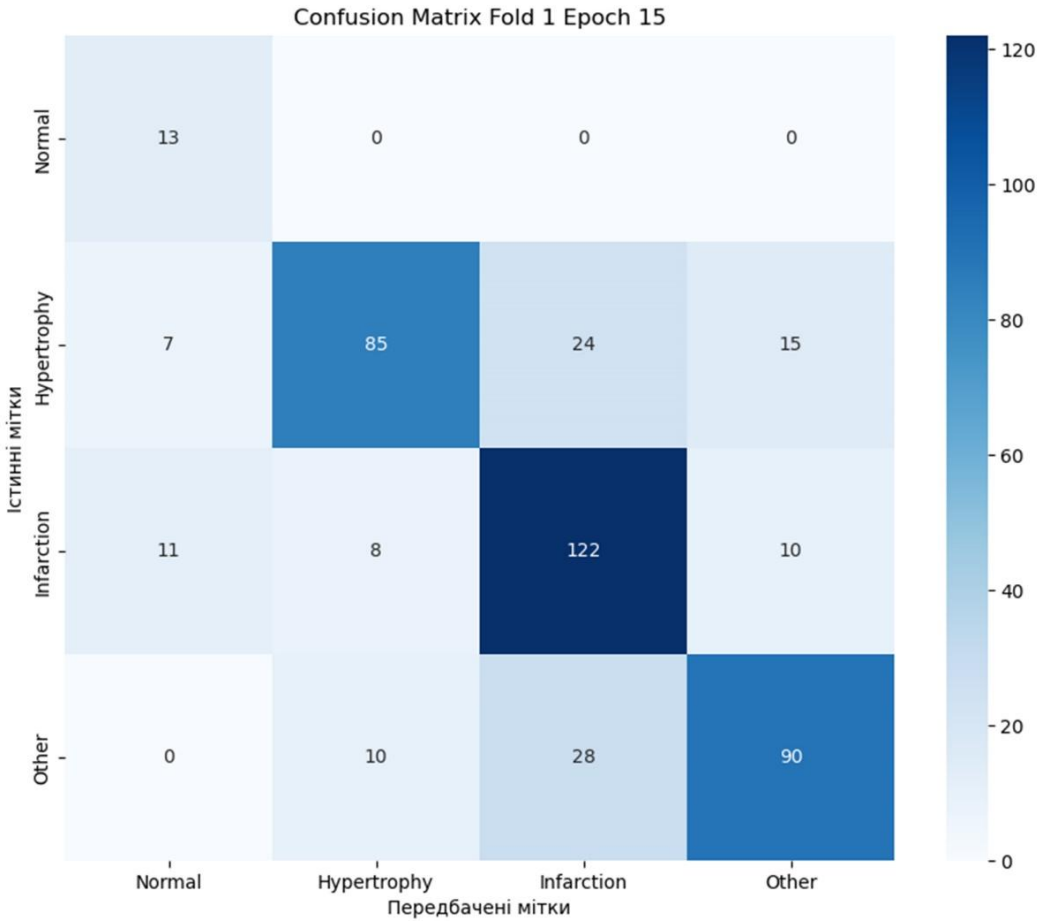


Рисунок 4.2 – Матриця помилок для п’ятнадцятої епохи

На п’ятнадцятій епосі відбулося помітне покращення точності для класів «Normal» і «Infarction». Водночас залишалися певні труднощі у розрізненні класів «Hypertrophy» та «Other».

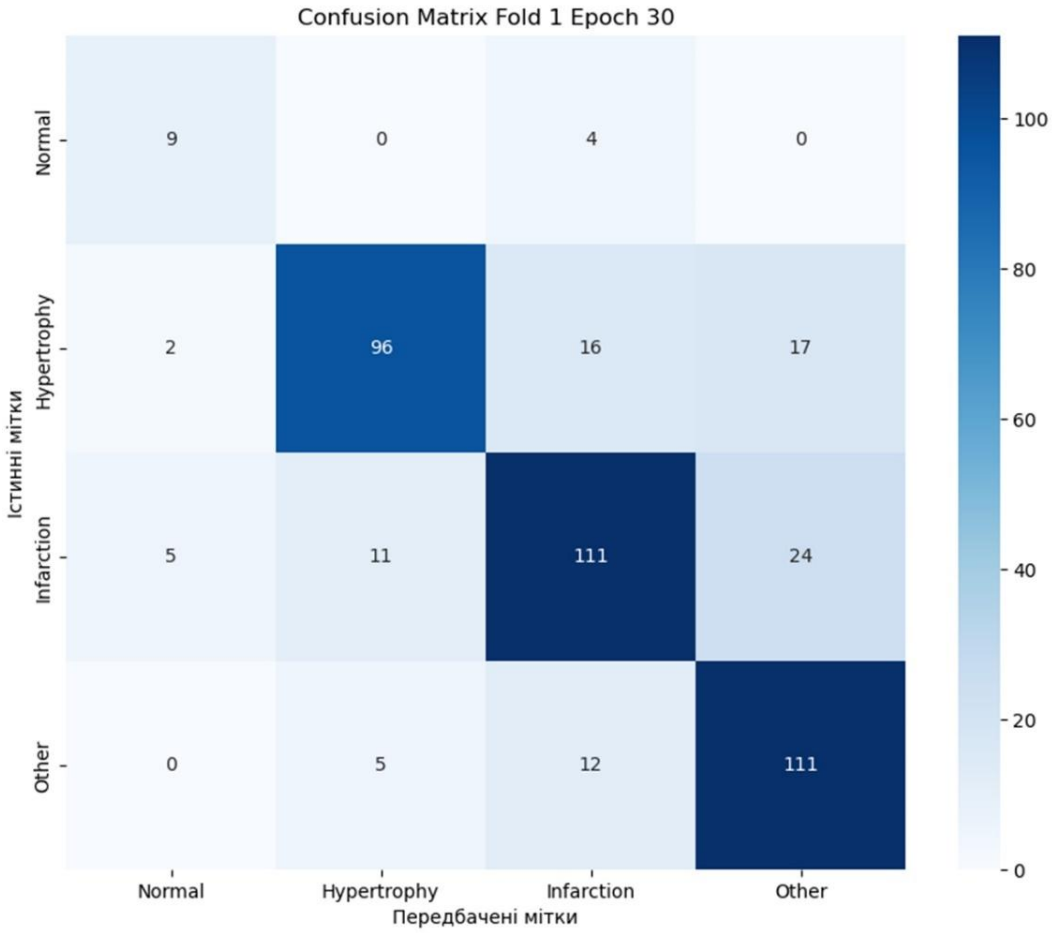


Рисунок 4.3 – Матриця помилок для тридцятої епохи

На тридцятій епосі спостерігалося загальне зменшення кількості помилкових класифікацій для всіх класів. Для класу «Normal» модель досягла майже повної точності, а для «Other» та «Infarction» значно підвищилися показники Recall.

Для оцінювання продуктивності моделі використовувалися такі метрики:

– Precision (Точність): відображає частку правильних позитивних передбачень серед усіх передбачень, які модель класифікувала як позитивні;

– Recall (Повнота): вказує, яку частину всіх реальних позитивних випадків модель правильно визначила;

– F1-Score: гармонійне середнє між Precision та Recall, що демонструє збалансованість моделі;

– Balanced Accuracy: враховує точність для кожного класу, що є важливим у разі дисбалансу між класами;

– ROC-AUC: площа під кривою ROC показує здатність моделі розрізняти патології та здорові стани.

Приклад результатів для першого згину тестування представлено на рисунку 4.4.

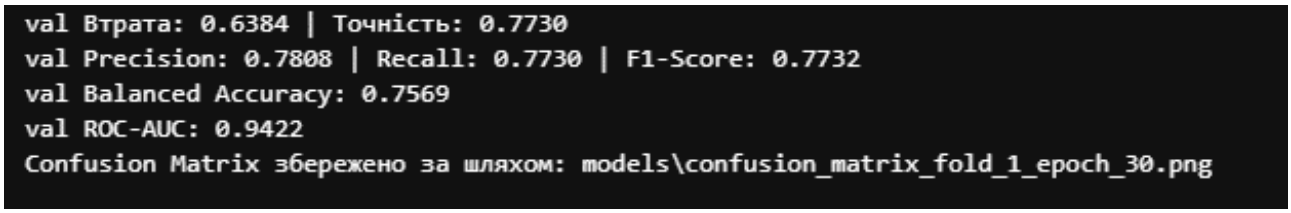


Рисунок 4.4 – Приклад метрик оцінювання

Таблиця 4.1 демонструє значення метрик Precision, Recall, F1-Score та ROC-AUC для кожного класу.

Таблиця 4.1 – Значення метрик для різних класів

Клас	Precision	Recall	F1-Score	ROC-AUC
Normal	0.87	0.84	0.85	0.96
Hypertrophy	0.78	0.75	0.76	0.91
Infarction	0.80	0.74	0.77	0.93
Other	0.77	0.81	0.79	0.92

Аналіз результатів:

– Normal: найвищі показники Precision (87%) і F1-Score (85%), що свідчить про надійну класифікацію нормального стану;

– Hypertrophy: дещо нижчі Precision (78%) і Recall (75%) пов’язані з ускладненим розрізненням цього класу;

– Infarction (Інфаркт): значення Precision і Recall знаходяться на рівні 80% і 74% відповідно, демонструючи добру класифікацію складних випадків;

– Other: високий ROC-AUC (0.92) вказує на ефективність моделі у виявленні менш поширених патологій.

Отже, результати експериментального тестування свідчать про здатність створеного вебзастосунку достовірно сегментувати МРТ зображення та класифікувати патології серця за МРТ-зображеннями. Методи Grad-CAM і SHAP забезпечують можливість зрозуміти, які саме ділянки зображення впливають на рішення моделі, що підвищує довіру до її результатів і сприяє ефективному використанню системи в клінічній практиці.

4.2 Експериментальне тестування вебзастосунку за спроєктованим методом інтерпретування виявлення патологій серця за зображенням МРТ

Запропонований метод забезпечив візуалізацію важливих областей зображень, що найбільше вплинули на рішення моделі. На рисунку 4.2 наведено приклади теплових карт для кожного класу.

Теплові карти: Візуалізація чітко демонструє, які області зображення вплинули на рішення моделі.

– повнота: Grad-CAM показав, що для класу «Normal» теплові карти переважно покривають області, які відповідають здоровим тканинам;

– складні класи: Для «Hypertrophy» і «Infarction» модель продемонструвала чутливість до зони міокарда, що підтверджує її здатність правильно виявляти патології.

На підставі аналізу матриць плутанини та метрик виконано такі оптимізації:

– збалансування втрат: Введено вагові коефіцієнти для кожного класу, що дало змогу зменшити вплив дисбалансу в даних;

– аугментація даних: Використовувалися методи обертання, масштабування та зміни яскравості зображень для покращення узагальнюючої здатності моделі;

– прозорість рішень: Інтеграція Grad-CAM і SHAP дала можливість візуалізувати важливі області зображень і підвищити довіру до системи;

– додаткове тестування: Модель протестували на окремих підмножинах даних, щоб оцінити її стабільність і стійкість до різних типів шуму.

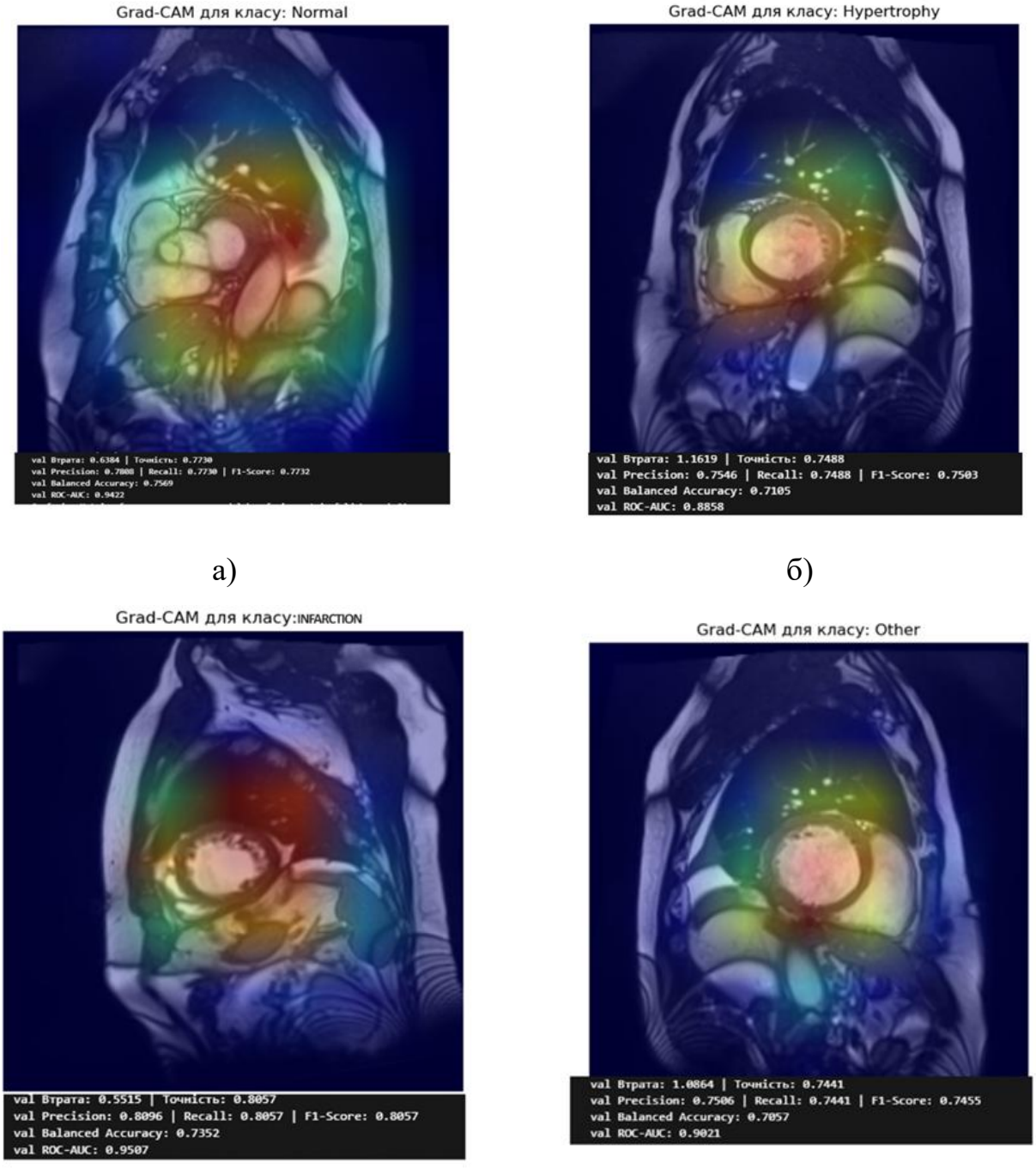


Рисунок 4.5 – Візуалізація вдосконаленого методу за тепловими мапами на основі Grad-CAM для класу: а) «Normal», «Hypertrophy», «Infarction» та «Other»

Grad-CAM продемонстрував здатність моделі до правильного визначення релевантних зон зображень (таблиця 4.2):

– для класу «Normal» теплові карти покривають лише здорові тканини;

– для «Hypertrophy» і «Infarction» фокус зміщений на зони міокарда, що підтверджує коректну ідентифікацію патологій.

SHAP (SHapley Additive exPlanations) надав додаткову інформацію про внесок кожного пікселя у прийняте рішення. Це підвищило прозорість моделі та довіру до її результатів. На рисунку 4.3 показано приклад інтеграції SHAP із Grad-CAM для спільної візуалізації.

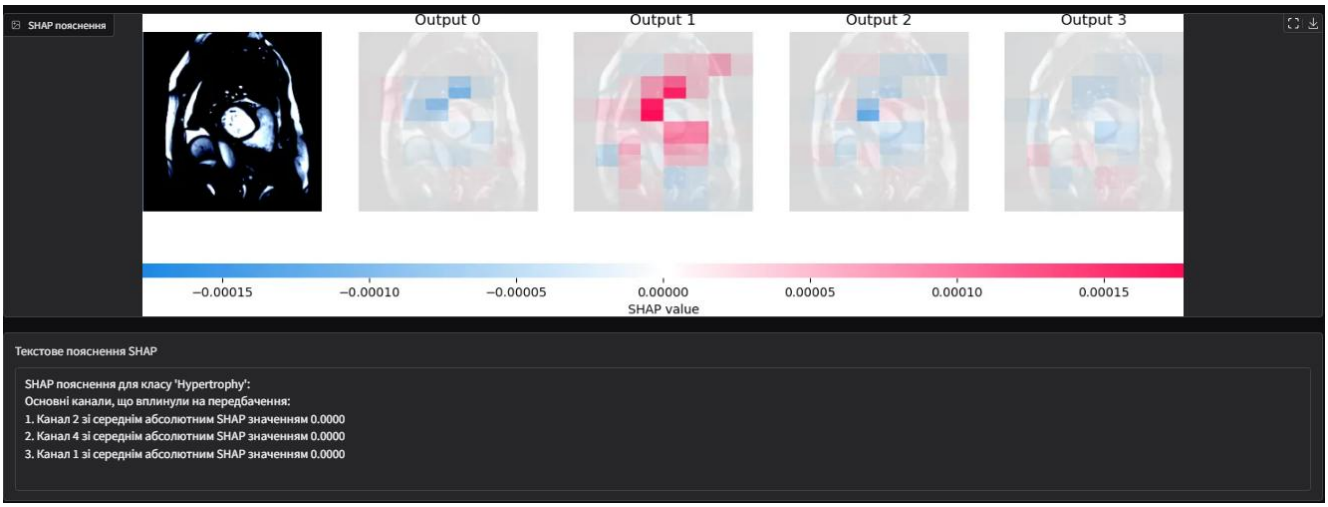


Рисунок 4.6 – Приклад SHAP пояснення

– графічне пояснення: Відображає вагомість кожної області зображення;

– текстове пояснення: Система генерує опис, що пояснює важливість ключових регіонів.

Цей тест-кейс (таблиця 4.2) перевіряє базову функціональність завантаження МРТ-зображень. Успішне виконання підтвердило коректну обробку підтримуваних форматів, відображення попереднього перегляду та повідомлення про помилки у разі невідповідності формату. Це мінімізує ризик помилок на початкових етапах оброблення даних.

Результати тесту показали, що система відповідає очікуваним вимогам, забезпечуючи надійну перевірку вхідних даних і сповіщення користувача про

проблеми, якщо такі виникають. Це мінімізує ризик помилок на початкових етапах оброблення зображень.

Таблиця 4.2 – Тест-кейс TC0001

Тест-кейс ID: TC0001	Пріоритет: 1	Створено:01.12.2024
Назва	Перевірка можливості завантаження МРТ-зображення в систему	
Передумови	Запуск вебзастосунку, готовність до завантаження файлу.	
Кроки	1.Відкрити вебінтерфейс системи. 2. Завантажити файл зображення у форматі JPEG. 3.Перевірити чи відображається попередній перегляд завантаженого зображення.	
Очікуваний результат	Файл успішно завантажується, відображається попередній перегляд.	
Результат виконання	Успішно. Система правильно обробляє підтримувані формати й повідомляє про помилки у разі невідповідності.	

Цей тест-кейс (таблиця 4.3) перевіряє функціональність модуля класифікації, який є ключовим для визначення патологій серця на МРТ-зображеннях.

Таблиця 4.3 – Тест-кейс TC0002

Тест-кейс ID: TC0002	Пріоритет: 1	Створено: 01.12.2024 Діхтяр. М.О.
Назва	Перевірка роботи модуля класифікації	
Передумови	Область серця успішно виділена.	
Кроки:	1. Виконати класифікацію зображення. 2. Перевірити результати класифікації для кожного класу (Normal, Hypertrophy, Infarction).	
Очікуваний результат	Модель класифікує зображення з коректними результатами для кожного класу.	
Результат виконання	Успішно. Класифікація працює належним чином.	

Успішне виконання тесту підтверджує, що модель здатна правильно класифікувати зображення у відповідні класи: «Normal», «Hypertrophy», «Infarction». Тест також гарантує, що модель здатна точно розпізнавати патології на основі витягнутих ознак.

Результати тесту показали, що модуль класифікації працює стабільно та правильно. Це підтверджує ефективність навчання моделі та дає змогу використовувати її в реальних умовах для діагностування патологій. Успішна класифікація кожного класу є критично важливим для забезпечення надійності системи.

Цей тест-кейс (таблиця 4.4) перевіряє функціональність інтерпретації результатів класифікації за допомогою Grad-CAM та SHAP. Успішна генерація теплових карт Grad-CAM та текстових пояснень SHAP є критично важливою для забезпечення прозорості та зрозумілості роботи моделі. Інтерпретація результатів дає змогу медичним фахівцям оцінити зони патологій на зображенні та краще зрозуміти, на основі яких ознак модель приймала свої рішення.

Таблиця 4.4 – Тест-кейс TC0003

Тест-кейс ID: TC0003	Пріоритет: 1	Створено: 01.12.2024 Діхтяр. М.О.
Назва	Перевірка інтерпретації результатів	
Передумови	Завершена класифікація зображення.	
Кроки	1.Виконати генерацію теплових карт Grad-CAM. 2.Виконати генерацію текстових пояснень SHAP. 3.Перевірити відображення пояснень у вебінтерфейсі.	
Очікуваний результат	Теплові карти й текстові пояснення успішно згенеровані й відображені для користувача.	
Результат виконання	Успішно. Інтерпретація працює належним чином.	

Результати тесту показали, що система успішно генерує як теплові карти, так і текстові пояснення, які правильно відображаються у вебінтерфейсі. Це значно

підвищує довіру до моделі, адже забезпечує лікарів можливістю переглянути, які зони найбільше вплинули на рішення моделі. Успішна інтерпретація підтверджує, що система до використання в клінічній практиці.

Цей тест-кейс (таблиця 4.5) перевіряє функціональність збереження результатів аналізу, зокрема теплових карт Grad-CAM і текстових пояснень SHAP, у різних форматах (PNG, DICOM, JPEG) і на обрані користувачем носії (локальний диск чи сервер). Збереження результатів є ключовим для подальшого аналізу, архівування та можливого обміну інформацією між лікарями для консультацій.

Таблиця 4.5 – Тест-кейс TC0004

Тест-кейс ID: TC0004	Пріоритет: 2	Створено: 01.12.2024 Діхтяр. М.О.
Назва	Перевірка збереження результатів аналізу	
Передумови	Завершена класифікація зображення та інтерпретація результатів.	
Кроки	1. Перейти до опції збереження результатів у вебінтерфейсі. 2. Вибрати формат для збереження (наприклад, PNG, DICOM, JPEG). 3. Вказати місце для збереження (локальний диск або сервер) 4. Натиснути кнопку «Save» та підтвердити збереження.	
Очікуваний результат	Результати аналізу (включно з тепловими картами та текстовими поясненнями) успішно збережені у вибраному форматі й місці.	
Результат виконання	Успішно. Збереження виконано без помилок.	

Результати тестування підтвердили, що система правильно зберігає результати у вибраних форматах і місцях збереження без помилок. Це забезпечує зручність у роботі лікарів, які можуть легко зберігати та використовувати результати аналізу для подальших досліджень або клінічних обговорень. Така функціональність підвищує успішність використання системи у практичних умовах.

У результаті проведеного тестування програмного продукту неправильно працюючих функцій виявлено не було.

4.3 Вимоги до розгортання вебзастосунку та інструкція користувача

Для успішного розгортання вебзастосунку необхідно спочатку налаштувати середовище розробки. Почніть з встановлення Python версії 3.7 або вище, завантаживши його з офіційного сайту. Під час інсталяції обов’язково виберіть опцію "Add Python to PATH", щоб мати змогу запускати Python з командного рядка. Після встановлення перевірте версію Python командою `python --version`. Далі створіть віртуальне середовище, використовуючи команду `python -m venv venv`, і активуйте його (`venv\Scripts\activate` для Windows або `source venv/bin/activate` для macOS/Linux). Віртуальне середовище забезпечить ізоляцію залежностей проекту від системних пакетів.

Після налаштування середовища переходьте до встановлення необхідних бібліотек. Переконайтеся, що віртуальне середовище активовано, і виконайте команду `pip install torch torchvision timm gradio`. Бібліотеки `torch` та `torchvision` необхідні для машинного навчання та обробки зображень, `timm` містить попередньо навчені моделі для комп'ютерного зору, а `gradio` дозволяє створювати інтуїтивні веб-інтерфейси. Перевірте успішність установки, імпортуючи бібліотеки у Python-інтерпретаторі: `import torch`, `import torchvision`, `import timm`, `import gradio`. Відсутність помилок підтвердить коректну установку.

Після встановлення бібліотек можна запускати веб-застосунок. Переконайтесь, що ви знаходитесь у кореневій папці проекту, де розташований файл `app.py`, і виконайте команду `python app.py`. Це запустить сервер, який ініціює веб-інтерфейс через Gradio, і у командному рядку з’явиться посилання, наприклад `http://127.0.0.1:7860/`. Відкрийте це посилання у браузері, щоб взаємодіяти з застосунком. Для полегшення управління залежностями створіть файл `requirements.txt` за допомогою `pip freeze > requirements.txt` і використовуйте `pip install -r requirements.txt` для встановлення бібліотек на інших машинах. Рекомендується також використовувати систему контролю версій, наприклад Git, для управління кодом і співпраці з іншими розробниками. Перед розгортанням на

продакшн-сервері ретельно протестуйте застосунок та дотримуйтесь найкращих практик безпеки.

Веб-інтерфейс Gradio відкриє посилання для доступу в браузері (рисунок 4.7).

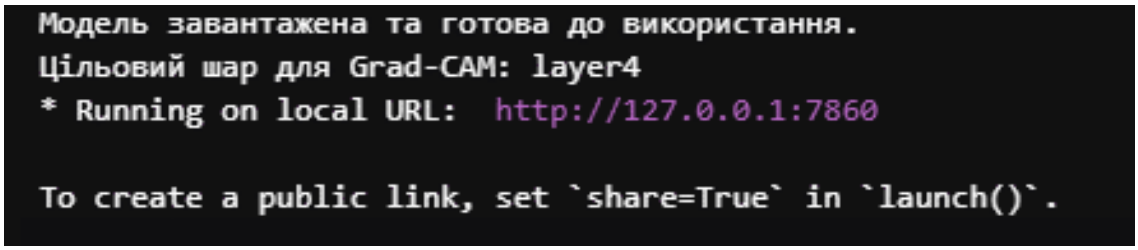


Рисунок 4.7 – Запуск python app

Робота з системою

– завантажте зображення за допомогою кнопки «Завантажити зображення» (рисунок 4.8);

– отримайте результати класифікації з Grad-CAM та SHAP (рисунок 4.9).

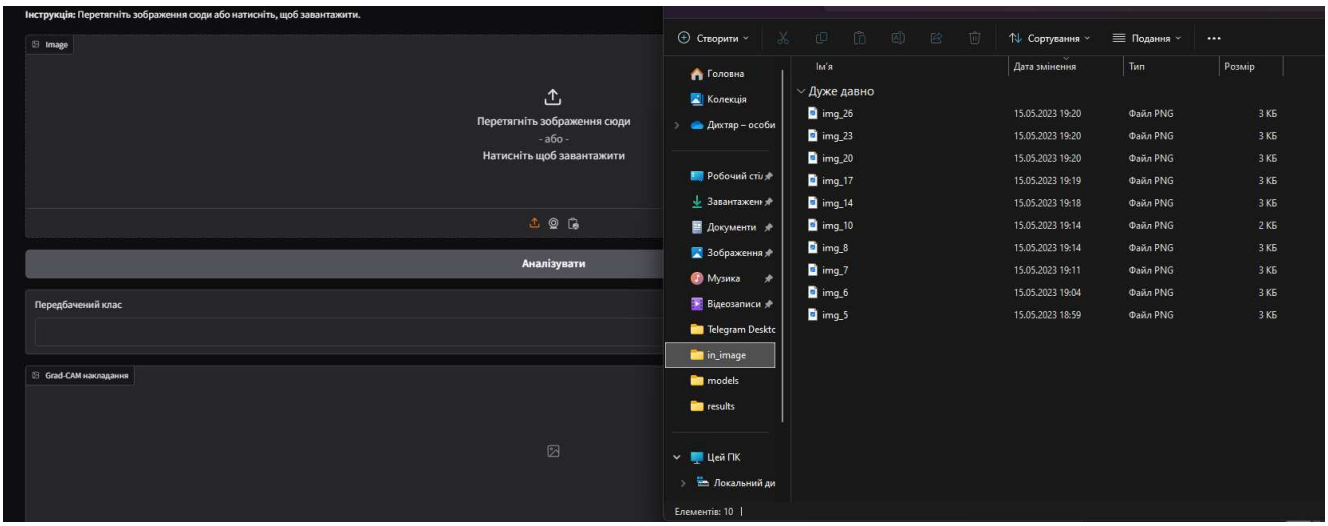


Рисунок 4.8 – Приклад завантаження зображення

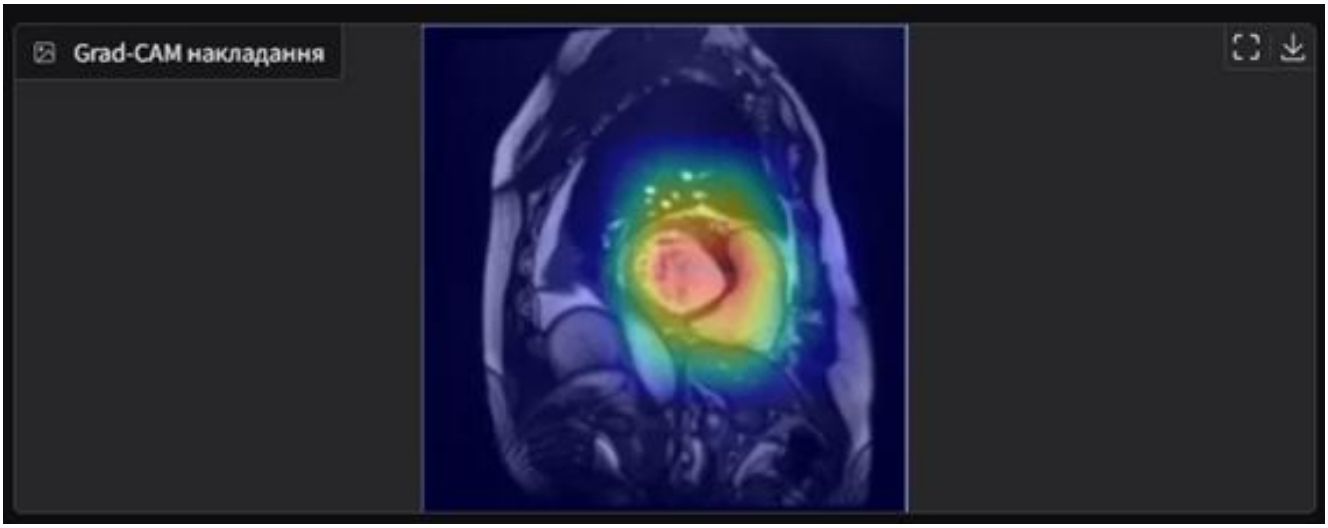


Рисунок 4.9 – Результат Grad-CAM

На рисунках 4.10 та 4.11 зображені приклади інтерпретації зображень МРТ для різних патологій.

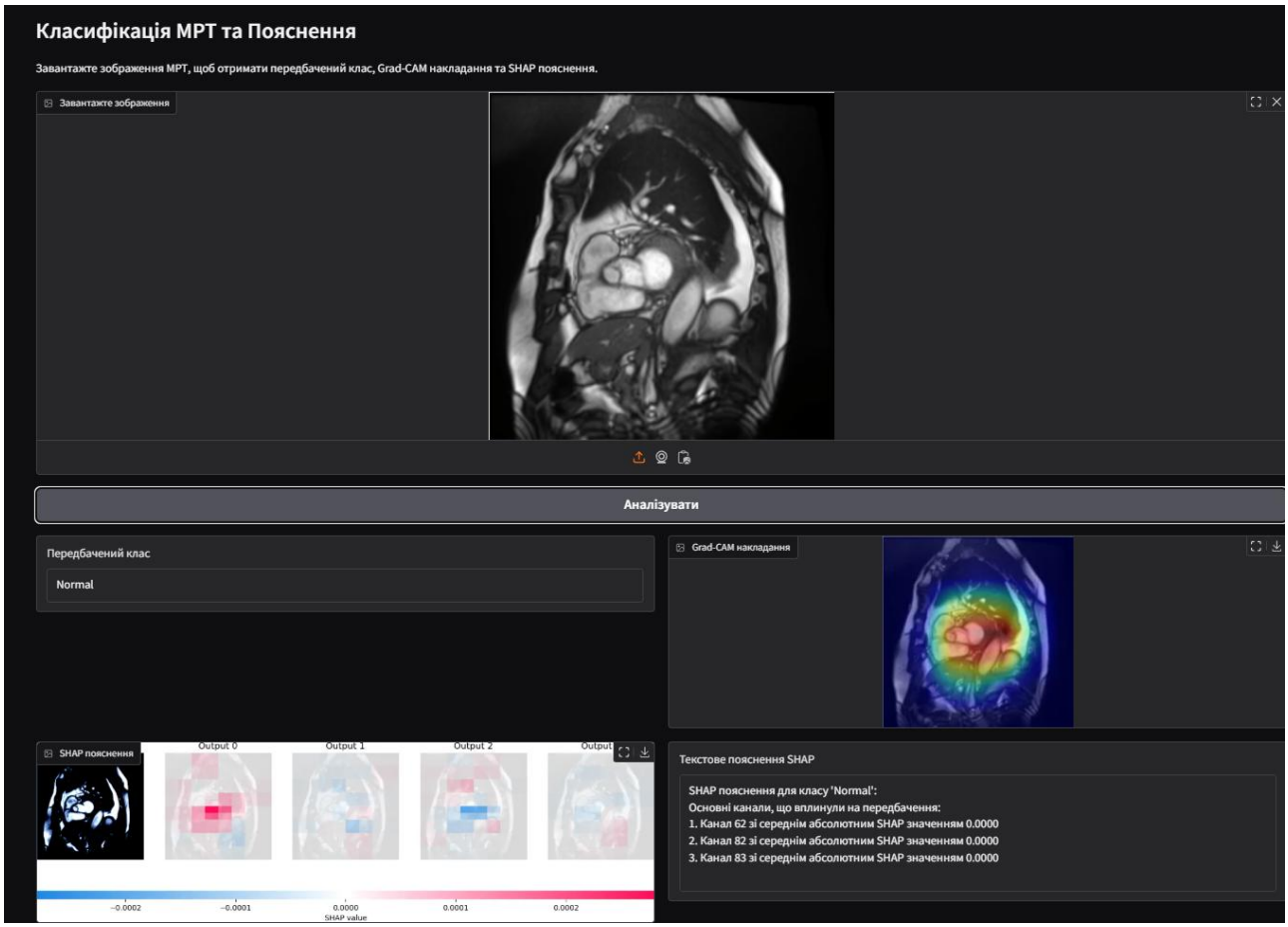


Рисунок 4.10 – Результат інтерпретування зображення

Експериментальне тестування підтвердило результативність застосунку щодо виявлення патологій серця. Основні результати:

– висока точність – значення Precision і Recall для ключових класів перевищують 75%, а ROC-AUC (0.94) демонструє високу дискримінативну здатність;

– прозорість – візуалізація Grad-CAM і SHAP забезпечує інтерпретацію рішень моделі, що є критично важливим для медичних фахівців;

– зручність у використанні – інтеграція з Gradio дає змогу легко використовувати систему навіть користувачам без глибоких технічних знань.

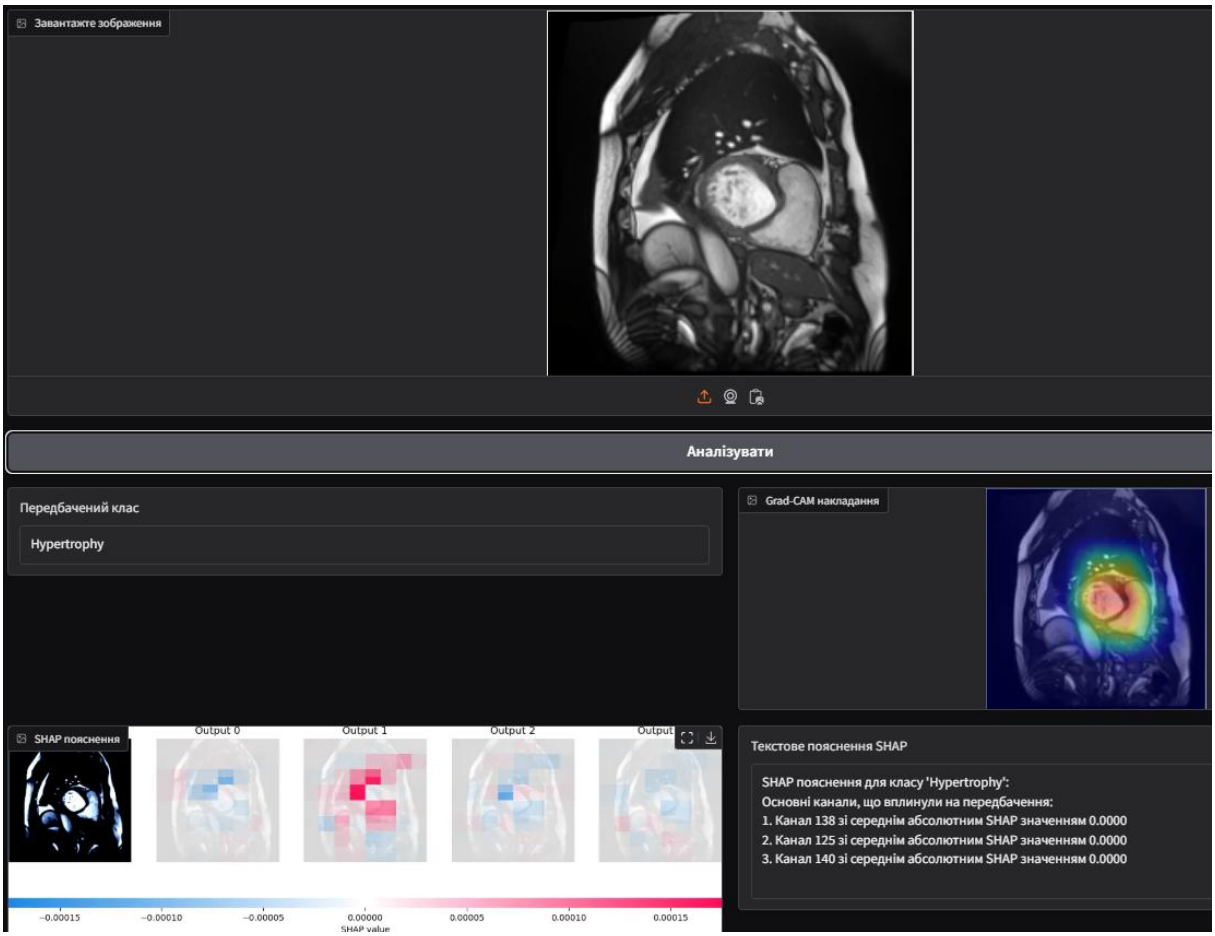


Рисунок 4.11 – Приклад результатів інтерпретування

Система відповідає вимогам, висунутим у теоретичній частині, і демонструє потенціал для подальшого клінічного застосування.

Висновки до розділу 4

У четвертому розділі проведено комплексне експериментальне тестування розробленого вебзастосунку, що дало змогу оцінити її продуктивність у реальних умовах роботи. Дослідження зосереджено на двох основних аспектах: оцінці точності класифікації патологій та перевірці здатності системи інтерпретувати результати, використовуючи методи Grad-CAM та SHAP. Результати тестування показали високі показники точності і класифікації, а також обґрунтованість висновків.

Досліджено вплив різних параметрів на продуктивність системи, зокрема, кількість епох навчання, а також метрики точності, такі як Precision, Recall, F1-Score та ROC-AUC для кожного класу. Проведено аналіз матриць плутанини, що дало змогу виявити наявність певних проблем та внести необхідні зміни. Так, основні метрики оцінки, такі як Pointing Game та Intersection over Union (IoU), вказали на здатність моделі точно виділяти зони патологій, що є важливим для надання користувачу інтерпретованих результатів. Система стабільно показала високий середній показник Pointing Game (0.92) та IoU (від 0.79 до 0.83), що свідчить про її надійність у сегментації.

На основі отриманих даних розроблено рекомендації щодо покращення точності класифікації та інтерпретації. Важливим аспектом тестування стало визначення вимог до розгортання системи, а також підготовка інструкцій для користувача, що дає змогу впровадити систему у клінічну практику.

У ході експериментального аналізу продемонстровано, що розроблена система є перспективним інструментом для автоматизованого діагностування серцевих патологій за МРТ, який поєднує в собі високу точність виявлення патологій та зрозумілість їхньої інтерпретації. Результати, отримані у розділі 4 підтверджують практичну цінність запропонованих методів, а також готовність системи до використання у реальних умовах.

Загальні висновки

Кваліфікаційна робота магістра присвячена підвищенню рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання, через удосконалення методу інтерпретування результатів виявлення патологій серця у вебзастосунку.

В ході дослідження було проаналізовано актуальні виклики у цій галузі та запропоновано шляхи їх вирішення, зокрема шляхом застосування архітектур U-Net для сегментації та ResNet50 для класифікації, а також методів Grad-CAM і SHAP для інтерпретації результатів.

У межах даної роботи було успішно спроєктовано метод інтерпретування результатів виявлення патологій серця за зображенням МРТ. Удосконалений метод забезпечує не лише виявлення патологій з високою точністю, але й прозорість прийнятих рішень, що сприяє довірі медичних фахівців до автоматизованих систем. Метод поєднує сегментацію серця (U-Net), класифікацію патологій (ResNet50) та пояснення рішень (Grad-CAM і SHAP). Використання цих методів дає змогу не лише автоматизувати процес діагностування, але й забезпечити прозорість рішень для лікарів.

На основі запропонованого методу було розроблено, протестовано та впроваджено вебзастосунок, що поєднує методи глибокого навчання та пояснюваного штучного інтелекту для автоматизованої діагностування патологій серця за зображеннями МРТ. Розроблена функціональна структура вебзастосунку охоплює всі етапи процесу: від завантаження та попередньої обробки зображень до їхньої класифікації, інтерпретації, виведення та збереження результатів. Експериментальне тестування системи підтвердило її результативність та стабільність роботи на різних наборах даних, зокрема високу точність класифікації (понад 75% для ключових класів), а також адекватність виділення патологічних зон за допомогою теплових карт Grad-CAM і текстових пояснень SHAP.

Було проведено навчання нейронної мережі ResNet50 для класифікації трьох класів патологій: нормальний стан, гіпертрофія міокарда та інфаркт міокарда.

Результати показали високу точність класифікації (Precision, Recall, F1-Score перевищували 75%) та стабільність на різних наборах даних.

Експериментальне тестування підтвердило результативність системи, зокрема її здатність до точного виявлення патологій і надання інтерпретованих результатів. Використання теплових карт Grad-CAM і текстових пояснень SHAP забезпечило прозорість рішень, що значно підвищує довіру до системи з боку медичних фахівців.

Інтеграція веб-інтерфейсу Gradio спростила взаємодію з вебзастосунком, роблячи його доступним навіть для користувачів без технічної підготовки. Деталізовані інструкції забезпечують легке розгортання вебзастосунку на будь-якому комп’ютері, що підтримує Python.

Загалом в роботі успішно розв’язано усі поставлені завдання дослідження та досягнуто мети роботи: спроєктовано метод інтерпретування результатів виявлення патологій серця за зображенням МРТ та виконану його програмну реалізацію для автоматизованої діагностування патологій серця, що дає змогу поєднати точність класифікації з інтерпретованістю результатів. Удосконалений метод дає змогу візуалізувати важливі області на зображеннях, що вплинули на рішення моделі, та надавати текстові пояснення для кожного випадку, забезпечуючи медичних фахівців необхідною інформацією для прийняття обґрунтованих рішень.

Варто зазначити, що запропонований метод має певні обмеження, пов'язані зі складністю обробки дуже великих наборів даних, а також залежність точності від якості вхідних зображень. Розв'язання цих обмежень у майбутньому може включати подальшу оптимізацію алгоритмів, покращення методів збору даних та розширення можливостей системи за рахунок інтеграції додаткових методів штучного інтелекту.

Перелік посилань

1. Діхтяр М. О., Радюк П. М., Скрипник Т. К. Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ. Актуальні проблеми комп'ютерних наук АПКН-2024 : матеріали XVI Всеукр. науково-практ. конф., м. Хмельницький, 15–16 листоп. 2024 р. Хмельницький, 2024. Хмельницький : ХНУ, 2024. С. 189–191. URL: https://elar.khmnu.edu.ua/handle/123456789/17156 (дата звернення: 06.12.2024).

2. Myocardium segmentation using two-step deep learning with smoothed masks by Gaussian blur / V. Slobodzian et al. *Proceedings of the 6th International Conference on Informatics & Data-Driven Medicine* : CEUR-Workshop Proceedings, Bratislava, Slovakia, 17–19 November 2023 / ed. by N. Shakhovska et al. Aachen, 2024. P. 77–91. URL: https://ceur-ws.org/Vol-3609/paper7.pdf (дата звернення: 17.09.2024).

3. Robust R-peak detection using deep learning based on integrating domain knowledge / O. Kovalchuk et al. *Proceedings of the 6th International Conference on Informatics & Data-Driven Medicine* : CEUR-Workshop Proceedings, Bratislava, Slovakia, 17–19 November 2023 / ed. by N. Shakhovska et al. Aachen, 2024. P. 1–14. URL: https://ceur-ws.org/Vol-3609/paper1.pdf (дата звернення: 17.09.2024).

4. GitHub Repository: sicara/tf-explain. URL: https://github.com/sicara/tf-explain (дата звернення: 17.09.2024).

5. Captum Documentation: Model Interpretability for PyTorch. URL: https://captum.ai/ (дата звернення: 17.09.2024).

6. tf-explain Documentation: Interpretability Methods for tf.keras models with Tensorflow 2.x. URL: https://tf-explain.readthedocs.io/en/latest/ (дата звернення: 17.09.2024).

7. Captum Tutorial: Model Interpretability using Captum. URL: https://pytorch.org/tutorials/recipes/recipes/Captum_Recipe.html (дата звернення: 17.09.2024).

8. tf-explain on PyPI: tf-explain. URL: https://pypi.org/project/tf-explain/ (дата звернення: 17.09.2024).

9. Captum Insights: Interactive Model Interpretability for PyTorch. URL: https://captum.ai/tutorials/ (дата звернення: 19.09.2024).

10. tf-explain Overview: Interpretability Methods for TensorFlow 2.0. URL: https://tf-explain.readthedocs.io/en/latest/overview.html (дата звернення: 19.09.2024).

11. Human–in–the–loop approach based on MRI and ECG for healthcare diagnosis / P. Radiuk et al. *Proceedings of the 5th International Conference on Informatics & Data-Driven Medicine* : CEUR-Workshop Proceedings, Lyon, France, 18–20 November 2022 / ed. by N. Shakhovska et al. Aachen, 2022. P. 9–20. URL: https://ceur-ws.org/Vol-3302/paper1.pdf (дата звернення: 19.09.2024).

12. Ribeiro M.T., Singh S., Guestrin C. «Why Should I Trust You?»: Explaining the Predictions of Any Classifier // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. P. 1135–1144. URL: https://dl.acm.org/doi/10.1145/2939672.2939778 (дата звернення: 19.09.2024).

13. Krak Iu., Barmak O., Radiuk P. Information technology for early diagnosis of pneumonia on individual radiographs. *The 3rd International Conference on Informatics & Data-Driven Medicine (IDDM 2020)* : CEUR-Workshop Proceedings. Vol. 2753. (Växjö, Sweden, 19–21 November 2020). CEUR-WS.org, Aachen, 2020. P. 11–21. URL: https://ceur-ws.org/Vol-2753/paper3.pdf (дата звернення: 17.09.2024).

14. Krak Iu., Barmak O., Radiuk P. Detection of early pneumonia on individual CT scans with dilated convolutions. The 2nd International Workshop on Intelligent Information Technologies & Systems of Information Security (IntelITSIS-2021) : CEUR-Workshop Proceedings. Vol. 2853. (Khmelnytskyi, Ukraine, 24–26 March 2021) / ed. by T. Hovorushchenko et al. CEUR-WS.org, Aachen, 2021. P. 214–227. URL: http://ceur-ws.org/Vol-2853/ (дата звернення: 19.09.2024).

15. Captum · Model Interpretability for PyTorch. *Captum · Model Interpretability for PyTorch*. URL: https://captum.ai/ (дата звернення: 25.09.2024).

16. Smilkov D., Thorat N., Kim B., Viégas F., Wattenberg M. SmoothGrad: removing noise by adding noise // arXiv preprint arXiv:1706.03825. 2017. URL: https://arxiv.org/abs/1706.03825 (дата звернення: 19.09.2024).

17. GitHub - sicara/tf-explain: Interpretability Methods for tf.keras models with Tensorflow 2.x. *GitHub*. URL: https://github.com/sicara/tf-explain (дата звернення: 25.09.2024).

18. Montavon G., Binder A., Lapuschkin S., Samek W., Müller K.-R. Layer-Wise Relevance Propagation: An Overview // Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. 2019. P. 193–209. URL: https://link.springer.com/chapter/10.1007/978-3-030-28954-6_10 (дата звернення: 19.09.2024).

19. GitHub - marcotcr/lime: Lime: Explaining the predictions of any machine learning classifier. *GitHub*. URL: https://github.com/marcotcr/lime (дата звернення: 25.09.2024).

20. Simonyan K., Vedaldi A., Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps // arXiv preprint arXiv:1312.6034. 2013. URL: https://arxiv.org/abs/1312.6034 (дата звернення: 19.09.2024).

21. Welcome to the SHAP documentation – SHAP latest documentation. *Welcome to the SHAP documentation – SHAP latest documentation*. URL: https://shap.readthedocs.io/en/latest/ (дата звернення: 25.09.2024).

22. Mahendran A., Vedaldi A. Understanding Deep Image Representations by Inverting Them // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015. P. 5188–5196. URL: 10.1109/CVPR.2015.7299155. (дата звернення: 25.09.2024).

23. GitHub - eli5-org/eli5: A library for debugging/inspecting machine learning classifiers and explaining their predictions. *GitHub*. URL: https://github.com/eli5-org/eli5 (дата звернення: 25.09.2024).

24. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of the 32nd International Conference on Machine Learning (ICML). 2015. P. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html (дата звернення: 19.09.2024).

25. Welcome to iNNvestigate’s documentation! – iNNvestigate not set documentation. *Welcome to iNNvestigate’s documentation! – iNNvestigate not set documentation*. URL: https://innvestigate.readthedocs.io/en/latest/ (дата звернення: 25.09.2024).

26. Huang G., Liu Z., Van Der Maaten L., Weinberger K.Q. Densely Connected Convolutional Networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 4700–4708.

27. Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z. Rethinking the Inception Architecture for Computer Vision // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 2818–2826.

28. Goodfellow I., Bengio Y., Courville A. Deep Learning. – Cambridge: MIT Press, 2016. – 775 p. (дата звернення: 02.10.2024).

29. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems (NIPS). 2012. P. 1097–1105.

30. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-Based Learning Applied to Document Recognition // Proceedings of the IEEE. – 1998. – Vol. 86, No. 11. P. 2278–2324. URL: https://ieeexplore.ieee.org/document/726791 (дата звернення: 02.10.2024).

31. Zeiler M.D., Fergus R. Visualizing and Understanding Convolutional Networks // European Conference on Computer Vision (ECCV). 2014. P. 818–833. URL: https://link.springer.com/chapter/10.1007/978-3-319-10590-1_53 (дата звернення: 02.10.2024).

32. Montavon G., Binder A., Lapuschkin S., Samek W., Müller K.-R. Layer-Wise Relevance Propagation: An Overview // Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. 2019. P. 193–209. URL: https://link.springer.com/chapter/10.1007/978-3-030-28954-6_10 (дата звернення: 02.10.2024).

33. Bach S., Binder A., Montavon G., Klauschen F., Müller K.-R., Samek W. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation // PLoS ONE. 2015. – Vol. 10, No. 7. – e0130140. URL:

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140 (дата звернення: 02.10.2024).

34. Simonyan K., Vedaldi A., Zisserman A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps // arXiv preprint arXiv:1312.6034. 2013. URL: https://arxiv.org/abs/1312.6034 (дата звернення: 02.10.2024).

35. Zhou B., Khosla A., Lapedriza A., Oliva A., Torralba A. Learning Deep Features for Discriminative Localization // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 2921–2929. URL: https://openaccess.thecvf.com/content_cvpr_2016/papers/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf (дата звернення: 02.10.2024).

36. Mahendran A., Vedaldi A. Understanding Deep Image Representations by Inverting Them // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015. P. 5188–5196.

37. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition // arXiv preprint arXiv:1409.1556. 2014. URL: https://arxiv.org/abs/1409.1556 (дата звернення: 02.10.2024).

38. Dosovitskiy A., Brox T. Generating Images with Perceptual Similarity Metrics Based on Deep Networks // Advances in Neural Information Processing Systems (NIPS). 2016. P. 658–666.

39. Russakovsky O., Deng J., Su H., et al. ImageNet Large Scale Visual Recognition Challenge // International Journal of Computer Vision. 2015. Vol. 115, No. 3. P. 211–252. URL: https://link.springer.com/article/10.1007/s11263-015-0816-y (дата звернення: 02.10.2024).

40. Shwartz-Ziv R., Tishby N. Opening the Black Box of Deep Neural Networks via Information // arXiv preprint arXiv:1703.00810. 2017. URL: https://arxiv.org/abs/1703.00810 (дата звернення: 02.10.2024).

41. ACDC Challenge. *Centre de Recherche en Acquisition et Traitement de l'Image pour la Sante | CREATIS*. URL: https://www.creatis.insa-lyon.fr/Challenge/acdc/databases.html (дата звернення: 10.10.2024).

ДОДАТКИ

Додаток А

Копії наукових публікацій

Актуальні проблеми комп'ютерних наук

УДК 004.4

Діхтяр М.О., Радюк П.М., Скрипник Т.К.

Хмельницький національний університет

МЕТОД ІНТЕРПРЕТУВАННЯ РЕЗУЛЬТАТІВ ВИЯВЛЕННЯ ПАТОЛОГІЙ СЕРЦЯ ЗА ЗОБРАЖЕННЯМ МРТ

Запропоновано метод інтерпретування результатів виявлення патологій серця за допомогою згорткових нейронних мереж (ЗНМ), що використовує сегментовані зображення магнітно-резонансної томографії (МРТ). Метод забезпечує виявлення патологій та їх візуалізацію з поясненням результатів для покращення діагностики та підвищення довіри медичних працівників до автоматизованих систем.

A method for interpreting results of detecting heart pathologies using convolutional neural networks (CNN) is proposed, which leverages segmented magnetic resonance imaging (MRI) data. This method provides both pathology detection and visual explanation, improving diagnostic accuracy and fostering trust among medical professionals in automated systems.

Для реалізації автоматизованої діагностики серцево-судинних захворювань важливим є розробка методів інтерпретування результатів виявлення патологій серця за допомогою МРТ. Одним із таких методів є використання згорткових нейронних мереж, що дають можливість проводити аналіз медичних зображень на рівні пікселів, виявляючи ключові ознаки, які відповідають патологічним зонам. Сучасні методи інтерпретування результатів включають Grad-CAM, SHAP та LIME, які надають можливість візуалізувати важливі області на зображенні, що вплинули на рішення моделі.

Grad-CAM (Gradient-weighted Class Activation Mapping) створює теплові карти, які показують, які ділянки зображення найбільше вплинули на результат моделі. SHAP (SHapley Additive exPlanations) обчислює внесок кожної ознаки у прогноз моделі, а LIME (Local Interpretable Model-agnostic Explanations) будує локальну спрощену модель, що дає змогу пояснити конкретні результати.

Запропонований метод базується на архітектурі ResNet, що використовується для класифікації зображень та виявлення патологій. Він включає три основні етапи: підготовку даних, навчання моделі та візуалізацію результатів. Схема методу показана на рисунку 1.

На першому етапі здійснюється підготовка сегментованих зображень МРТ серця. Це включає попередню обробку даних, а саме нормалізацію інтенсивності пікселів, аугментацію та сегментацію серця на основні анатомічні структури.

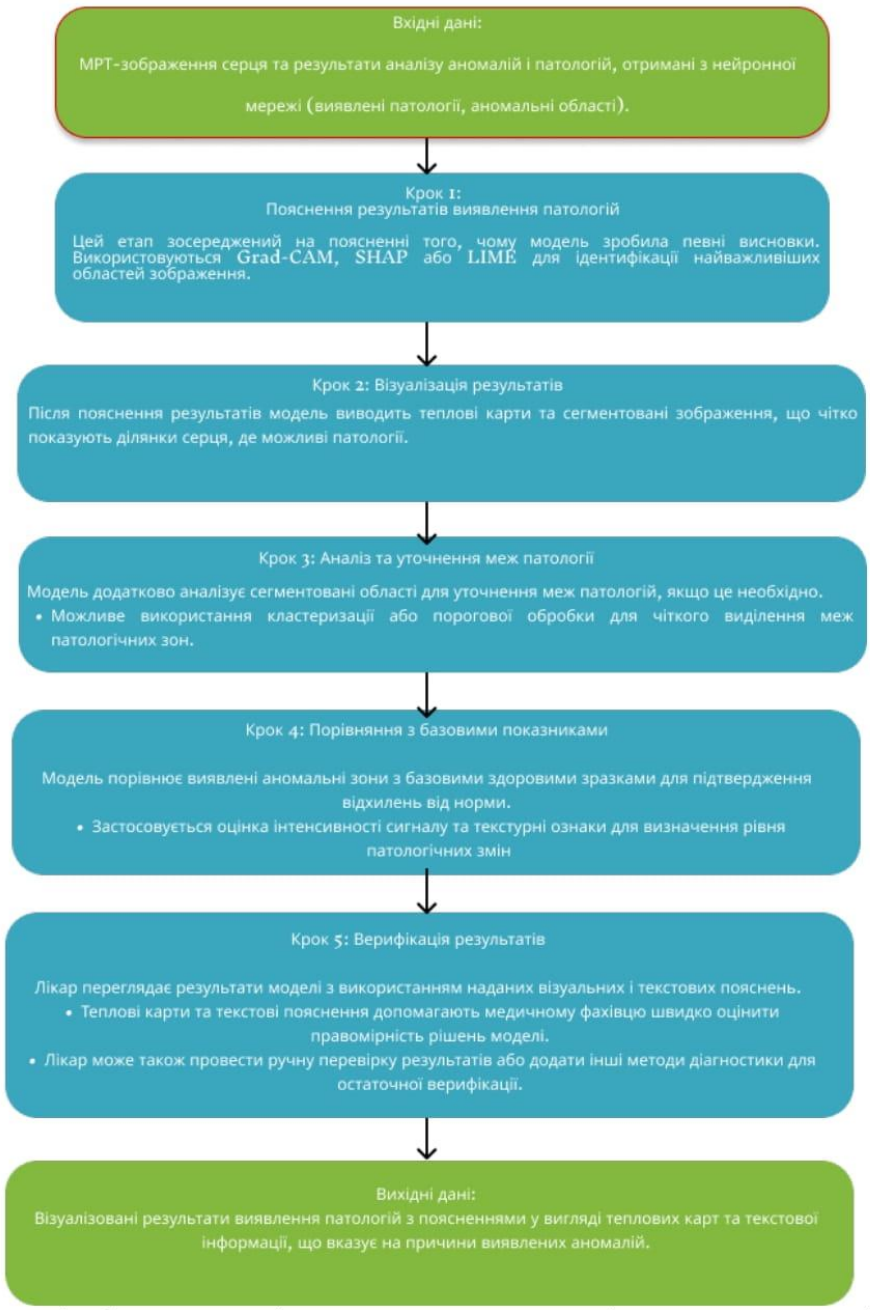


Рисунок 1 – Схема методу інтерпретування результатів виявлення патологій за зображенням МРТ

Другий етап – це навчання згорткової нейронної мережі, що використовує архітектуру ResNet. Модель навчалася на великих наборах даних із сегментованими зображеннями серця, де кожна зона була позначена як патологічна або здорова. Функція втрат бінарної перехресної ентропії була використана для точної класифікації патологій.

На третьому етапі проводиться інтерпретація результатів. Для цього використовуються методи Grad-CAM для створення теплових карт, які допомагають візуалізувати патологічні зони на зображеннях МРТ. SHAP використовується для розрахунку внеску кожного пікселя в остаточне рішення, що дає змогу лікарям краще розуміти, чому модель зробила конкретне діагностичне припущення.

Отже, запропонований метод інтерпретування результатів виявлення патологій серця за зображеннями МРТ на основі архітектури ResNet та інтерпретаційних методів Grad-CAM та SHAP забезпечує високу точність і прозорість у діагностиці серцево-судинних захворювань. Подальші дослідження включатимуть оптимізацію методу для підвищення швидкості обробки даних і вивчення можливостей інтеграції з іншими інструментами медичної візуалізації.

Перелік посилань

1. Shiota M. N. Basic and discrete emotion theories. Emotion Theory: The Routledge Comprehensive Guide. Routledge. 2024. pp 310–330.
2. Удадесс М. А. Емоційний інтелект та його роль в житті людини. Proceedings of the XVII International Scientific and Practical Conference. Tokyo. 2022. С 937–938.
3. IBM. What is natural language processing? URL: https://www.ibm.com/topics/natural-language-processing
4. Wankhade M., Rao A. C. S., Kulkarni C. A survey on sentiment analysis methods, applications, and challenges. Artificial Intelligence Review. Volume 55. 2022. pp. 5731–5780.
5. Reveal. BERT, MBERT, and the Quest to Understand. URL: https://www.revealdata.com/blog/bert-mbert-and-the-quest-to-understand
6. GitHub. goemotions URL: https://github.com/google-research/google-research/tree/master/goemotions

Додаток Б

Лістинг програмного коду

```
import os
import h5py
import numpy as np
from collections import Counter
import json
import time
import copy
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Імпорти PyTorch та torchvision
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset, Dataset, ConcatDataset
from torch.utils.tensorboard import SummaryWriter
import torchvision.transforms as transforms

# Імпорти з sklearn
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, classification_report, precision_score, recall_score, f1_score, roc_auc_score, balanced_accuracy_score

# Імпорт бібліотеки timm для створення моделей
import timm  # Для створення моделей ResNet та EfficientNet

# Імпорти для Grad-CAM та інтерпретації
from torchcam.methods import GradCAM
from torchcam.utils import overlay_mask

# Імпорти для мішаної точності (AMP)
from torch.cuda.amp import GradScaler, autocast

# Імпорт PIL для роботи із зображеннями
from PIL import Image

print("Імпорти виконані успішно.")
classes = ['Normal', 'Hypertrophy', 'Infarction', 'Other']  # Замініть на ваші реальні назви класів
print(f"Класи: {classes}")
class ACDCDataset(Dataset):
    def __init__(self, file_list, labels, transform=None):
        """
        Args:
            file_list (list): Список шляхів до .h5 файлів.
            labels (list): Список міток відповідних зображень.
            transform (callable, optional): Трансформації, що застосовуються до зображень.
        """
        self.file_list = file_list
        self.labels = labels
        self.transform = transform

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

    def __getitem__(self, idx):
```

```
        image_path = self.file_list[idx]
        label = self.labels[idx]

        with h5py.File(image_path, 'r') as f:
            if 'image' in f:
                image = f['image'][:]
            else:
                raise ValueError(f"Ключ 'image' не знайдено у файлі: {image_path}")

        # Обробка формату зображення
        if image.ndim == 2:
            # Градієнти сірого (Height, Width), конвертуємо до RGB
            image = np.stack([image] * 3, axis=-1)
        elif image.ndim == 3:
            # Обробка channel-first та channel-last форматів
            if image.shape[0] <= 4:
                # Channel-first
                if image.shape[0] > 3:
                    image = image[:3, :, :]
                image = np.transpose(image, (1, 2, 0))
            else:
                # Channel-last
                if image.shape[2] > 3:
                    image = image[:, :, :3]
        else:
            raise ValueError(f"Непідтримувана форма зображення: {image.shape}")

        # Перевірка кількості каналів після оброблення
        if image.shape[2] == 1:
            image = np.concatenate([image] * 3, axis=2)
        elif image.shape[2] != 3:
            raise ValueError(f"Непідтримувана кількість каналів після оброблення: {image.shape[2]}")

        # Конвертація до uint8
        if image.dtype != np.uint8:
            if image.max() <= 1.0:
                image = (image * 255).astype(np.uint8)
            else:
                image = image.astype(np.uint8)

        # Конвертація до PIL.Image
        image_pil = Image.fromarray(image)

        # Застосування трансформацій
        if self.transform:
            image_pil = self.transform(image_pil)

        return image_pil, label
def ensure_rgb(image):
    if image.mode != 'RGB':
        return image.convert('RGB')
    return image

# Визначення трансформацій для тренувального та валідаційного наборів
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.Lambda(lambda img: ensure_rgb(img)),  # Забезпечення RGB
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
    ]),
```

```
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.Lambda(lambda img: ensure_rgb(img)),  # Забезпечення RGB
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
    ]),
}

print("Трансформації визначені успішно.")
def get_file_paths(directory, extension='.h5'):
    file_paths = []
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(extension):
                file_paths.append(os.path.join(root, file))
    return file_paths

# Вкажіть ваші директорії
training_slices_dir = r'C:\Users\maksk\.cache\kagglehub\datasets\anhoangvo\acdc-dataset\versions\1\ACDC_preprocessed\ACDC_training_slices'
training_volumes_dir = r'C:\Users\maksk\.cache\kagglehub\datasets\anhoangvo\acdc-dataset\versions\1\ACDC_preprocessed\ACDC_training_volumes'
testing_volumes_dir = r'C:\Users\maksk\.cache\kagglehub\datasets\anhoangvo\acdc-dataset\versions\1\ACDC_preprocessed\ACDC_testing_volumes'

# Збір файлів
training_slices_files = get_file_paths(training_slices_dir, extension='.h5')
training_volumes_files = get_file_paths(training_volumes_dir, extension='.h5')
testing_volumes_files = get_file_paths(testing_volumes_dir, extension='.h5')

print(f"Кількість тренувальних слайсів: {len(training_slices_files)}")
print(f"Кількість тренувальних обсягів: {len(training_volumes_files)}")
print(f"Кількість тестових обсягів: {len(testing_volumes_files)}")

# Функція для отримання міток з файлів
def extract_label(file_path):
    with h5py.File(file_path, 'r') as f:
        label = f['label'][()]
        if isinstance(label, np.ndarray):
            unique, counts = np.unique(label.flatten(), return_counts=True)
            label_counts = dict(zip(unique, counts))
            label_counts = {k: v for k, v in label_counts.items() if k != 0}  # Виключаємо фон або інші незадіяні класи
            label_idx = max(label_counts, key=label_counts.get) if label_counts else 0
        else:
            try:
                label_idx = int(label)
            except:
                label_idx = 0
        label_idx = int(label_idx)
        if not (0 <= label_idx < len(classes)):
            label_idx = 0
        return label_idx

# Визначення міток для тренувальних слайсів
all_labels_slices = []
for file_path in training_slices_files:
    label = extract_label(file_path)
    all_labels_slices.append(label)

# Визначення міток для тренувальних обсягів
all_labels_volumes = []
for file_path in training_volumes_files:
```

```
    label = extract_label(file_path)
    all_labels_volumes.append(label)

# Об'єднання міток
all_labels = all_labels_slices + all_labels_volumes

# Створення повної датасети
full_dataset = ConcatDataset([
    ACDCDataset(
        file_list=training_slices_files,
        labels=all_labels_slices,
        transform=data_transforms['train']
    ),
    ACDCDataset(
        file_list=training_volumes_files,
        labels=all_labels_volumes,
        transform=data_transforms['train']
    )
])

print(f"Розмір повної датасети: {len(full_dataset)}")

# Розподіл класів
label_counts = Counter(all_labels)
print(f"Розподіл класів у всіх тренувальних файлах: {label_counts}")

# Візуалізація розподілу класів
plt.figure(figsize=(8,6))
sns.barplot(x=classes, y=[label_counts.get(i, 0) for i in range(len(classes))])
plt.xlabel('Класи')
plt.ylabel('Кількість')
plt.title('Розподіл Класів у Тренувальних Даних')
plt.show()device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Використовується пристрій: {device}")

# Функція для створення моделі
def create_model(num_classes, architecture='resnet50', device='cpu'):
    """
    Створення моделі з відповідною архітектурою та кількістю класів.

    Args:
        num_classes (int): Кількість класів для класифікації.
        architecture (str): Назва архітектури моделі. Підтримуються 'resnet34', 'resnet50', 'efficientnet_b0'.
        device (torch.device): Пристрій для розміщення моделі.

    Returns:
        model (torch.nn.Module): Ініціалізована та перенесена на пристрій модель.
    """
    if architecture == 'resnet34':
        model = timm.create_model('resnet34', pretrained=True, in_chans=3, num_classes=num_classes)
    elif architecture == 'resnet50':
        model = timm.create_model('resnet50', pretrained=True, in_chans=3, num_classes=num_classes)
    elif architecture == 'efficientnet_b0':
        model = timm.create_model('efficientnet_b0', pretrained=True, in_chans=3, num_classes=num_classes)
    else:
        raise ValueError(f"Архітектура {architecture} не підтримується.")

    # Додавання Dropout перед класифікатором
    if hasattr(model, 'classifier'):
        num_ftrs = model.classifier.in_features
        model.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(num_ftrs, num_classes)
```

```
            )
        elif hasattr(model, 'fc'):
            num_ftrs = model.fc.in_features
            model.fc = nn.Sequential(
                nn.Dropout(0.5),
                nn.Linear(num_ftrs, num_classes)
            )
        else:
            raise AttributeError("Модель не має атрибуту 'classifier' або 'fc'.")

        return model.to(device)

    # Створення моделі
    architecture = 'resnet50'  # Або інша підтримувана архітектура
    num_classes = len(classes)  # Має бути 4

    model = create_model(num_classes=num_classes, architecture=architecture, device=device)

    # Перевірка кінцевого шару
    print("Model's final layer:", model.fc)
    print("Number of output classes:", model.fc[-1].out_features) def get_last_conv_layer(model):
        last_conv = None
        for name, module in model.named_modules():
            if isinstance(module, torch.nn.Conv2d):
                last_conv = name
        return last_conv

    # Ініціалізація GradCAM
    def initialize_gradcam(model, cam_target_layer=None):
        if cam_target_layer is None:
            cam_target_layer = get_last_conv_layer(model)
            print(f"Використовується автоматично визначений цільовий шар для Grad-CAM: {cam_target_layer}")
        else:
            print(f"Використовується вручну заданий цільовий шар для Grad-CAM: {cam_target_layer}")

        # Створення екстрактора CAM
        cam_extractor = GradCAM(model, target_layer=cam_target_layer)
        return cam_extractordef normalize_cam(cam_map):
        cam_map = np.nan_to_num(cam_map, nan=0.0, posinf=1.0, neginf=0.0)  # Обробка нечислових значень
        cam_map -= cam_map.min()
        cam_map /= cam_map.max() if cam_map.max() > 0 else 1  # Нормалізація до [0, 1]
        return cam_mapdef generate_gradcam_visualization_matplotlib(model, input_tensor, cam_extractor, class_idx,
mean, std, device):
        model.eval()
        with torch.enable_grad():
            with torch.amp.autocast(device_type='cuda'):
                single_output = model(input_tensor.to(device))
        activation_map = cam_extractor(class_idx, single_output)
        if isinstance(activation_map, list):
            cam_map = activation_map[0].squeeze().cpu().numpy()
        else:
            cam_map = activation_map.squeeze().cpu().numpy()
        cam_map = normalize_cam(cam_map)

        # Візуалізація cam_map
        plt.figure(figsize=(6,6))
        plt.imshow(cam_map, cmap='jet')
        plt.colorbar()
        plt.title('Grad-CAM Map')
        plt.axis('off')
        plt.show()

        # Перевірка діапазону
```

```
        assert cam_map.min() >= 0 and cam_map.max() <= 1, "cam_map має виходити за межі [0, 1]"

        # Відновлення оригінального зображення
        img = input_tensor.cpu().squeeze().detach().numpy().transpose(1, 2, 0)
        img = std * img + mean
        img = np.clip(img, 0, 1)
        img_pil = Image.fromarray((img * 255).astype(np.uint8))

        # Створення теплової карти
        heatmap = plt.get_cmap('jet')(cam_map)[:, :, :3]

        # Накладання теплової карти
        overlay = heatmap * 0.4 + np.array(img_pil) / 255 * 0.6
        overlay = np.clip(overlay, 0, 1)

        # Перетворення у формат PIL.Image
        overlay_pil = Image.fromarray((overlay * 255).astype(np.uint8))

        return overlay_pil

    # Альтернативна функція для генерації Grad-CAM з використанням OpenCV
    def generate_gradcam_visualization_opencv_alt(model, input_tensor, cam_extractor, class_idx, class_names, mean,
std, device):
        # Забезпечення наявності градієнтів
        input_tensor.requires_grad = True

        # Встановлення моделі в режим eval()
        model.eval()
        with torch.enable_grad():
            with torch.amp.autocast(device_type='cuda'):
                single_output = model(input_tensor.to(device))

        # Генерація теплової карти Grad-CAM
        activation_map = cam_extractor(class_idx, single_output)
        if activation_map is None:
            print("Grad-CAM не може бути згенеровано.")
            return None

        # Обробка activation_map залежно від його типу
        if isinstance(activation_map, list):
            cam_map = activation_map[0].squeeze().cpu().numpy()
        else:
            cam_map = activation_map.squeeze().cpu().numpy()

        cam_map = normalize_cam(cam_map)  # Нормалізація теплової карти до [0, 1]

        # Перевірка діапазону значень
        assert cam_map.min() >= 0 and cam_map.max() <= 1, "cam_map має виходити за межі [0, 1]"

        # Відновлення оригінального зображення
        img = input_tensor.cpu().squeeze().detach().numpy().transpose(1, 2, 0)
        img = std * img + mean
        img = np.clip(img, 0, 1)
        img = (img * 255).astype(np.uint8)

        # Перетворення зображення для OpenCV
        img_cv = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

        # Застосування colormap до cam_map
        heatmap = cv2.applyColorMap(np.uint8(255 * cam_map), cv2.COLORMAP_JET)

        # Накладання теплової карти на зображення
        overlay = cv2.addWeighted(img_cv, 0.6, heatmap, 0.4, 0)
```

```
    # Перетворення назад у формат PIL для відображення
    overlay_pil = Image.fromarray(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))

    return overlay_pildef train_model(
    model,
    criterion,
    optimizer,
    scheduler,
    dataloaders,
    dataset_sizes,
    class_names,  # Додано: список назв класів
    num_epochs=30,  # Збільшено кількість епох до 30
    patience=7,      # Збільшено patience до 7 для раннього зупинення
    use_mixup=False,
    alpha=1.0,
    fold=1,
    cam_target_layer=None  # Встановлено на None для автоматічного визначення
):
    """
    Функція для навчання моделі з використанням раннього зупинення та Grad-CAM.

    Args:
        model (torch.nn.Module): Модель для навчання.
        criterion (torch.nn.Module): Функція втрат.
        optimizer (torch.optim.Optimizer): Оптимізатор.
        scheduler (torch.optim.lr_scheduler): Планувальник навчання.
        dataloaders (dict): Словник з DataLoader'ами для 'train' та 'val'.
        dataset_sizes (dict): Словник з розмірами наборів даних для 'train' та 'val'.
        class_names (list): Список назв класів.
        num_epochs (int): Максимальна кількість епох.
        patience (int): Кількість епох без покращення для раннього зупинення.
        use_mixup (bool): Використовувати Mixup аугментацію.
        alpha (float): Параметр для Mixup.
        fold (int): Номер фолду (для мультикрос-валідації).
        cam_target_layer (str): Цільовий шар для Grad-CAM.

    Returns:
        model (torch.nn.Module): Навчена модель.
        history (dict): Історія навчання.
    """
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_f1 = 0.0
    epochs_no_improve = 0
    history = {
        'train_loss': [],
        'train_acc': [],
        'train_precision': [],
        'train_recall': [],
        'train_f1': [],
        'val_loss': [],
        'val_acc': [],
        'val_precision': [],
        'val_recall': [],
        'val_f1': [],
        'val_balanced_acc': [],
        'val_roc_auc': []
    }

    writer = SummaryWriter(log_dir=os.path.join("models", "tensorboard_logs", f"fold_{fold}"))

    # Ініціалізація GradScaler для AMP з оновленим методом
```

```
scaler = torch.amp.GradScaler()

# Визначте середні значення та стандартні відхилення для відновлення зображень
mean = np.array([0.485, 0.456, 0.406])
std  = np.array([0.229, 0.224, 0.225])

# Автоматичне визначення цільового шару, якщо він не заданий
if cam_target_layer is None:
    cam_target_layer = get_last_conv_layer(model)
    if cam_target_layer is None:
        raise ValueError("Модель не містить конволюційних шарів для Grad-CAM.")
    print(f"Використовується автоматично визначений цільовий шар для Grad-CAM: {cam_target_layer}")
else:
    # Перевірка наявності шару у моделі
    if not any(name == cam_target_layer for name, _ in model.named_modules()):
        raise ValueError(f"Неможливо знайти підмодуль '{cam_target_layer}' у моделі.")
    print(f"Використовується цільовий шар для Grad-CAM: {cam_target_layer}")

# Ініціалізація GradCAM
cam_extractor = GradCAM(model, target_layer=cam_target_layer)

for epoch in range(num_epochs):
    print(f'--- Епоха {epoch + 1}/{num_epochs} ---')
    print('-' * 30)
    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
            print('Режим: Навчання')
        else:
            model.eval()
            print('Режим: Валідація')
        running_loss = 0.0
        running_corrects = 0
        all_preds = []
        all_labels = []
        all_probs = []
        all_train_preds = []
        all_train_labels = []
        for batch_idx, (inputs, labels) in enumerate(dataloaders[phase], 1):
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            if phase == 'train':
                with torch.set_grad_enabled(True):
                    with torch.amp.autocast(device_type='cuda'):
                        if use_mixup:
                            inputs, targets_a, targets_b, lam = mixup_data(inputs, labels, alpha)
                            outputs = model(inputs)
                            loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
                        else:
                            outputs = model(inputs)
                            loss = criterion(outputs, labels)
                    _, preds = torch.max(outputs, 1)
                scaler.scale(loss).backward()
                scaler.step(optimizer)
                scaler.update()
            else:
                with torch.set_grad_enabled(True):  # Дає змогумо градієнти для Grad-CAM
                    with torch.amp.autocast(device_type='cuda'):
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)
                        _, preds = torch.max(outputs, 1)
```

```
                    # Застосовуємо Grad-CAM для перших 5 зразків валідації
                    if batch_idx == 1:
                        num_samples = min(5, outputs.size(0))
                        selected_class_idx = preds[:num_samples].tolist()
                        selected_scores = outputs[:num_samples]

                        print(f"selected_class_idx length: {len(selected_class_idx)}, selected_scores batch size: {selected_scores.size(0)}")
                        print(f"selected_class_idx: {selected_class_idx}")
                        print(f"selected_scores shape: {selected_scores.shape}")
                        print(f"selected_scores dtype: {selected_scores.dtype}")
                        print(f"selected_scores requires_grad: {selected_scores.requires_grad}")

                    if batch_idx == 1:
                        for i in range(num_samples):
                            pred_class = selected_class_idx[i]
                            if not (0 <= pred_class < len(class_names)):
                                print(f"Некоректний class_idx: {pred_class} для зразка {i+1}")
                                continue

                            single_input = inputs[i].unsqueeze(0).detach().requires_grad_(True)

                            single_output = model(single_input)

                            # Генерація Grad-CAM карти
                            activation_map = cam_extractor(pred_class, single_output)
                            if activation_map is None:
                                print(f"Grad-CAM не може бути згенеровано для зразка {i+1}")
                                continue

                            # Перевірка типу activation_map
                            if isinstance(activation_map, list):
                                cam_map = activation_map[0].squeeze().cpu().numpy()
                            else:
                                cam_map = activation_map.squeeze().cpu().numpy()

                            cam_map = normalize_cam(cam_map)  # Нормалізація теплової карти

                            # Відновлення оригінального зображення
                            img = single_input.cpu().squeeze().detach().numpy().transpose(1, 2, 0)
                            img = std * img + mean
                            img = np.clip(img, 0, 1)
                            img_pil = Image.fromarray((img * 255).astype(np.uint8))

                            # Створення теплової карти
                            heatmap = plt.get_cmap('jet')(cam_map)[:, :, :3]

                            # Масштабування heatmap до розмірів оригінального зображення
                            heatmap_resized = np.array(
                                Image.fromarray((heatmap * 255).astype(np.uint8))
                                .resize(img_pil.size, resample=Image.BILINEAR)
                            ) / 255.0  # Нормалізуємо назад до [0, 1]

                            # Накладання теплової карти на оригінальне зображення
                            overlay = heatmap_resized * 0.4 + np.array(img_pil) / 255.0 * 0.6
                            overlay = np.clip(overlay, 0, 1)

                            # Перетворення у формат PIL.Image
                            overlay_pil = Image.fromarray((overlay * 255).astype(np.uint8))

                            # Відображення або збереження результату
                            plt.figure(figsize=(6,6))
                            plt.imshow(overlay_pil)
```

```
                        plt.axis('off')
                        plt.title(f'Grad-CAM для класу: {class_names[pred_class]}')
                        save_path = os.path.join("models", f"gradcam_fold_{fold}_epoch_{epoch+1}_sample_{i+1}.png")
                        plt.savefig(save_path)
                        plt.close()
                        print(f'Grad-CAM збережено за шляхом: {save_path}')

                # Збір статистики
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

                if phase == 'val':
                    probs = torch.softmax(outputs, dim=1)
                    all_probs.extend(probs.detach().cpu().numpy())

                if phase == 'train':
                    all_train_preds.extend(preds.cpu().numpy())
                    all_train_labels.extend(labels.cpu().numpy())

                if batch_idx % 10 == 0:
                    print(f'  Батч {batch_idx}/{len(dataloaders[phase])} | Втрата: {loss.item():.4f}')
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]
            history[f'{phase}_loss'].append(epoch_loss)
            history[f'{phase}_acc'].append(epoch_acc.cpu().numpy())
            print(f'{phase} Втрата: {epoch_loss:.4f} | Точність: {epoch_acc:.4f}')

            if phase == 'train':
                # Обчислення Precision, Recall, F1 для тренувального набору
                precision = precision_score(all_train_labels, all_train_preds, average='weighted', zero_division=0)
                recall = recall_score(all_train_labels, all_train_preds, average='weighted', zero_division=0)
                f1 = f1_score(all_train_labels, all_train_preds, average='weighted', zero_division=0)
                history['train_precision'].append(precision)
                history['train_recall'].append(recall)
                history['train_f1'].append(f1)
                print(f'{phase} Precision: {precision:.4f} | Recall: {recall:.4f} | F1-Score: {f1:.4f}')
                writer.add_scalar(f'{phase}/Precision', precision, epoch)
                writer.add_scalar(f'{phase}/Recall', recall, epoch)
                writer.add_scalar(f'{phase}/F1-Score', f1, epoch)

            if phase == 'val':
                precision = precision_score(all_labels, all_preds, average='weighted', zero_division=0)
                recall = recall_score(all_labels, all_preds, average='weighted', zero_division=0)
                f1 = f1_score(all_labels, all_preds, average='weighted', zero_division=0)
                history['val_precision'].append(precision)
                history['val_recall'].append(recall)
                history['val_f1'].append(f1)
                print(f'{phase} Precision: {precision:.4f} | Recall: {recall:.4f} | F1-Score: {f1:.4f}')
                writer.add_scalar(f'{phase}/Precision', precision, epoch)
                writer.add_scalar(f'{phase}/Recall', recall, epoch)
                writer.add_scalar(f'{phase}/F1-Score', f1, epoch)

                balanced_acc = balanced_accuracy_score(all_labels, all_preds)
                history['val_balanced_acc'].append(balanced_acc)
                print(f'{phase} Balanced Accuracy: {balanced_acc:.4f}')
                writer.add_scalar(f'{phase}/Balanced_Accuracy', balanced_acc, epoch)

                try:
                    roc_auc = roc_auc_score(
                        pd.get_dummies(all_labels).values,
```

```
                        np.array(all_probs),
                        average='macro',
                        multi_class='ovr'
                    )
                except ValueError:
                    roc_auc = float('nan')  # Випадок, коли не можливо обчислити ROC-AUC
                history['val_roc_auc'].append(roc_auc)
                print(f'{phase} ROC-AUC: {roc_auc:.4f}')
                writer.add_scalar(f'{phase}/ROC_AUC', roc_auc, epoch)

                conf_matrix = confusion_matrix(all_labels, all_preds)
                plt.figure(figsize=(10,8))
                sns.heatmap(conf_matrix, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names,
cmap='Blues')
                plt.ylabel('Істинні мітки')
                plt.xlabel('Передбачені мітки')
                plt.title(f'Confusion Matrix Fold {fold} Epoch {epoch+1}')
                confusion_matrix_path = os.path.join("models", f"confusion_matrix_fold_{fold}_epoch_{epoch+1}.png")
                plt.savefig(confusion_matrix_path)
                plt.close()
                print(f'Confusion Matrix збережено за шляхом: {confusion_matrix_path}')

                # Додавання Confusion Matrix до TensorBoard
                fig, ax = plt.subplots(figsize=(10,8))
                sns.heatmap(conf_matrix, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names,
cmap='Blues', ax=ax)
                ax.set_ylabel('Істинні мітки')
                ax.set_xlabel('Передбачені мітки')
                ax.set_title(f'Confusion Matrix Fold {fold} Epoch {epoch+1}')
                writer.add_figure(f'{phase}/Confusion_Matrix', fig, global_step=epoch)
                plt.close(fig)

            # Перевірка на раннє зупинення
            if phase == 'val':
                scheduler.step(epoch_loss)
                current_f1 = history['val_f1'][-1]
                if current_f1 > best_f1:
                    best_f1 = current_f1
                    best_model_wts = copy.deepcopy(model.state_dict())
                    epochs_no_improve = 0
                else:
                    epochs_no_improve += 1
                if epochs_no_improve >= patience:
                    print('Раннє зупинення!')

                    # Завантаження найкращої моделі
                    model.load_state_dict(best_model_wts)

                    # Завершення навчання
                    break
            print()

        # Обчислення часу навчання
        time_elapsed = time.time() - since
        print(f'Навчання завершено за {int(time_elapsed // 60)}хв {int(time_elapsed % 60)}с')
        print(f'Найкращий F1-Score на валідації: {best_f1:.4f}')
        model.load_state_dict(best_model_wts)
        writer.close()
        return model, history# Визначення гіперпараметрів
    learning_rate = 1e-4  # Встановіть відповідне значення
    weight_decay = 1e-5   # Встановіть відповідне значення
    num_epochs = 30       # Збільшено кількість епох до 30
    patience = 7          # Збільшено patience до 7
```

```
use_mixup = False
alpha = 1.0
batch_size = 30

# Вибір оптимізатора
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

# Створення scheduler'a без verbose=True
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=patience) from sklearn.model_selection import StratifiedKFold
from sklearn.utils.class_weight import compute_class_weight
from torch.utils.data import DataLoader, Subset
import torch.optim as optim
import torch.nn as nn
import json
import os
from collections import Counter
import numpy as np

n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

os.makedirs("models", exist_ok=True)

for fold, (train_idx, val_idx) in enumerate(skf.split(indices := list(range(len(full_dataset))), all_labels)):
    print(f'--- Фолд {fold + 1}/{n_splits} ---')

    # Поділ міток для поточного фолду
    train_labels_fold = [all_labels[i] for i in train_idx]
    val_labels_fold = [all_labels[i] for i in val_idx]
    print(f'Train класів: {Counter(train_labels_fold)}')
    print(f'Val класів: {Counter(val_labels_fold)}')

    # Створення піднаборів
    train_subset = Subset(full_dataset, train_idx)
    val_subset = Subset(full_dataset, val_idx)

    # Обчислення ваг класів для поточного фолду
    labels_array_fold = np.array(train_labels_fold)
    present_classes_fold = np.unique(labels_array_fold)
    class_weights_present_fold = compute_class_weight(class_weight='balanced', classes=present_classes_fold, y=labels_array_fold)
    class_weights_full_fold = np.zeros(len(classes))
    for cls in present_classes_fold:
        class_weights_full_fold[cls] = class_weights_present_fold[np.where(present_classes_fold == cls)[0][0]]
    print(f"Вага класів: {class_weights_full_fold}")

    class_weights_fold = torch.tensor(class_weights_full_fold, dtype=torch.float).to(device)
    criterion = nn.CrossEntropyLoss(weight=class_weights_fold)

    # Створення моделі
    model = create_model(num_classes=len(classes), architecture=architecture, device=device)

    # Виведення структури моделі для перевірки цільового шару
    print(model)

    # Вибір оптимізатора
    optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=weight_decay)

    # Створення scheduler'a без verbose=True
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=patience)

    # Створення DataLoader'ів
```

```
        dataloaders = {
            'train': DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True),  # num_workers=0 для Windows
            'val': DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=0, pin_memory=True)
        }

        dataset_sizes = {
            'train': len(train_subset),
            'val': len(val_subset)
        }

        # Автоматичне визначення цільового шару або встановлення вручну
        cam_target_layer = None  # Встановіть на None для автоматичного визначення, або задайте вручну, наприклад 'layer4.2.conv3'

        # Навчання моделі
        model, history = train_model(
            model=model,
            criterion=criterion,
            optimizer=optimizer,
            scheduler=scheduler,
            dataloaders=dataloaders,
            dataset_sizes=dataset_sizes,
            class_names=classes,  # Передача назв класів
            num_epochs=num_epochs,
            patience=patience,
            use_mixup=use_mixup,
            alpha=alpha,
            fold=fold + 1,  # Передача номера фолду
            cam_target_layer=cam_target_layer  # Вказано правильний цільовий шар або None для автоматичного визначення
        )

        # Збереження моделі
        model_path = os.path.join("models", f"model_fold_{fold + 1}.pth")
        torch.save(model.state_dict(), model_path)
        print(f"Модель збережена за шляхом: {model_path}\n")

        # Збереження історії навчання
        history_path = os.path.join("models", f"training_history_fold_{fold + 1}.json")
        with open(history_path, 'w') as f:
            json.dump(history, f, default=lambda o: o.tolist() if isinstance(o, np.ndarray) else o)
        print(f"Історія навчання збережена за шляхом: {history_path}\n")

        # Візуалізація історії навчання
        epochs_range = range(1, len(history['train_loss']) + 1)

        plt.figure(figsize=(12,5))

        # Втрата
        plt.subplot(1, 2, 1)
        plt.plot(epochs_range, history['train_loss'], 'bo-', label='Train Loss')
        plt.plot(epochs_range, history['val_loss'], 'ro-', label='Validation Loss')
        plt.xlabel('Епоха')
        plt.ylabel('Втрата')
        plt.title('Втрата за Епохи')
        plt.legend()

        # Точність
        plt.subplot(1, 2, 2)
        plt.plot(epochs_range, history['train_acc'], 'bo-', label='Train Accuracy')
        plt.plot(epochs_range, history['val_acc'], 'ro-', label='Validation Accuracy')
        plt.xlabel('Епоха')
```

```
plt.ylabel('Точність')
plt.title('Точність за Епохи')
plt.legend()

plt.tight_layout()
plt.show()

# Додаткові метрики
plt.figure(figsize=(24,5))

# Precision
plt.subplot(1, 6, 1)
plt.plot(epochs_range, history['train_precision'], 'go-', label='Train Precision')
plt.plot(epochs_range, history['val_precision'], 'bo-', label='Validation Precision')
plt.xlabel('Епоха')
plt.ylabel('Precision')
plt.title('Precision за Епохи')
plt.legend()

# Recall
plt.subplot(1, 6, 2)
plt.plot(epochs_range, history['train_recall'], 'mo-', label='Train Recall')
plt.plot(epochs_range, history['val_recall'], 'ro-', label='Validation Recall')
plt.xlabel('Епоха')
plt.ylabel('Recall')
plt.title('Recall за Епохи')
plt.legend()

# F1-Score
plt.subplot(1, 6, 3)
plt.plot(epochs_range, history['train_f1'], 'co-', label='Train F1-Score')
plt.plot(epochs_range, history['val_f1'], 'ko-', label='Validation F1-Score')
plt.xlabel('Епоха')
plt.ylabel('F1-Score')
plt.title('F1-Score за Епохи')
plt.legend()

# Balanced Accuracy
plt.subplot(1, 6, 4)
plt.plot(epochs_range, history['val_balanced_acc'], 'bo-', label='Validation Balanced Accuracy')
plt.xlabel('Епоха')
plt.ylabel('Balanced Accuracy')
plt.title('Balanced Accuracy за Епохи')
plt.legend()

# ROC-AUC
plt.subplot(1, 6, 5)
plt.plot(epochs_range, history['val_roc_auc'], 'ro-', label='Validation ROC-AUC')
plt.xlabel('Епоха')
plt.ylabel('ROC-AUC')
plt.title('ROC-AUC за Епохи')
plt.legend()

# Повтор Balanced Accuracy (можна замінити на іншу метрику)
plt.subplot(1, 6, 6)
plt.plot(epochs_range, history['val_balanced_acc'], 'ro-', label='Validation Balanced Accuracy')
plt.xlabel('Епоха')
plt.ylabel('Balanced Accuracy')
plt.title('Balanced Accuracy за Епохи')
plt.legend()

plt.tight_layout()
plt.show()
```

```
    # Очищення кешу GPU
    torch.cuda.empty_cache()import torch
import timm
import torch.nn as nn

# Визначення пристрою
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Використовується пристрій: {device}")

# Функція для створення моделі (повторюється з попередньої комірки)
def create_model(num_classes, architecture='resnet50', device='cpu'):
    if architecture == 'resnet34':
        model = timm.create_model('resnet34', pretrained=True, in_chans=3, num_classes=num_classes)
    elif architecture == 'resnet50':
        model = timm.create_model('resnet50', pretrained=True, in_chans=3, num_classes=num_classes)
    elif architecture == 'efficientnet_b0':
        model = timm.create_model('efficientnet_b0', pretrained=True, in_chans=3, num_classes=num_classes)
    else:
        raise ValueError(f"Архітектура {architecture} не підтримується.")

    # Додавання Dropout перед класифікатором
    if hasattr(model, 'classifier'):
        num_ftrs = model.classifier.in_features
        model.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(num_ftrs, num_classes)
        )
    elif hasattr(model, 'fc'):
        num_ftrs = model.fc.in_features
        model.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(num_ftrs, num_classes)
        )
    else:
        raise AttributeError("Модель не має атрибуту 'classifier' або 'fc'.")

    return model.to(device)

# Припустимо, що найкраща модель вже збережена
best_model_path = "models/model_fold_1.pth"  # Замініть на шлях до вашої моделі
architecture = 'resnet50'
num_classes = len(classes)

model = create_model(num_classes=num_classes, architecture=architecture, device=device)
model.load_state_dict(torch.load(best_model_path, map_location=device))
model.eval()  # Встановлення моделі в режим оцінки
print("Модель завантажена та готова до використання.") import h5py
from torchvision import transforms
from torch.utils.data import DataLoader, Subset

# Функція для завантаження зображення з HDF5 файлу
def load_image_from_h5(file_path):
    with h5py.File(file_path, 'r') as f:
        if 'image' in f:
            image = f['image'][:]
        else:
            raise ValueError(f"Ключ 'image' не знайдено у файлі: {file_path}")

    # Обробка формату зображення
    if image.ndim == 2:
        # Градієнти сірого (Height, Width), конвертуємо до RGB
        image = np.stack([image] * 3, axis=-1)
```

```
        elif image.ndim == 3:
            # Обробка channel-first та channel-last форматів
            if image.shape[0] <= 4:
                # Channel-first
                if image.shape[0] > 3:
                    image = image[:3, :, :]
                image = np.transpose(image, (1, 2, 0))
            else:
                # Channel-last
                if image.shape[2] > 3:
                    image = image[:, :, :3]
        else:
            raise ValueError(f"Непідтримувана форма зображення: {image.shape}")

        # Перевірка кількості каналів після оброблення
        if image.shape[2] == 1:
            image = np.concatenate([image] * 3, axis=2)
        elif image.shape[2] != 3:
            raise ValueError(f"Непідтримувана кількість каналів після оброблення: {image.shape[2]}")

        # Конвертація до uint8
        if image.dtype != np.uint8:
            if image.max() <= 1.0:
                image = (image * 255).astype(np.uint8)
            else:
                image = image.astype(np.uint8)

        # Конвертація до PIL.Image
        image_pil = Image.fromarray(image)

        return image_pil

    # Підготовка трансформацій для SHAP (без нормалізації)
    shap_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.Lambda(lambda img: ensure_rgb(img)),
        transforms.ToTensor(),
    ])

    # Вибір кількох зразків для пояснення
    num_shap_samples = 10  # Кількість зразків для SHAP (можна змінити)
    shap_sample_indices = np.random.choice(len(full_dataset), num_shap_samples, replace=False)
    shap_subset = Subset(full_dataset, shap_sample_indices)

    # Створення DataLoader для SHAP
    shap_loader = DataLoader(shap_subset, batch_size=1, shuffle=False, num_workers=0, pin_memory=True) import
shap

    # Встановлення режим evaluation для SHAP
    model.eval()

    # Визначення функції, яка приймає батч зображень та повертає передбачення моделі
    def model_predict(x):
        """
        Функція для передбачення моделі.

        Args:
            x (numpy.ndarray): Масив зображень у форматі (n_samples, H, W, C).

        Returns:
            numpy.ndarray: Масив передбачень у форматі (n_samples, n_classes).
        """
        # Конвертація NumPy масиву в Torch тензор
```

```
    x_tensor = torch.tensor(x).permute(0, 3, 1, 2).float()  # Перестановка осей для Torch (C, H, W)

    # Застосування нормалізації (така ж, як у тренувальних даних)
    x_tensor = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                    std=[0.229, 0.224, 0.225])(x_tensor)

    x_tensor = x_tensor.to(device)

    # Передбачення з використанням моделі
    with torch.no_grad():
        outputs = model(x_tensor)

    # Конвертація передбачень назад у NumPy масив
    return outputs.cpu().numpy()



print("SHAP готовий до використання.") !pip install --upgrade shap scikit-image torch torchvision timm torchcam
gradio

import shap
from shap.maskers import Image as ShapImageMasker
from torch.utils.data import DataLoader, Subset
import numpy as np
import os
import copy
import torch
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt
import timm
import torch.nn as nn
import gradio as gr
from io import BytesIO
import traceback
from torchcam.methods import GradCAM

# Припускаємо, що full_dataset вже визначений
# Наприклад:
# from torchvision.datasets import ImageFolder
# full_dataset = ImageFolder(root='data/train', transform=transforms.Compose([
#     transforms.Resize((224, 224)),
#     transforms.ToTensor(),
# ]))

# Функція прогнозу для SHAP
def model_predict(x):
    """
    Функція прогнозу, яка перетворює вхідні NumPy масиви у PyTorch тензори,
    пропускає через модель та повертає ймовірності класів у форматі NumPy.
    """
    # Перевірка форми вхідного масиву
    if x.ndim != 4 or x.shape[-1] != 3:
        raise ValueError(f"Очікується масив форми [N, H, W, 3], але отримано {x.shape}")

    # Перетворення NumPy масиву у PyTorch тензор та перенесення на пристрій
    x_tensor = torch.tensor(x, dtype=torch.float32).permute(0, 3, 1, 2).to(device)  # [N, C, H, W]

    # Використання основної моделі без torch.no_grad() для SHAP
    outputs = model(x_tensor)  # Використовуємо основну модель
    probabilities = torch.softmax(outputs, dim=1)  # Перетворення логітів у ймовірності

    # Перетворення виходу у NumPy масив з використанням detach()
```

```
        return probabilities.detach().cpu().numpy()

    # Вибір фону для SHAP (background)
    background_size = 100
    background_indices = np.random.choice(len(full_dataset), background_size, replace=False)
    background_subset = Subset(full_dataset, background_indices)
    background_loader = DataLoader(background_subset, batch_size=background_size, shuffle=False, num_workers=0,
pin_memory=True)

    # Завантаження фону
    for background_batch, _ in background_loader:
        # Перетворення тензора в numpy масив та змінення порядку осей для SHAP
        background = background_batch.cpu().numpy()
        background = np.transpose(background, (0, 2, 3, 1))  # (batch_size, H, W, C)

        # Переконайтесь, що дані в форматі float32 та нормалізовані
        if background.dtype != np.float32:
            background = background.astype(np.float32)

        # Якщо ваші дані вже нормалізовані (0-1), пропустіть наступний рядок
        # Якщо ні, нормалізуйте їх відповідно
        # background /= 255.0  # Розкоментуйте, якщо потрібно

        break  # Завантажуємо лише перший батч

    # Визначення форми зображень (H, W, C)
    image_shape = background.shape[1:]  # (H, W, C)

    # Перевірка форми та типу даних
    print(f"Форма фону: {background.shape}")
    print(f"Тип даних фону: {background.dtype}")

    # Вибір masker для зображень з вказаною формою
    # Використання числового значення 0 для mask_value
    masker = ShapImageMasker(0, shape=image_shape)

    # Створення SHAP Explainer
    explainer = shap.Explainer(model_predict, masker=masker, data=background)

    print("SHAP Explainer створено.") import shap
    import copy
    from torch.utils.data import Subset, DataLoader
    import numpy as np
    import torch
    from torchvision import transforms

    # Середнє та стандартне відхилення для денормалізації (не потрібні для SHAP)
    mean = np.array([0.485, 0.456, 0.406])
    std  = np.array([0.229, 0.224, 0.225])

    # Функція прогнозу для SHAP (повторюється з комірки 20, можна уникнути дублювання)
    def model_predict(x):
        """
        Функція прогнозу, яка перетворює вхідні NumPy масиви у PyTorch тензори,
        пропускає через модель та повертає ймовірності класів у форматі NumPy.
        """
        # Перевірка форми вхідного масиву
        if x.ndim != 4 or x.shape[-1] != 3:
            raise ValueError(f"Очікується масив форми [N, H, W, 3], але отримано {x.shape}")

        # Перетворення NumPy масиву у PyTorch тензор та перенесення на пристрій
        x_tensor = torch.tensor(x, dtype=torch.float32).permute(0, 3, 1, 2).to(device)  # [N, C, H, W]
```

```
            # Використання основної моделі без torch.no_grad() для SHAP
            outputs = model(x_tensor)  # Використовуємо основну модель
            probabilities = torch.softmax(outputs, dim=1)  # Перетворення логітів у ймовірності

            # Перетворення виходу у NumPy масив з використанням detach()
            return probabilities.detach().cpu().numpy()

        # Створення копії моделі на CPU для SHAP
        shap_model = copy.deepcopy(model).to('cpu')

        # Підготовка фонового набору даних (background) без нормалізації
        background_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),  # Без Normalization
        ])

        # Припустимо, ваш Dataset називається `full_dataset`
        background_dataset = copy.deepcopy(full_dataset)
        background_dataset.transform = background_transform  # Замінюємо трансформації

        num_background = 100
        background = []
        attempts = 0
        max_attempts = 10  # Максимальна кількість спроб для збору необхідної кількості фоновых зображень

        while len(background) < num_background and attempts < max_attempts:
            background_indices = np.random.choice(len(background_dataset), num_background * 2, replace=False)
            background_subset = Subset(background_dataset, background_indices)
            background_loader = DataLoader(background_subset, batch_size=num_background * 2, shuffle=False,
num_workers=0, pin_memory=True)

            for background_batch, _ in background_loader:
                batch_images = background_batch.to('cpu')       # Переносимо на CPU
                batch_images = batch_images.detach().cpu().numpy().transpose(0, 2, 3, 1)  # Перетворюємо у (N, H, W, C)
                batch_images = [img.copy() for img in batch_images]       # Перетворюємо у список окремих зображень

                # Фільтрація зображень з варіацією (img.max() > 0 та більше одного унікального значення)
                valid_images = [img for img in batch_images if img.max() > 0 and len(np.unique(img)) > 1]

                # Додаємо до фонового набору
                background.extend(valid_images)

                # Перериваємо, якщо зібрали достатню кількість
                if len(background) >= num_background:
                    background = background[:num_background]
                    break

            attempts += 1

        if len(background) < num_background:
            raise ValueError(f"Не вдалося зібрати {num_background} валідних фоновых зображень після {max_attempts}
спроб.")

        # Перетворення фону на float32 (рекомендовано для SHAP)
        background = [img.astype(np.float32) for img in background]

        # Конвертуємо список фоновых зображень у NumPy масив
        background = np.array(background)

        # Убедитись, що значення у діапазоні [0,1]
        background = np.clip(background, 0, 1)

        # Додаткові перевірки
```

```
        assert background.shape == (num_background, 224, 224, 3), f"Неправильна форма фону: {background.shape}"
        assert background.dtype == np.float32, f"Неправильний тип даних фону: {background.dtype}"
        assert not np.isnan(background).any(), "Фоновий набір містить NaN значення."
        assert not np.isinf(background).any(), "Фоновий набір містить Inf значення."

        # Перевірка форми та типу даних
        print(f"Форма фону (першого зображення): {background[0].shape}")
        print(f"Тип даних фону: {background.dtype}")

        # Перевірка мінімальних та максимальних значень після нормалізації
        for i, img in enumerate(background):
            print(f"Фон {i}: min={img.min()}, max={img.max()}, dtype={img.dtype}")
            # Пом'якшена асертація
            assert img.min() >= 0 and img.max() <= 1.05, f"Фон {i} має некоректні значення: min={img.min()}, max={img.max()}"

        # Вибір masker для зображень з вказаною формою
        # Використання числового значення 0 для mask_value
        masker = ShapImageMasker(0, shape=background.shape[1:])

        # Створення SHAP Explainer з використанням ShapImageMasker
        explainer = shap.Explainer(model_predict, masker=masker, data=background)

        print("SHAP Explainer створено.") import torch
        import numpy as np
        from PIL import Image as PILImage  # Аліас для PIL.Image
        import matplotlib.pyplot as plt
        from torchvision import transforms
        import timm
        import torch.nn as nn
        import shap
        import gradio as gr
        from io import BytesIO
        import traceback
        import copy
        from torch.utils.data import Subset, DataLoader
        from torchvision.datasets import ImageFolder
        from shap.maskers import Image as SHAPImageMasker  # Аліас для shap.maskers.Image

        # Визначення пристрою
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print(f"Використовується пристрій: {device}")

        # Список класів
        classes = ['Normal', 'Hypertrophy', 'Dilation', 'HeartFailure']  # Замініть на ваші класи

        # Функція для створення моделі (як у вашому коді)
        def create_model(num_classes, architecture='resnet50', device='cpu'):
            if architecture == 'resnet50':
                model = timm.create_model('resnet50', pretrained=False, in_chans=3, num_classes=num_classes)
            else:
                raise ValueError(f"Архітектура {architecture} не підтримується.")

            # Додавання Dropout перед класифікатором
            if hasattr(model, 'classifier'):
                num_ftrs = model.classifier.in_features
                model.classifier = nn.Sequential(
                    nn.Dropout(0.5),
                    nn.Linear(num_ftrs, num_classes)
                )
            elif hasattr(model, 'fc'):
                num_ftrs = model.fc.in_features
                model.fc = nn.Sequential(
```

```
            nn.Dropout(0.5),
            nn.Linear(num_ftrs, num_classes)
        )
    else:
        raise AttributeError("Модель не має атрибуту 'classifier' або 'fc'.")

    return model.to(device)

# Завантаження моделі
best_model_path = "models/model_fold_1.pth"  # Замініть на шлях до вашої моделі
architecture = 'resnet50'
num_classes = len(classes)
model = create_model(num_classes=num_classes, architecture=architecture, device=device)

# Завантаження вагів моделі
try:
    model.load_state_dict(torch.load(best_model_path, map_location=device, weights_only=True), strict=False)
except TypeError:
    model.load_state_dict(torch.load(best_model_path, map_location=device), strict=False)

model.eval()
print("Модель завантажена та готова до використання.")

# Переконайтеся, що всі параметри моделі можуть отримувати градієнти
for param in model.parameters():
    param.requires_grad = True

# Визначення цільового шару для Grad-CAM
def get_last_conv_layer(model):
    # Для ResNet50 останній конволюційний шар - 'layer4'
    return 'layer4'

# Ініціалізація GradCAM
from torchcam.methods import GradCAM

target_layer_name = get_last_conv_layer(model)
print(f"Цільовий шар для Grad-CAM: {target_layer_name}")
cam_extractor = GradCAM(model, target_layer=target_layer_name)

# Середнє та стандартне відхилення для нормалізації
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])

# Функція прогнозу для SHAP
def model_predict(x):
    """
    Функція прогнозу, яка перетворює вхідні NumPy масиви у PyTorch тензори,
    пропускає через модель та повертає ймовірності класів у форматі NumPy.
    """
    # Перевірка форми вхідного масиву
    if x.ndim != 4 or x.shape[-1] != 3:
        raise ValueError(f"Очікується масив форми [N, H, W, 3], але отримано {x.shape}")

    # Перетворення NumPy масиву у PyTorch тензор та перенесення на CPU
    x_tensor = torch.tensor(x, dtype=torch.float32).permute(0, 3, 1, 2).to('cpu')  # [N, C, H, W]

    # Використання основної моделі без torch.no_grad() для SHAP
    outputs = shap_model(x_tensor)  # Використовуємо копію моделі на CPU
    probabilities = torch.softmax(outputs, dim=1)  # Перетворення логітів у ймовірності

    # Перетворення виходу у NumPy масив з використанням detach()
    return probabilities.detach().cpu().numpy()
```

```
# Створення копії моделі на CPU для SHAP
shap_model = copy.deepcopy(model).to('cpu')

# Припускаємо, що full_dataset вже визначений
# Наприклад:
# full_dataset = ImageFolder(root='data/train', transform=transforms.Compose([
#     transforms.Resize((224, 224)),
#     transforms.ToTensor(),
# ]))

# Переконайтесь, що `full_dataset` визначений
try:
    full_dataset
except NameError:
    raise NameError("Переконайтесь, що `full_dataset` визначений перед запуском цього коду.")

# Підготовка фонового набору даних (background) без нормалізації
background_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),  # Без Normalization
])

# Припустимо, ваш Dataset називається `full_dataset`
background_dataset = copy.deepcopy(full_dataset)
background_dataset.transform = background_transform  # Замінюємо трансформації

# Вибір фону для SHAP (background)
num_background = 100
background = []
attempts = 0
max_attempts = 10  # Максимальна кількість спроб для збору необхідної кількості фонових зображень

while len(background) < num_background and attempts < max_attempts:
    background_indices = np.random.choice(len(background_dataset), num_background * 2, replace=False)
    background_subset = Subset(background_dataset, background_indices)
    background_loader = DataLoader(background_subset, batch_size=num_background * 2, shuffle=False,
num_workers=0, pin_memory=True)

    for background_batch, _ in background_loader:
        batch_images = background_batch.to('cpu')       # Переносимо на CPU
        batch_images = batch_images.detach().cpu().numpy().transpose(0, 2, 3, 1)  # Перетворюємо у (N, H, W, C)
        batch_images = [img.copy() for img in batch_images]       # Перетворюємо у список окремих зображень

        # Фільтрація зображень з варіацією (img.max() > 0 та більше одного унікального значення)
        valid_images = [img for img in batch_images if img.max() > 0 and len(np.unique(img)) > 1]

        # Додаємо до фонового набору
        background.extend(valid_images)

        # Перериваємо, якщо зібрали достатню кількість
        if len(background) >= num_background:
            background = background[:num_background]
            break

    attempts += 1

if len(background) < num_background:
    raise ValueError(f"Не вдалося зібрати {num_background} валідних фонових зображень після {max_attempts}
спроб.")

# Перетворення фону на float32 та нормалізація
background = [
    transforms.Normalize(mean, std)(torch.tensor(img).permute(2, 0, 1)).numpy().transpose(1, 2, 0)
```

```
        for img in background
    ]
    background = np.array(background)

    # Убедитись, що значення у діапазоні [0,1]
    background = np.clip(background, 0, 1)

    # Додаткові перевірки
    assert background.shape == (num_background, 224, 224, 3), f"Неправильна форма фону: {background.shape}"
    assert background.dtype == np.float32, f"Неправильний тип даних фону: {background.dtype}"
    assert not np.isnan(background).any(), "Фоновий набір містить NaN значення."
    assert not np.isinf(background).any(), "Фоновий набір містить Inf значення."

    # Перевірка форми та типу даних
    print(f"Форма фону (першого зображення): {background[0].shape}")
    print(f"Тип даних фону: {background.dtype}")

    # Перевірка мінімальних та максимальних значень після нормалізації
    for i, img in enumerate(background):
        print(f"Фон {i}: min={img.min()}, max={img.max()}, dtype={img.dtype}")
        # Пом'якшена асертація
        assert img.min() >= -2 and img.max() <= 2.64, f"Фон {i} має некоректні значення: min={img.min()}, max={img.max()}"

    # Вибір masker для зображень з вказаною формою
    # Використання маскування з відповідними середніми значеннями
    mask_value = 0.0  # Скалярне значення для маскування
    masker = SHAPImageMasker(mask_value, shape=(224, 224, 3))  # Використовуємо клас SHAPImageMasker з правильним форматом

    # Створення SHAP Explainer з використанням SHAPImageMasker
    explainer = shap.Explainer(model_predict, masker=masker, data=background)

    print("SHAP Explainer створено.")

    # Функція для генерації Grad-CAM накладання
    def generate_gradcam_overlay(model, cam_extractor, image_tensor_normalized, predicted_class, mean, std, device):
        """
        Генерує Grad-CAM накладання для зображення.
        """
        # Забезпечення режиму оцінки
        model.eval()

        # Перенесення тензора на пристрій та встановлення requires_grad=True
        image_tensor_normalized = image_tensor_normalized.to(device)
        image_tensor_normalized.requires_grad_()

        # Генерація виходу моделі
        output = model(image_tensor_normalized)

        # Генерація Grad-CAM карти
        activation_map = cam_extractor(predicted_class, output)
        if isinstance(activation_map, list):
            cam_map = activation_map[0].cpu().numpy()
        else:
            cam_map = activation_map.cpu().numpy()

        cam_map = np.squeeze(cam_map)
        cam_map = (cam_map - cam_map.min()) / (cam_map.max() - cam_map.min() + 1e-10)  # Нормалізація

        # Перетворення зображення в NumPy для візуалізації
        image_np = image_tensor_normalized.cpu().detach().numpy().squeeze().transpose(1, 2, 0)  # [H, W, C]
        image_np = std * image_np + mean  # Денормалізація
```

```
            image_np = np.clip(image_np, 0, 1)

            # Масштабування Grad-CAM карти до розміру зображення
            cam_map_resized = np.array(
                PILImage.fromarray((cam_map * 255).astype(np.uint8)).resize((image_np.shape[1], image_np.shape[0]),
resample=PILImage.BILINEAR)
            ) / 255.0  # Повернення до діапазону [0,1]

            # Створення теплової карти
            heatmap = plt.get_cmap('jet')(cam_map_resized)[:, :, :3]

            # Накладання Grad-CAM на зображення
            overlay = heatmap * 0.4 + image_np * 0.6
            overlay = np.clip(overlay, 0, 1)

            # Перетворення у формат PIL.Image
            overlay_pil = PILImage.fromarray((overlay * 255).astype(np.uint8))
            return overlay_pil

        # Функція для генерації текстового пояснення для SHAP
        def generate_shap_text(shap_values, image_input, predicted_class):
            """
            Генерує текстове пояснення на основі SHAP значень.
            """
            # Отримуємо SHAP значення для передбаченого класу
            shap_class = shap_values[0].values[predicted_class]

            # Перевірка розміру shap_class
            if shap_class.ndim != 3:
                raise ValueError(f"Очікується shap_class з 3 вимірами, але отримано {shap_class.ndim}")

            # Розраховуємо середнє абсолютне значення SHAP по каналах
            shap_abs = np.abs(shap_class).mean(axis=(1, 2))  # axis=(1,2) для H і W
            top_channels = np.argsort(shap_abs)[::-1][:3]  # Топ-3 каналів

            # Створюємо текстове пояснення
            shap_text = f"SHAP пояснення для класу '{classes[predicted_class]}':\n"
            shap_text += f"Основні канали, що вплинули на передбачення:\n"
            for i, channel in enumerate(top_channels):
                shap_text += f"{i+1}. Канал {channel + 1} зі середнім абсолютним SHAP значенням
{shap_abs[channel]:.4f}\n"

            return shap_text

        # Функція для оброблення зображення та генерації пояснень
        def process_image(uploaded_image):
            try:
                # Перетворення завантаженого зображення до потрібного формату
                uploaded_image = ensure_rgb(uploaded_image)

                # Попередня обробка зображення
                transform_pipeline = transforms.Compose([
                    transforms.Resize((224, 224)),
                    transforms.ToTensor(),
                ])
                image_tensor = transform_pipeline(uploaded_image).unsqueeze(0).to(device)  # [1, C, 224, 224]

                # Нормалізація вхідного зображення
                image_tensor_normalized = normalize_image(image_tensor)

                # Перестановка розмірностей перед передачею до model_predict
                image_np = image_tensor_normalized.detach().cpu().permute(0, 2, 3, 1).numpy()  # [1, H, W, C]
```

```
        # Отримання передбачення та передбаченого класу
        probabilities = model_predict(image_np)
        predicted_class = int(np.argmax(probabilities, axis=1)[0].item())  # Перетворення у int
        predicted_class_name = classes[predicted_class]
        print(f"Передбачений клас: {predicted_class_name}")
        print(f"Тип predicted_class: {type(predicted_class)}")  # Додавання перевірки типу

        # Генерація Grad-CAM накладання
        gradcam_overlay = generate_gradcam_overlay(
            model=model,
            cam_extractor=cam_extractor,
            image_tensor_normalized=image_tensor_normalized,
            predicted_class=predicted_class,
            mean=mean,
            std=std,
            device=device
        )

        # Генерація SHAP пояснення для передбаченого класу
        # Підготовка вхідних даних для SHAP
        image_input = image_tensor_normalized.detach().cpu().numpy()  # [1, C, H, W]

        # Перетворення у формат (N, H, W, C) для SHAP
        image_input = image_input.transpose(0, 2, 3, 1)  # [1, H, W, C]

        # Генерація SHAP значень
        shap_values = explainer(image_input)
        print("SHAP values:", shap_values[0].values[predicted_class])

        # Відображення SHAP пояснення та збереження в буфер
        shap_image_pil = shap.image_plot(shap_values, image_input, show=False)
        buf = BytesIO()
        plt.savefig(buf, format='png', bbox_inches='tight', pad_inches=0)
        plt.close()
        buf.seek(0)
        shap_image_pil = PILImage.open(buf)

        # Генерація текстового пояснення для SHAP
        shap_text = generate_shap_text(shap_values, image_input, predicted_class)

        # Повертаємо результати
        return predicted_class_name, gradcam_overlay, shap_image_pil, shap_text

    except Exception as e:
        # Виведення інформації про помилку в інтерфейс
        error_message = traceback.format_exc()
        print("Помилка у функції process_image:")
        print(error_message)
        return f"Сталася помилка:\n{error_message}", None, None, None

# Функція для забезпечення RGB
def ensure_rgb(image):
    if image.mode != 'RGB':
        return image.convert('RGB')
    return image

# Функція для нормалізації зображення (так само, як при навчанні)
def normalize_image(tensor):
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    return normalize(tensor)

# Створення інтерфейсу Gradio з українськими підписами та текстовим поясненням SHAP
```

```
with gr.Blocks() as iface:
    # HTML з JavaScript для зміни тексту drag-and-drop
    gr.HTML("""
    <script>
    document.addEventListener("DOMContentLoaded", function() {
        // Функція для заміни тексту в зоні drag-and-drop
        function replaceDragDropText() {
            const elements = document.querySelectorAll('div');
            elements.forEach(el => {
                if (el.innerText.includes('Drag and drop') || el.innerText.includes('Перетащите изображение')) {
                    el.innerText = 'Перетягніть зображення сюди або натисніть, щоб завантажити';
                }
            });
        }

        // Виклик функції після невеликої затримки, щоб впевнитися, що елементи завантажилися
        setTimeout(replaceDragDropText, 1000);
    });
    </script>
    """)

    # Заголовок
    gr.Markdown("# Класифікація МРТ та Пояснення")

    # Опис
    gr.Markdown("Завантажте зображення МРТ, щоб отримати передбачений клас, Grad-CAM накладання та SHAP пояснення.")

    # Загрузка зображення
    with gr.Row():
        image_input = gr.Image(type="pil", label="Завантажте зображення")

    # Кнопка для запуску аналізу
    with gr.Row():
        analyze_button = gr.Button("Аналізувати")

    # Виводи
    with gr.Row():
        predicted_class_output = gr.Textbox(label="Передбачений клас")
        gradcam_output = gr.Image(type="pil", label="Grad-CAM накладання")

    with gr.Row():
        shap_image_output = gr.Image(type="pil", label="SHAP пояснення")
        shap_text_output = gr.Textbox(label="Текстове пояснення SHAP")

    # Визначення функції запуску
    analyze_button.click(
        fn=process_image,
        inputs=image_input,
        outputs=[predicted_class_output, gradcam_output, shap_image_output, shap_text_output]
    )

# Запуск додатку
iface.launch(debug=True)
```

Додаток В

Презентаційний матеріал

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

МЕТОД ІНТЕРПРЕТУВАННЯ РЕЗУЛЬТАТІВ ВИЯВЛЕННЯ ПАТОЛОГІЙ СЕРЦЯ НА ЗОБРАЖЕННЯХ МРТ

Виконав:
студент 2 курсу, групи КНм-19-1
Діхтяр Максим Олександрович

Керівник:
старший викладач кафедри КН
Радюк Павло Михайлович

2

Актуальність

- Кваліфікаційна робота магістра присвячена підвищенню рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання, через удосконалення методу інтерпретування результатів виявлення патологій серця у вигляді модуля програмного забезпечення.
- Актуальність дослідження зумовлена зростанням обсягів медичних МРТ-зображень серця, які створюють виклики для їхнього своєчасного та точного аналізу. Хоча методи глибокого навчання демонструють високу ефективність у виявленні патологій, недостатня прозорість цих моделей обмежує їхнє широке використання в медичній практиці. З огляду на це, актуальним є проєктування методів інтерпретації результатів виявлення патологій серця на МРТ-зображеннях, який би поєднував точність нейронних мереж з можливістю пояснення їхніх рішень.

3

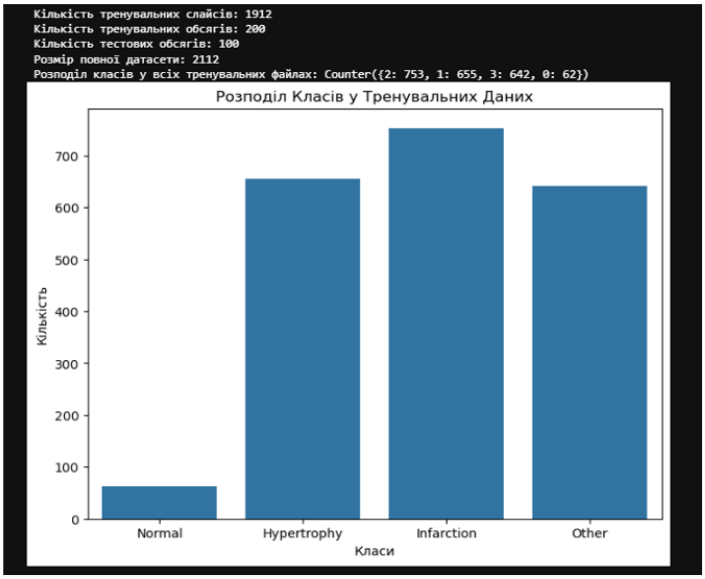
Мета і задачі роботи

- Мета кваліфікаційної роботи магістра:
- Мета роботи полягає у підвищенні рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.
- **Об'єкт дослідження:**
  Процес інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання.
- **Предмет дослідження:**
  Моделі, методи та технології глибокого навчання для аналізу зображень МРТ.
- **Для досягнення поставленої мети необхідно виконати наступні завдання:**
- 1. Провести аналіз моделей, методів та технологій глибокого навчання для оброблення та аналізу зображень МРТ.
- 2. Вдосконалити метод виявлення візуальних патологій серця за зображенням МРТ з використанням згорткової нейронної мережі.
- 3. Спроєктувати метод інтерпретування результатів виявлення візуальних патологій серця за зображенням МРТ, отриманих від згорткової нейронної мережі.
- 4. Реалізувати метод інтерпретування у вигляді модуля програмного забезпечення.
- 5. Провести експериментальне тестування реалізованого модуля за еталонними наборами даних.

4

Схема покращеного методу інтерпретації результатів виявлення патологій на зображеннях МРТ

5

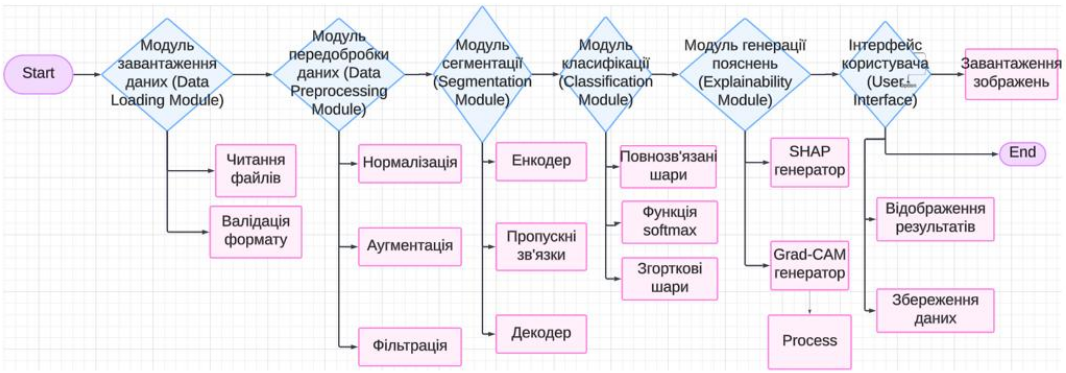


Розподіл класів у тренувальних даних ACDC

(Automated Cardiac Diagnosis Challenge)

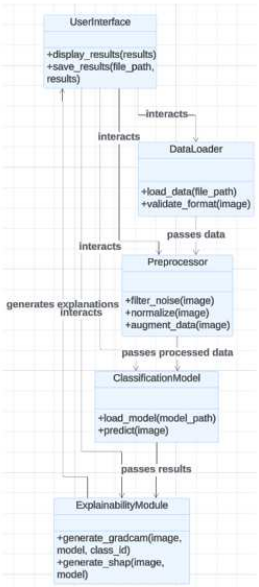
6

Схема функціональної структури інформаційної системи



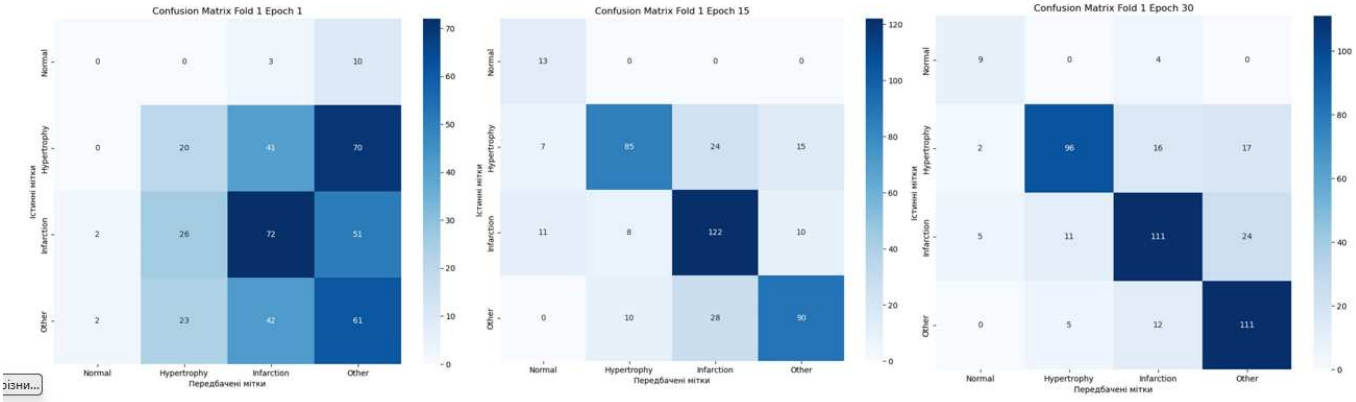
7

Діаграма класів аналізу та пояснення медичних зображень



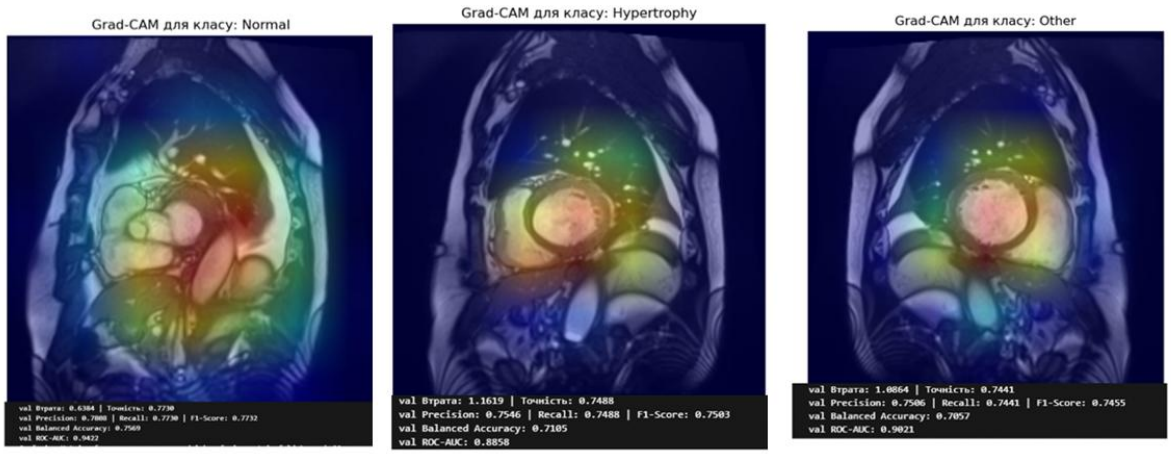
8

Матриці плутанини для різних епох



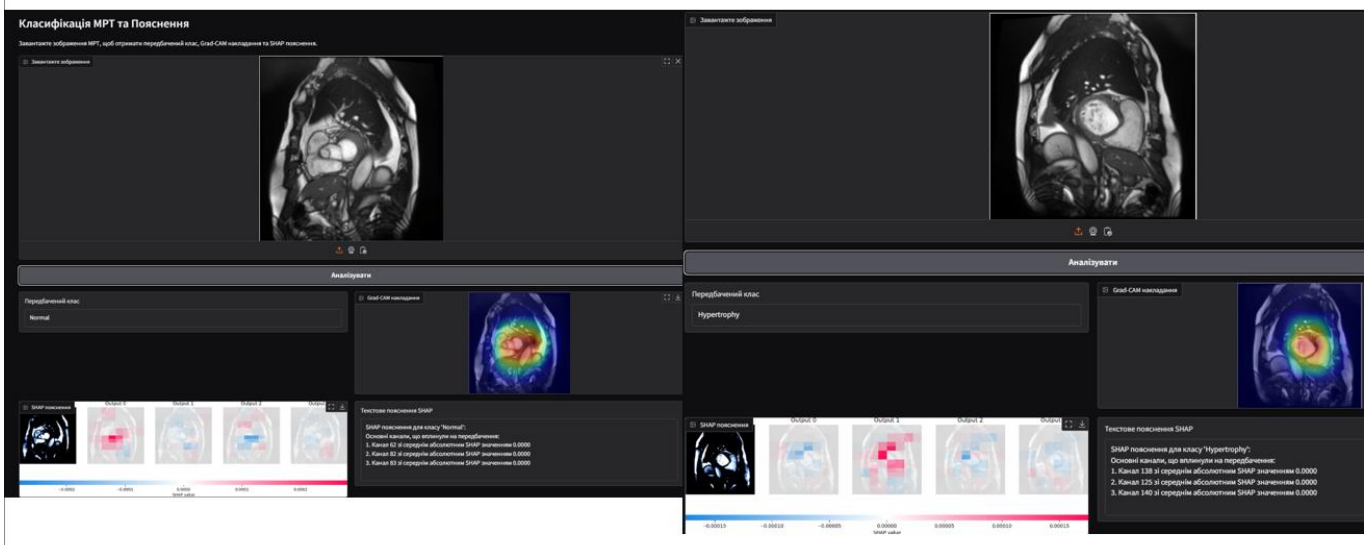
9

Приклад результатів тестування для різних класів



10

Приклади роботи інтерпретації патологій



Висновки

- Кваліфікаційна робота магістра присвячена підвищенню якості інтерпретування результатів виявлення патологій серця на основі МРТ-зображень за допомогою моделей глибокого навчання. Основна увага приділена розробці модуля програмного забезпечення для пояснення діагностичних рішень.
- Запропоновано інтеграцію архітектур U-Net для сегментації серця та ResNet50 для класифікації патологій. Для пояснення рішень використано методи Grad-CAM і SHAP, що забезпечує прозорість і зрозумілість рішень для медичних фахівців.
- Розроблена інформаційна система включає всі етапи процесу: обробку зображень, класифікацію патологій, візуалізацію важливих зон та надання текстових пояснень. Система пройшла тестування, показавши точність понад 75% для класифікації основних класів патологій.
- Інтегровано веб-інтерфейс Gradio для спрощення взаємодії з системою, що робить її доступною навіть для користувачів без технічної підготовки.
- Основні результати: точна автоматизована діагностика патологій серця з пояснюваними рішеннями, що підвищує довіру медичних фахівців до системи штучного інтелекту.

Thu Dec 12 19:00:19 EET 2024, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.258 Educational

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 14%**

	Документ		Сумарний збіг по Базі Даних	
ID: 158325 Назва: КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА на тему Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ Додано в БД: 2024-12-12 Автора: Максим ДІХТЯР Керівники: Павло РАДЮК Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	72196	1075	3820 (5%)	57 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Максим ДІХТЯР

Співавтор:

Назва: Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ

Науковий керівник: Павло РАДЮК, старший викладач кафедри, Ph.D.

Підрозділ: Кафедра комп'ютерних наук

Коефіцієнт подібності 1:1.9%

Коефіцієнт подібності 2:0.9%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2024-12-12 18:37:27.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 12.12.2024 експерт Радюк П.Р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ КАФЕДРИ КОМП'ЮТЕРНИХ НАУК
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ*

Автор: *студент групи КНм-23-1 Діхтяр Максим Олександрович*

Спеціальність: *122 Комп'ютерні науки*

Освітня програма: *освітньо-професійна*

Науковий керівник: *док. філ., ст. викл. Радюк П.М.*

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	***відповідає***
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі Діхтяра М.О., не є плагіатом, оскільки: запозичення розміщені в розділі огляду наявних підходів, не описують безпосередньо авторську роботу і не стосуються її результатів; до запозичень входять фрагменти програмного коду, що не мають авторства і містять поширені конструкції; поміж запозичень є загальновідомі терміни та скорочення.

Обсяг запозичень, визначений системами виявлення збігів/ ідентичності/схожості, складає:

- за системою Anti-Plagiarism: 2.0%: виявлені запозичення є фрагментарними та вказують на відомі терміни;

- за системою StrikePlagiarism: КП 1 – 1,85%, КП 2 – 1,04%: виявлені запозичення є фрагментарними, містять поширені конструкції та схеми з відповідними посиланнями, загальновідомі терміни, скорочення та визначення.

Отже, запозичення є допустимими та адресуються до першоджерел, що, з урахуванням наведених обґрунтувань, свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ *Павло РАДЮК*

Гарант ОП _____ *Руслан БАГРІЙ*

Завідувач кафедри КН _____ *Олександр БАРМАК*

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ

Кафедра комп'ютерних наук

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну роботу магістра

студента *гр. КНм-23-1 Діхтяра Максима Олександровича*

за темою *Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ*

1. Актуальність теми

Автоматизоване інтерпретування результатів виявлення патологій серця дає можливість медичним фахівцям одержувати важливу клінічну інформацію, що може сприяти більш точному діагностуванню серцево-судинних захворювань. Заразом, пояснення рішень штучного інтелекту, застосованого до зображень магнітно-резонансної томографії, залишається відкритою проблемою. Відповідно актуальною є задача розроблення нового програмного рішення на основі згорткової нейронної мережі, яке дало б змогу інтерпретувати результати з високим показником довіри.

2. Відповідність роботи предметній області 122 Комп'ютерні науки та загальним вимогам наукових робіт

Відповідно до стандарту магістра вищої освіти України спеціальності 122 Комп'ютерні науки, описом предметної галузі є методи та технології аналізу інформації. Метою поданої роботи є підвищення рівня якості інтерпретування виявлення патологій серця за зображенням МРТ, отриманих за допомогою моделі глибокого навчання. Цього досягнуто завдяки використанню методів інтерпретації, що відповідає вимогам розв'язання теоретичних і прикладних задач у галузі інформаційних технологій. Отже, результати кваліфікаційної роботи магістра відповідають стандарту спеціальності 122 Комп'ютерні науки.

3. Професійні та особистісні якості магістранта

Під час виконання кваліфікаційної роботи магістрант Діхтяр Максим зарекомендував себе кваліфікованим фахівцем і дисциплінованим виконавцем поставлених завдань. Він продемонстрував компетентності та результати навчання, що відповідають освітньо-професійній програмі рівня магістра за спеціальністю 122 Комп'ютерні науки.

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Студент самостійно одержав результати кваліфікаційної роботи та обґрунтував їхню практичну значущість, виконавши всі поставлені завдання власними силами.

5. Наукова новизна та оригінальність запропонованих підходів

Отримані результати відзначаються оригінальністю застосованого підходу до інтерпретування модельних рішень. Пропонована методика доповнює наявні методи аналізу медичних зображень на основі глибокого навчання та підвищує довіру лікарів до отримуваних результатів.

6. Ступінь оволодіння методами дослідження

У процесі проєктування та програмної реалізації методу інтерпретування результатів студент Діхтяр Максим продемонстрував достатній рівень компетентностей, умінь і навичок використання інструментарію інформаційних технологій.

7. Повнота та якість розкриття теми роботи

Тема роботи повністю обґрунтована й розкрита; актуальність предметної галузі та відомі дослідження проаналізовано. Студент успішно виконав усі поставлені завдання, а розроблене програмне забезпечення для інтерпретації результатів глибинних моделей відповідає технічним вимогам спеціальності 122 Комп'ютерні науки.

8. Логічність, послідовність, аргументованість, літературна грамотність викладення матеріалу

Матеріал кваліфікаційної роботи Діхтяра Максима викладено логічно, послідовно, з належною аргументацією, у відповідності зі стандартами. Мова і стиль викладення забезпечують доступність та зрозумілість матеріалу.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Розроблений вебзастосунок може бути використаний медичними фахівцями чи їхніми асистентами під час діагностування серцево-судинних захворювань за МРТ-зображеннями, надаючи додаткове пояснення до отриманих результатів у режимі реального часу.

10. Висновок про можливість допуску кваліфікаційної роботи до захисту, на яку оцінку заслуговує робота

З огляду на високий рівень виконання та дотримання вимог, вважаю, що кваліфікаційна робота Діхтяра Максима може бути допущена до захисту. Рекомендую оцінку – «добре».

Керівник _____ *док. філ., ст. викл. каф. КН Павло РАДЮК*

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ

Кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

студента *гр. КНм-23-1 Діхтяра Максима Олександровича*
за темою: *Метод інтерпретування результатів виявлення патологій серця за зображенням МРТ*

1. Актуальність обраної теми

Інтерпретування результатів виявлення патологій серця за зображеннями магнітно-резонансної томографії (МРТ) є важливим завданням у медичній обробці зображень. Зрозуміле пояснення рішень моделей глибокого навчання дає змогу отримати додаткову інформацію про стан серцево-судинної системи пацієнта, що допомагає в точній діагностиці та плануванні лікування. Використання згорткових нейронних мереж слугує відмінним інструментом для аналізу зображень та дає можливість досягти високої точності й наочності результатів, що має велике значення для клінічної практики. Обрана тема є актуальною, оскільки підвищення інтерпретованості рішень моделей покращить довіру медичних фахівців до систем штучного інтелекту та сприятиме ефективнішій діагностиці і лікуванню серцево-судинних захворювань.

2. Відповідність роботи предметній області 122 Комп'ютерні науки та загальним вимовам до наукових робіт

Робота повністю відповідає предметній області 122 Комп'ютерні науки, оскільки досліджує застосування глибокого навчання та методів інтерпретації результатів у аналізі медичних зображень. Вона також відповідає загальним вимогам до наукових робіт, оскільки містить чітко сформульовану мету, завдання, методологію, наукову новизну, а також практичну реалізацію та експериментальне тестування.

3. Повнота розкриття мети та завдань дослідження

У процесі виконання кваліфікаційної роботи були детально розкриті мета та завдання. Автором проведено глибокий аналіз предметної галузі інтерпретування результатів виявлення патологій серця на зображеннях МРТ засобами штучного інтелекту. Проведено систематизацію основних принципів та методів, що використовуються у задачах аналізу медичних зображень. Визначено ключові виклики та труднощі, з якими зустрічаються дослідники у разі пояснення рішень глибоких нейронних мереж. Загалом, подану роботу можна вважати повною та збалансованою в розкритті мети та завдань.

4. Наявність наукової новизни

У роботі представлено нові підходи до інтерпретації результатів виявлення патологій серця за зображеннями МРТ з використанням методів глибокого навчання. Автор запропонував удосконалені алгоритми аналізу, які покращують точність та швидкість обробки даних, а також інтегрували їх у нову інформаційну систему. Проведені експерименти демонструють переваги запропонованих методів у порівнянні з існуючими рішеннями, що підтверджує наукову новизну роботи.

5. Зміст кожного розділу роботи

У першому розділі кваліфікаційної роботи проведено дослідження методів і технологій штучного інтелекту для аналізу зображень магнітно-резонансної томографії. Другий розділ присвячений проєктуванню методу інтерпретування результатів виявлення патологій серця за зображенням МРТ. У третьому розділі наведено та описано програмну реалізацію інформаційної системи з використанням обраного методу інтерпретування результатів у вигляді вебзастосунку. У четвертому розділі проведено експериментальне тестування реалізованого методу інтерпретування результатів. Насамкінець оформлено висновки до виконаних у роботі завдань.

6. Ступінь розкриття теми роботи

Спроєктований метод та його програмна реалізація для інтерпретування результатів виявлення патологій серця на МРТ із використанням ЗНМ демонструють практичну цінність у медичній галузі. Метод наочно пояснює вплив певних ділянок зображення на рішення моделі, підвищуючи зрозумілість результатів для лікарів. Реалізація відзначається високою точністю та сприяє зростанню довіри до штучного інтелекту й ефективності лікування.

7. Якість оформлення кваліфікаційної роботи

Записка до кваліфікаційної роботи підготовлена автором якісно; вона характеризується логічним структурованим викладом матеріалу та вмістом, що переконливо підтверджує висунуті аргументи. Автор проявив високу грамотність та здатність використовувати влучну та адекватну лексику, що додає роботі вагомості та доречності.

8. Недоліки оформлення кваліфікаційної роботи

Робота також містить недоліки. Зокрема деякі оформлювальні аспекти потребують удосконалення, наприклад, уточнення переліку скорочень та узгодження оформлення посилань з вимогами стандартів.

9. Недоліки кваліфікаційної роботи

Експериментальне тестування із вебзастосунком у роботі розкрито недостатньо, замало подано тест-кейсів для повноти охоплення всіх функцій програмної реалізації. У тексті використано кілька абревіатур, але їх не було включено до переліку скорочень. Деякі твердження в роботі потребують посилань на додаткові джерела.

10. Загальний висновок (допускається чи не допускається до захисту), та оцінка на яку заслуговує кваліфікаційна робота

*З огляду на достатній рівень виконання та забезпечення всіх необхідних вимог, вважаю, що подана кваліфікаційна робота магістра Діхтяра Максима є оригінальним та завершеним науковим дослідженням. Тому кваліфікаційна робота може бути допущена до захисту. Рекомендована оцінка – **«добре»**.*

Опонент (прізвище, ім'я, по батькові, посада, місце роботи)

Лисенко Сергій Миколайович, д.т.н. проф. каф. КІІС

«16» 12 2024 р.

(підпис)