

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
НЕЙРОМЕРЕЖНИХ МЕТОДІВ РОЗПІЗНАВАННЯ
ПУХЛИНИ ГОЛОВНОГО МОЗКУ ЗА ЗНІМКОМ MRT
RESEARCH AND SOFTWARE IMPLEMENTATION OF
NEURAL NETWORK METHODS FOR RECOGNITION OF
BRAIN TUMOR IN MRI IMAGES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220

Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

КАМІНСЬКИЙ Д.О.
(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.
(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.
(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ КНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 122 Комп'ютерні науки _____
(код і найменування)
Освітня програма (спеціалізація) _____ Комп'ютерні науки _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
«_____» _____ 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КАМІНСЬКОГО Дениса Олексійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація нейромережних методів розпізнавання пухлини головного мозку за знімком МРТ. Research and Software Implementation of Neural Network Methods for Recognition of Brain Tumor in MRI Images

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 24 » квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблемної області. 2. Матеріали і методи. 3. Проєктування програми. 4. Розробка програми. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	1 тиждень	Розділ 1
3	Дослідження методів розпізнавання	2-3 тижні	Розділ 2
4	Розробка архітектури програми	4 тиждень	Розділ 3
5	Розробка програми	5-6 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
8	Захист роботи	8 тиждень	

Студент(ка)

_____ Денис КАМІНСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Валерій ЛЬОВКІН
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи містить 145 с., 12 табл., 83 рис., 3 дод., 23 джерела.

ПУХЛИНА МОЗКУ, ДІАГНОСТУВАННЯ, МРТ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, PYTHON, TENSORFLOW, FLASK.

Об'єкт дослідження – процес діагностування наявності пухлини мозку.

Предмет дослідження – методи та програмні засоби діагностування наявності пухлини мозку за знімком МРТ.

Мета роботи – дослідження та розробка програмного забезпечення з використання нейронних мереж для діагностування наявності пухлини мозку за знімком МРТ пацієнта.

Матеріали, методи та технічні засоби: середовище розробки Visual Studio Code 2023, мова програмування Python, бібліотеки Pandas, Tensorflow, Keras та Flask, моделі на основі нейронних мереж.

Результати. Проведено аналіз предметної області, встановлено вимоги до програмного забезпечення. Досліджено можливі методи та моделі діагностування з використанням нейронних мереж, розроблено вебінтерфейс.

Висновки. Розроблене програмне забезпечення дозволяє завантажувати знімок МРТ пацієнта через вебінтерфейс та проводити діагностування на наявність пухлини на ньому. Наукова новизна роботи полягає у запропонованому методі діагностування пухлин головного мозку, який характеризується використанням модифікованої архітектури AlexNet. Ця модифікація дозволяє значно зменшити кількість налаштовуваних параметрів, що спрощує вимоги до навчання моделі, при цьому забезпечуючи покращення точності діагностування.

Галузь використання – медичне діагностування. Розроблене програмне забезпечення призначене для допомоги роботі лікарів терапевтів або невропатологів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 145 pages, 12 tables, 83 figures, 3 appendixes, 23 sources.

BRAIN TUMOR, DIAGNOSIS, MRI, CONVOLUTIONAL NEURAL NETWORKS, PYTHON, TENSORFLOW, FLASK.

Object of research - the process of diagnosing the presence of a brain tumor.

The subject of the study is methods and software for diagnosing the presence of a brain tumor from an MRI image.

Purpose of the study - research and development of software for the use of neural networks to diagnose the presence of a brain tumor from an MRI image of a patient.

Materials, methods and techniques: Visual Studio Code 2023 development environment, Python programming language, Pandas, Tensorflow, Keras and Flask libraries, models based on neural networks.

Results. The subject area was analyzed and software requirements were established. Possible methods and models of diagnostics using neural networks were investigated, and a web interface was developed.

Conclusions. The developed software allows you to upload a patient's MRI image through a web interface and diagnose the presence of a tumor on it. The scientific novelty of the work lies in the proposed method of diagnosing brain tumors, which is characterized by the use of a modified AlexNet architecture. This modification allows to significantly reduce the number of configurable parameters, which simplifies the requirements for model training, while providing improved diagnostic accuracy.

The field of application is medical diagnostics. The developed software is designed to help the work of general practitioners or neurologists.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Діагностування пухлин головного мозку як проблема класифікації	11
1.1.1 Різновиди пухлин головного мозку.....	11
1.1.2 Діагностика пухлини головного мозку	13
1.1.3 Класифікація пухлин головного мозку	13
1.2 Огляд існуючих методів машинного навчання.....	15
1.3 Висновки до першого розділу.....	16
2 Матеріали і методи.....	18
2.1 Огляд методів інтелектуального аналізу даних	18
2.1.1 Архітектура AlexNet	20
2.1.2 Архітектура VGG16	21
2.1.3 Архітектура VGG19	22
2.1.4 Архітектура Inception v3.....	22
2.1.5 Архітектура ResNet50	23
2.2 Аналіз набору даних	25
2.3 Метод діагностування пухлин головного мозку.....	26
2.4 Висновки до другого розділу	29
3 Проектування програми	31
3.1 Вибір мови програмування	31
3.2 Вибір середовища розробки.....	32
3.3 Вибір вебфреймворку	33
3.4 Вибір бази даних	35
3.5 Проектування структури бази даних.....	35
3.6 Архітектура програмного забезпечення	38
3.7 Визначення функціональних вимог до застосунку	39
3.8 Висновки до третього розділу	42

4 Розробка програми	44
4.1 Алгоритм функціонування застосунку	44
4.2 Структура програмного застосунку	46
4.3 Висновки до четвертого розділу.....	50
5 Експлуатація, тестування та експериментальне дослідження програми	51
5.1 Опис застосування програми	51
5.2 Умови виконання програми	51
5.3 Інструкція по експлуатації програми	51
5.3.1 Інструкція з експлуатації програми для лікаря.....	54
5.3.2 Інструкція з експлуатації програми для пацієнта.....	63
5.3.3 Інструкція з експлуатації програми для адміністратора	65
5.4 Методика та результати тестування.....	74
5.5 Тестування застосунку та його результати	82
5.6 Висновки до п'ятого розділу.....	83
Висновки	85
Перелік джерел посилання	87
Додаток А Технічне завдання	90
Додаток Б Текст програми	94
Додаток В Слайди презентації.....	136

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	– Convolutional Neural Network;
GBM	– Glioblastoma Multiforme;
IDE	– Integrated development kit;
ORM	– Object Relational Mapper;
ReLU	– Rectified Linear Units;
RGB	– Red, Green, Blue;
SVM	– Support Vector Machine;
ViTs	– Vision Transformers;
БД	– база даних;
БНМ	– багатошарова нейронна мережа;
ЗНМ	– згорткова нейронна мережа;
КТ	– комп'ютерна томографія;
МРТ	– магнітно-резонансна томографія;
НМ	– нейронна мережа;
ПЕТ	– позитронно-емісійна томографія;
ПЗ	– програмне забезпечення;
ШНМ	– штучна нейронна мережа.

ВСТУП

Пухлина головного або спинного мозку є одним з критичних захворювань, які виникають у величезній кількості людей. На даний момент оцінюється, що 24810 дорослих, 5230 дітей віком до 20 років мають таке захворювання у США у 2023 році, у 2020 році кількість померлих від цієї хвороби у світі складала 251329 осіб, що є десятою за кількістю смертей хворобою у світі [1]. Ці цифри підкреслюють гостру потребу в інноваційних підходах у сфері медичної діагностики.

Наслідки пухлин головного мозку можуть бути важкими і різноманітними. Вони включають постійні головні болі, судоми, психічні або поведінкові зміни, проблеми з пам'яттю або зміни особистості, прогресуючу слабкість або параліч з одного боку тіла, а також проблеми із зором чи мовленням [2]. Ці симптоми глибоко впливають на життя людей, роблячи своєчасне виявлення та діагностику пухлин головного мозку ключовими факторами ефективного лікування.

У сфері охорони здоров'я людський елемент, незалежно від його кваліфікації, вразливий до втоми та можливих помилок. Складність медичної діагностики означає, що навіть досвідчені лікарі можуть допускати критичні помилки. Це підкреслює важливість технологічної підтримки.

Запропоноване програмне забезпечення допомагає подолати обмеження людської витривалості, сприяючи більш повній та своєчасній оцінці результатів сканувань магнітно-резонансної томографії (МРТ). Співпраця між штучним інтелектом і медичним досвідом позиціонується не як заміна клінічного судження, а як його доповнення, яке розширює можливості медичних працівників і покращує шанси на успішне лікування пацієнтів.

Для отримання більш точної класифікації захворювання важливо впроваджувати сучасні методи діагностики. Серед них використання штучного інтелекту (ШІ), зокрема нейронних мереж (НМ), відіграє важливу роль. Спільне використання цих технологій з лікарем може помітно зменшити

кількість неточних діагнозів та сприяти збереженню здоров'я та життя пацієнтів.

Отже, основною метою даної роботи є розробка програмного забезпечення з використання нейронних мереж для діагностування наявності пухлини мозку за знімком МРТ пацієнта. Дана мета спрямована на зниження ризику невірного визначення діагнозу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Перший розділ присвячений детальному аналізу предметної області, пов'язаної з проблемою діагностування пухлин мозку. У розділі розглянуто основні поняття, класифікацію пухлин головного мозку, методи їх діагностування. Також буде розглянуто існуючі методи машинного навчання для класифікації зображень.

1.1 Діагностування пухлин головного мозку як проблема класифікації

Пухлина головного мозку – це маса аномально зростаючих клітин у головному мозку або черепі. Вона може бути доброякісною (не раковою) або злоякісною (раковою). На відміну від інших видів раку, рак, що виникає з тканини мозку (первинний рак мозку), рідко поширюється. Незалежно від того, доброякісні вони чи злоякісні, всі пухлини мозку є серйозними. Зростаюча пухлина з часом стискає і пошкоджує інші структури мозку [3].

1.1.1 Різновиди пухлин головного мозку

Існує дві категорії пухлин головного мозку: первинні та вторинні.

Коли пухлини головного мозку виникають у дітей, вони, швидше за все, є первинними. У дорослих вторинні пухлини мозку зустрічаються набагато частіше, ніж первинні [4].

До первинних пухлин головного мозку відносять будь-яку пухлину, яка починається в головному мозку. Первинні пухлини можуть починатися з клітин мозку, оболонок навколо мозку (мозкових оболонок), нервів або залоз. Пухлини можуть безпосередньо руйнувати клітини мозку. Вони також можуть пошкоджувати клітини, викликаючи запалення, чинячи тиск на інші частини мозку і підвищуючи тиск всередині черепа [5].

Причина виникнення первинних пухлин головного мозку невідома. Існує багато факторів ризику, які можуть відігравати певну роль:

- деякі спадкові захворювання підвищують ризик виникнення пухлин головного мозку, включаючи нейрофіброматоз, синдром фон Гіппеля-Ліндау, синдром Лі-Фраумені та синдром Турко;
- лімфоми, які починаються в мозку у людей з ослабленою імунною системою, іноді пов'язані з інфікуванням вірусом Епштейна-Барр [5].

Первинні пухлини поділяються на гліальні (складаються з клітин центральної нервової системи та периферичної нервової системи, які не виробляють електричних імпульсів [6]) і негліальні (розвиваються на або в структурах мозку, включаючи нерви, кровоносні судини й залози), а також на доброякісні та злоякісні [7].

Вторинні пухлини мозку виникають, коли рак починається в іншому місці й поширюється на мозок. Коли рак поширюється, це називається метастатичним раком [4].

Іноді пухлина головного мозку може бути першою ознакою раку, який почався десь в іншому місці організму [4].

Вторинні пухлини найчастіше виникають у легенях або молочній залозі. Інші види раку, особливо меланома (різновид раку шкіри), нирково-клітинний рак (різновид раку нирок) і лімфома (рак імунної системи), можуть поширюватися на мозок. Коли це відбувається, рак є таким самим, як і початковий рак. Наприклад, рак легенів, який поширюється на мозок, відомий як метастатичний рак легенів, оскільки клітини пухлини нагадують аномальні клітини легенів. Вторинні пухлини мозку зустрічаються набагато частіше, ніж первинні [3].

Метастатичні пухлини вважаються раком і є злоякісними. Вони вражають майже кожного четвертого хворого на рак, або приблизно 150 000 людей на рік. До 40 відсотків людей, хворих на рак легенів, мають метастатичні пухлини головного мозку. У минулому результати лікування

пацієнтів з такими пухлинами були дуже поганими, а середній показник виживання становив лише кілька тижнів [7].

1.1.2 Діагностика пухлини головного мозку

Діагностика починається зі збору анамнезу. Лікар запитує про симптоми, шкідливі звички, перенесені хвороби та лікування. Він також проводить неврологічне обстеження. Після лікар призначить одне з цих досліджень:

- комп'ютерна томографія (КТ). Цей тест створює зображення поперечного перерізу мозку. Для цього використовується рентгенівська камера, яка обертається навколо тіла. Іноді перед скануванням у вену вводять барвник, щоб зробити пухлину більш помітною;

- магнітно-резонансна томографія (МРТ). Цей тест використовує потужний магніт, радіохвилі та комп'ютер для створення зображень мозку. МРТ може забезпечити кращий огляд деяких ділянок мозку, ніж КТ. Для покращення зображень у кров може вводитися спеціальний барвник. Магнітно-резонансна ангиограма схожа на МРТ, але вона дивиться на потік крові в артеріях. Це може допомогти лікарям знайти аневризми або краще визначити пухлини;

- позитронно-емісійна томографія (ПЕТ). Для цього тесту радіоактивну глюкозу (цукор) вводять у вену. Сканер, що обертається, виділяє ділянки, де клітини споживають багато глюкози (ракові клітини використовують більше глюкози, ніж нормальні клітини) [3].

Якщо є підозра, що пухлина мозку є вторинним раком, можуть бути проведені візуалізаційні дослідження інших органів [3].

1.1.3 Класифікація пухлин головного мозку

До доброякісних відносять такі пухлини:

- гангліоцитоми, гангліоми та анапластичні гангліогліоми – це рідкісні пухлини нервової системи, зазвичай виникають у молодих людей. Ці пухлини містять неопластичні нервові клітини і є відносно добре диференційованими;

- менінгіоми – найпоширеніші доброякісні пухлини головного мозку, що складають 10-15% всіх новоутворень головного мозку. Вони походять з мозкових оболонок і лише малий відсоток з них є злоякісними;

- шванноми – це доброякісні пухлини, які розвиваються уздовж нервів і часто витісняють залишки нормального нерва. Акустичні невроми є найпоширенішим типом шванном, які виникають з вестибулокохлеарного нерва і можуть призводити до серйозних ускладнень при розростанні [7].

Найпоширенішим типом пухлин головного мозку у дорослих є гліоми. На них припадає 78% злоякісних пухлин мозку. Вони виникають з гліальних клітин мозку, таких як астроцити, епендимні клітини та олігодендрогліальні клітини. До гліальних пухлин відносяться наступні:

- астроцитоми – найпоширеніша гліома, на яку припадає близько половини всіх первинних пухлин головного і спинного мозку. Астроцитоми розвиваються з астроцитів. Вони можуть виникати в багатьох частинах мозку, але найчастіше в головному мозку. Астроцитоми можуть розвиватися у людей різного віку, але вони більш поширені у дорослих, особливо у чоловіків середнього віку;

- епендимоми виникають внаслідок неопластичної трансформації епендимних клітин, що вистилають шлуночкову систему, і становлять два-три відсотки всіх пухлин головного мозку. Більшість з них мають чіткі межі, але деякі – ні;

- мультиформна гліобластома (Glioblastoma multiforme (GBM)) - найбільш агресивний тип гліальних пухлин. Ці пухлини швидко ростуть, поширюються на інші тканини й мають поганий прогноз. Вони можуть складатися з декількох різних типів клітин, таких як астроцити та

олігодендроцити. GBM частіше зустрічається у людей у віці від 50 до 70 років і частіше у чоловіків, ніж у жінок;

- медулобластоми зазвичай виникають у мозочку, найчастіше у дітей. Це пухлини високого ступеня злоякісності, але вони зазвичай добре піддаються променевої та хімієтерапії;

- олігодендрогліоми походять з клітин, які виробляють мієлін, що є ізоляцією для нервової проводки мозку [7].

1.2 Огляд існуючих методів машинного навчання

Нижче наведено огляд декількох найпопулярніших методів машинного навчання для класифікації зображень.

Згорткові нейронні мережі або Convolutional Neural Network (CNN) – це глибинні штучні нейронні мережі (ШНМ) прямого поширення. Фактично вони є різновидом БНМ, адаптованої до обробки зображень. Згорткові мережі інспіровані організацією зорової кори тварин, де окремі нейрони кори реагують на стимули лише в обмеженій області зорового поля (рецептивному полі). Рецептивні поля різних нейронів частково перекриваються таким чином, що вони покривають усе зорове поле [8].

Мащини опорних векторів або Support vector machine (SVM) використовуються для задач класифікації та регресії в керованому машинному навчанні, з особливим застосуванням у класифікації зображень. Зображення, оброблені як двовимірні масиви пікселів, містять значення, які представляють інтенсивність кожного пікселя в колірних каналах RGB. Для класифікації зображення за допомогою SVM спочатку виділяються такі характеристики, як значення кольору, виявлення країв і текстури, а потім використовуються як вхідні дані для алгоритму SVM. Алгоритм SVM визначає гіперплощину, яка розділяє різні класи в просторі ознак, прагнучи максимізувати відстань між найближчими точками різних класів, відомими як опорні вектори. Такий підхід дозволяє SVM ефективно обробляти багатовимірні дані та зменшує

ризик перенавчання порівняно з іншими алгоритмами, такими як нейронні мережі [9].

Трансформатори зору або Vision Transformers (ViTs) – це клас моделей глибокого навчання, адаптованих з трансформерів в обробці природної мови для вирішення завдань комп'ютерного зору. На відміну від традиційних методів, які покладаються на згорткові нейронні мережі (CNN), ViTs розбивають зображення на фрагменти і застосовують механізми самоуваги для виявлення глобальних залежностей всередині зображення. Такий підхід дозволяє ШНМ досягати чудової продуктивності в різних завданнях обробки зображень, включаючи, але не обмежуючись ними, класифікацію зображень, виявлення об'єктів і семантичну сегментацію. Трансформатори зору ефективні в класифікації зображень завдяки своїй здатності вивчати глобальні особливості зображення за допомогою механізмів самоуваги. На відміну від CNN, які зосереджуються на локальних особливостях, ViTs аналізують взаємозв'язки між усіма частинами зображення, що дозволяє їм більш ефективно розпізнавати складні патерни. Крім того, масштабованість трансформантних моделей дозволяє ефективно навчатися на великих наборах даних, що ще більше підвищує їхню продуктивність у задачах класифікації зображень, забезпечуючи все більш точні результати зі збільшенням розміру набору даних [10].

1.3 Висновки до першого розділу

У даному розділі було розглянуто основні поняття та класифікацію пухлин головного мозку, що надає розуміння важливості даного проєкту в медичній сфері. Для загального розуміння було розглянуто різні методи діагностики пухлин головного мозку, хоч в даній роботі будуть використовуватися лише знімки МРТ. Також було розглянуто різні методи машинного навчання, які використовуються для класифікації зображень. Далі вже буде детально розглянуто метод глибоких згорткових нейронних мереж.

Пухлини мозку становлять серйозну загрозу для життя та здоров'я людини, тому задача їх діагностування є критичною. Отже, підвищення точності діагностування є надзвичайно актуальним і важливим завданням сьогодення. Для вирішення цієї задачі необхідно виконати наступні завдання:

- аналіз предметної області та розробка технічного завдання;
- експериментальне дослідження різних архітектур CNN;
- експериментальне дослідження архітектури з найбільшою точністю для подальшого покращення результату на основі запропонованого рішення;
- розробка вебінтерфейсу;
- проведення тестування програмного забезпечення.

2 МАТЕРІАЛИ І МЕТОДИ

В даному розділі має бути розглянуто існуючі методи інтелектуального аналізу даних та буде проведено аналіз набору даних. Основним результатом має стати опис власного методу діагностування.

2.1 Огляд методів інтелектуального аналізу даних

ЗНМ складається з шарів входу та виходу, а також із декількох прихованих шарів. На відміну загальної парадигми БНМ, приховані шари ЗНМ зазвичай є спеціалізованими і складаються зі згорткових шарів, агрегувальних шарів, повноз'єднаних шарів та шарів нормалізації [8].

На рис. 2.1 наведено схему ЗНМ.

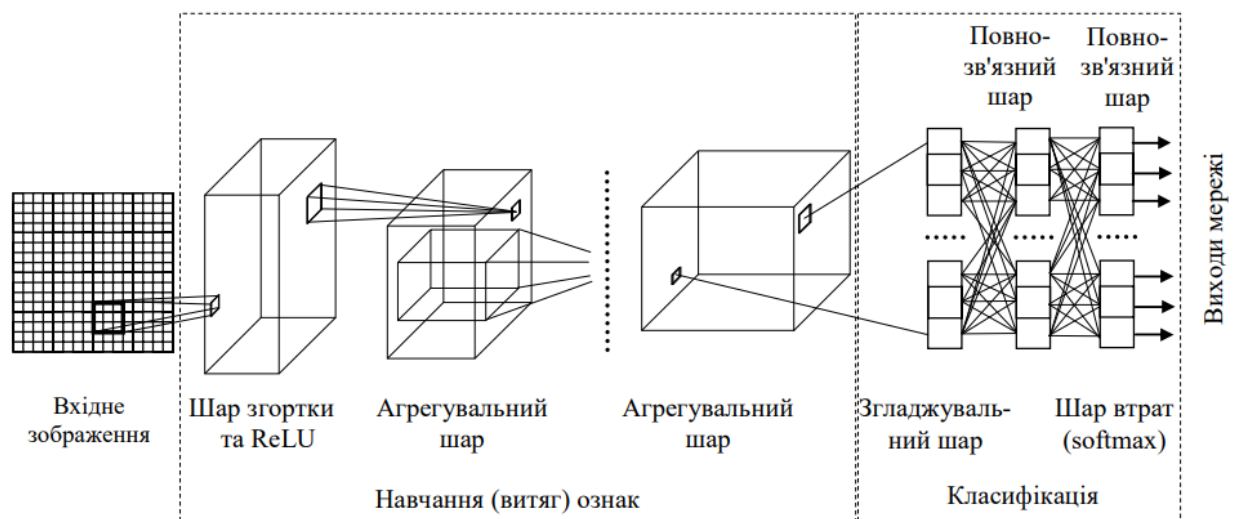


Рисунок 2.1 - Архітектура типової згорткової нейронної мережі [8]

Згортковий шар (convolutional layer) застосовує до входу операцію згортки, передаючи результат до наступного шару. Згортка імітує реакцію окремого нейрона на зоровий стимул. Кожен згортковий нейрон обробляє дані лише для свого рецептивного поля [8].

Параметри шару складаються з набору фільтрів для навчання (або ядер), які мають невеличке рецептивне поле, але простягаються на всю

глибину вхідної ємності. Протягом прямого проходу кожен фільтр здійснює згортку за шириною та висотою вхідної ємності, обчислюючи скалярний добуток даних фільтру та входу, і формуючи двовимірну карту збудження цього фільтру. В результаті мережа навчається, які фільтри активуються, коли вона виявляє певний конкретний тип ознаки у певному просторовому положенні у вході. Складання карт збудження всіх фільтрів уздовж виміру глибини формує повну ємність виходу згорткового шару. Таким чином, кожен запис в ємності виходу може також трактуватися як вихід нейрона, що реагує на невеличку область у вході та має спільні параметри з нейронами тієї ж карти збудження [8].

Агрегувальний шар (pooling layer) ЗНМ призначений для локального або глобального агрегування та об'єднує виходи кластерів нейронів одного шару до одного нейрону наступного шару. Агрегування є різновидом нелінійного зниження дискретизації, ідея якого полягає у тому, що точне положення ознаки не так важливе, як її грубе положення відносно інших ознак. Агрегувальний шар слугує поступовому скороченню просторового розміру подання для зменшення кількості параметрів та об'єму обчислень у мережі, і відтак також для контролю перенавчання. В архітектурі ЗНМ є звичним періодично вставляти агрегувальний шар між послідовними згортковими шарами. Операція агрегування забезпечує ще один різновид інваріантності відносно паралельного перенесення. Агрегувальний шар діє незалежно на кожен зріз глибини входу, і зменшує його просторовий розмір [8].

Максимізаційне агрегування (max pooling) використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Воно розділяє вхідне зображення на набір прямокутників без перекриттів, і для кожної такої підобласті виводить її максимум [8].

Усереднювальне агрегування (average pooling) використовує усереднене значення з кожного з кластерів нейронів попереднього шару [8].

Шар зрізаних лінійних вузлів (Rectified Linear Units layer, ReLU) застосовує ненасичувальну передавальну функцію ReLU. Він посилює нелінійні властивості функції ухвалення рішення і мережі в цілому, не зачіпаючи рецептивних полів згорткового шару. Для посилення нелінійності застосовуються й інші функції, наприклад, насичувальні гіперболічний тангенс та сигмоїдна функція [8].

Згладжувальний шар (flatten layer) згортає просторові розмірності входу в розмірність каналу [8].

Повноз'єднаний шар (fully connected layer) з'єднує кожен нейрон одного шару з кожним нейроном наступного шару і забезпечує високорівневі міркування. Такий шар є різновидом шару традиційної БНМ [8].

Шар втрат (loss layer) визначає, як навчання штрафує відхилення між передбаченими та справжніми мітками класів, і є, як правило, завершальним шаром [8].

Навчання ЗНМ здійснюють шляхом використання техніки зворотного поширення помилки [8].

Отже, розглянувши основні компоненти та функції згорткових нейронних мереж, далі наведено розгляд конкретних архітектур, що використовуються для розв'язання конкретних задач обробки зображень.

2.1.1 Архітектура AlexNet

AlexNet – архітектура, яка була розроблена Алексом Крижевським. Вона була опублікована разом з Іллею Суцкевером та консультантом Крижевського Джеффрі Хінтоном і є згортковою нейронною мережею або CNN [11].

Після участі в ImageNet Large Scale Visual Recognition Challenge у 2012 році, AlexNet здобула популярність. Вона увійшла до топ-5 з помилкою в 15,3%. Це на 10,8% менше, ніж у другого призера [11].

Основний підсумок оригінальної статті полягав у тому, що глибина моделі була абсолютно необхідною для її високої продуктивності. Це було досить дорого в обчислювальному плані, але стало можливим завдяки графічним процесорам під час навчання [11].

На рис. 2.2 наведено ілюстрацію архітектури AlexNet.

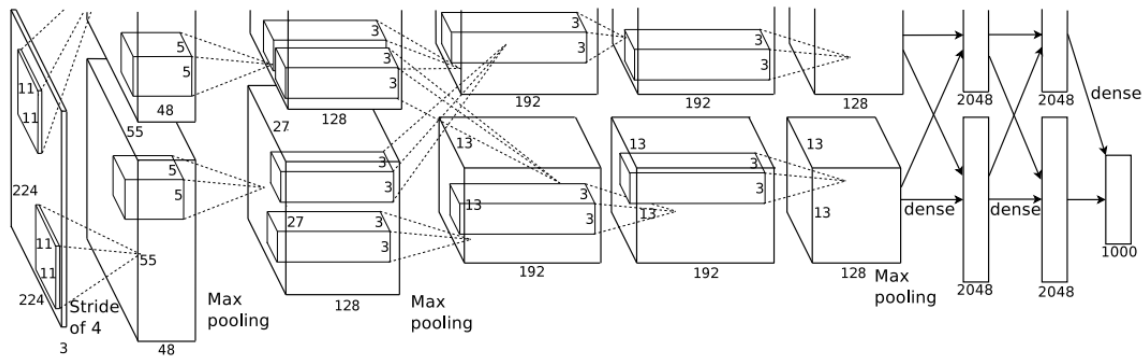


Рисунок 2.2 – Ілюстрація архітектури AlexNet [12]

2.1.2 Архітектура VGG16

VGG16 – це згортова нейромережева модель, запропонована К.Симоняном та А.Зіссерманом з Оксфордського університету в статті "Very Deep Convolutional Networks for Large-Scale Image Recognition". Модель досягає 92,7% точності в топ-5 тестах ImageNet. Це була одна з відомих моделей, представлених на ILSVRC-2014. Покращення порівняно з AlexNet полягає в заміні фільтрів великого розміру ядра (11 і 5 у першому і другому згорткових шарах відповідно) на кілька фільтрів розміром 3×3 ядра, що йдуть один за одним. Модель VGG16 навчалась протягом декількох тижнів і використовувала графічні процесори NVIDIA Titan Black [13].

На рис. 2.3 наведено ілюстрацію архітектури VGG16.

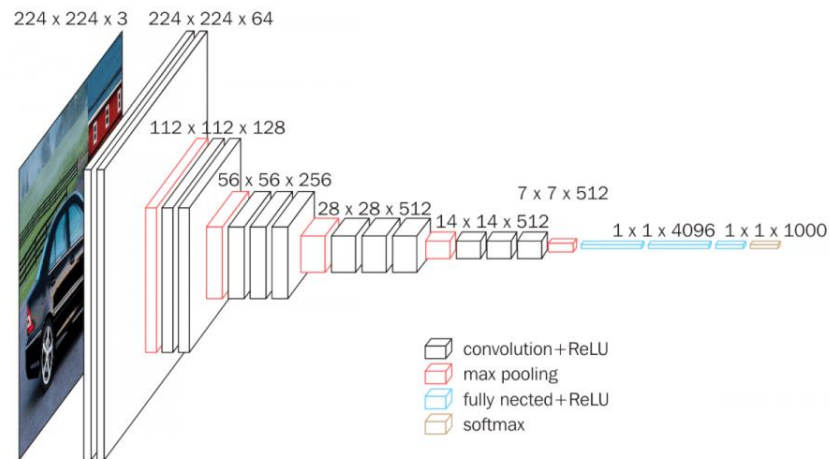


Рисунок 2.3 – Ілюстрація архітектури VGG16 [13]

2.1.3 Архітектура VGG19

VGG19 є розширенням VGG16, має 19 шарів замість 16. Модель має ту саму архітектуру, що й VGG16, з додатковими шарами згортки та максимального об'єднання. VGG19 показує кращі результати в деяких тестах, але вона й більш затратна через більшу кількість параметрів.

На рис. 2.4 наведено ілюстрацію архітектури VGG19.

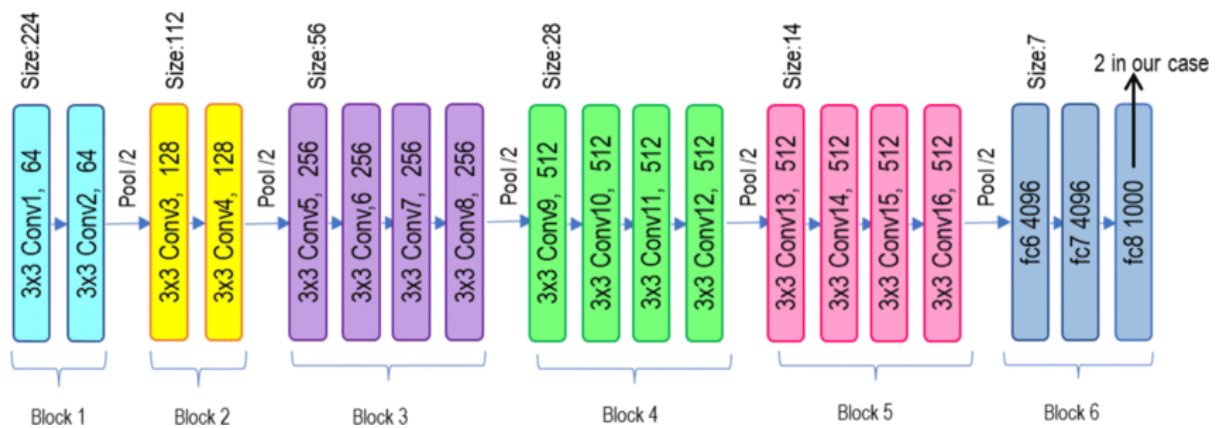


Рисунок 2.4 – Ілюстрація архітектури VGG19 [14]

2.1.4 Архітектура Inception v3

Inception v3 – це модель розпізнавання зображень, яка демонструє точність понад 78,1% на наборі даних ImageNet. Вона базується на

оригінальній статті: "Rethinking the Inception Architecture for Computer Vision" авторства Szegedy та ін. [15].

Сама модель складається з симетричних і асиметричних будівельних блоків, включаючи згортки, середнє об'єднання, максимальне об'єднання, конкатенації, відсікання і повністю з'єднані шари. Паquetна нормалізація широко використовується в моделі й застосовується до вхідних даних активації. Втрати обчислюються за допомогою Softmax [15].

На рис. 2.5 наведено ілюстрацію архітектури Inception v3.

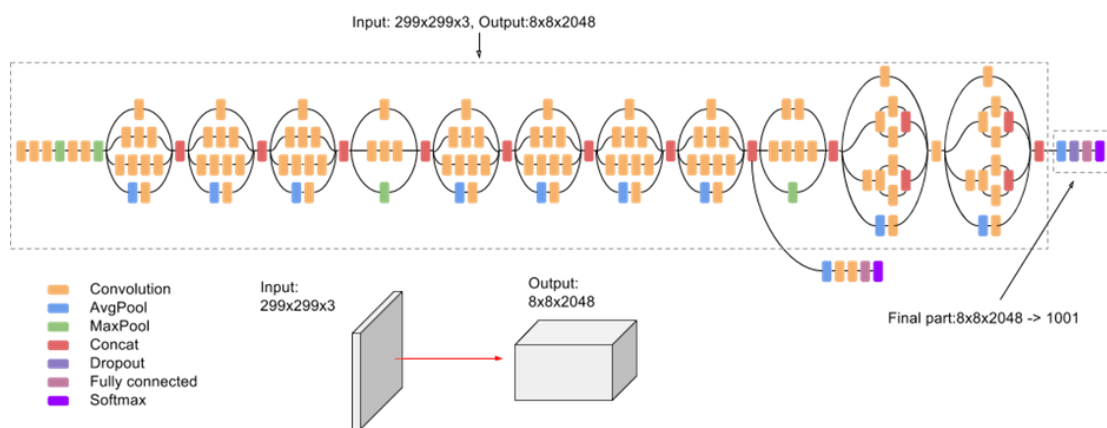


Рисунок 2.5 – Ілюстрація архітектури Inception v3 [15]

2.1.5 Архітектура ResNet50

ResNet розшифровується як "Залишкова мережа" (Residual Network) і є особливим типом згорткової нейронної мережі, представленим у статті 2015 року "Deep Residual Learning for Image Recognition" Хе Каймінгом, Чжаном Сянью, Рен Шаоціном і Сунь Цзянем [16].

ResNet-50 – це 50-шарова згорткова нейронна мережа (48 згорткових шарів, один шар MaxPool і один шар усередненого об'єднання). Залишкові нейронні мережі – це тип штучних нейронних мереж, які формують мережі шляхом складання залишкових блоків [16].

Початковою архітектурою ResNet була ResNet-34, яка складалася з 34 зважених шарів. Вона забезпечила новий спосіб додати більше згорткових

шарів до CNN, не стикаючись з проблемою зникаючого градієнта, використовуючи концепцію коротких з'єднань. Коротке з'єднання "перескакує" через деякі шари, перетворюючи звичайну мережу на залишкову [16].

Стандартна мережа була заснована на нейронних мережах VGG (VGG16 і VGG19) – кожна згорткова мережа мала фільтр 3×3 . Однак ResNet має менше фільтрів і є менш складною, ніж VGGNet [16].

На рис. 2.6 наведено ілюстрацію архітектури ResNet50.

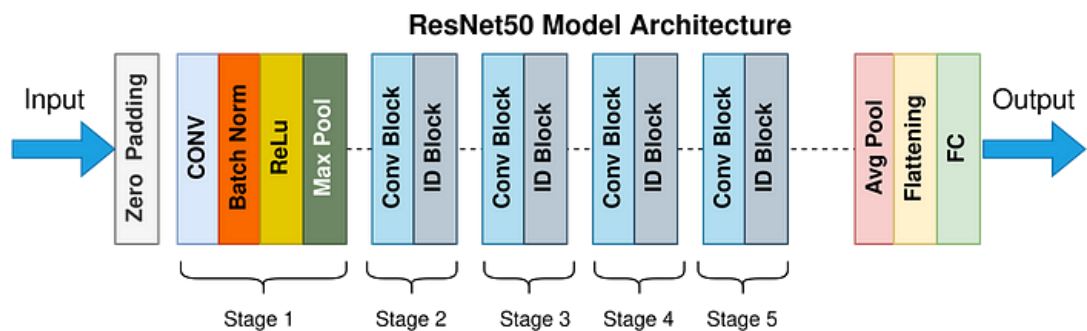


Рисунок 2.6 – Ілюстрація архітектури ResNet50 [17]

Для кращого розуміння ефективності наведених архітектур згорткових нейронних мереж у табл. 2.1 наведено порівняння основних характеристик і результатів на загальнодоступній великомасштабній базі даних ImageNet.

Таблиця 2.1 – Порівняння основних характеристик різних архітектур

Модель	Кількість шарів	Кількість параметрів	Розмір вхідного зображення	Точність на ImageNet (%)
AlexNet	8	60 млн	227x227	84.6
VGG16	16	138 млн	224x224	92.7
VGG19	19	144 млн	224x224	92.7
Inception v3	48	23 млн	299x299	93.7
ResNet50	50	25.6 млн	224x224	95.2

2.2 Аналіз набору даних

Для тренування та тестування моделей буде використано набір даних з відкритого джерела Kaggle [18]. Цей набір даних містить 7,023 зображення, створені за результатами МРТ, серед яких виділені знімки з наступними діагнозами:

- менінгіома (meningioma);
- гліома (glioma);
- пухлина гіпофіза (pituitary);
- зображення без пухлин (notumor).

На рис. 2.7 зображено розподіл вибірки на тестову і навчальну.

На рис. 2.8 зображено розподіл на класи у навчальній вибірці (зліва) й тестовій (справа).

На рис. 2.9 наведено приклади зображень з набору даних.

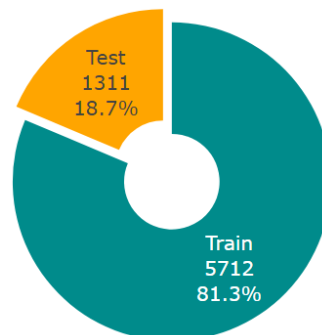


Рисунок 2.7 – Розподіл вибірки на тестову і навчальну

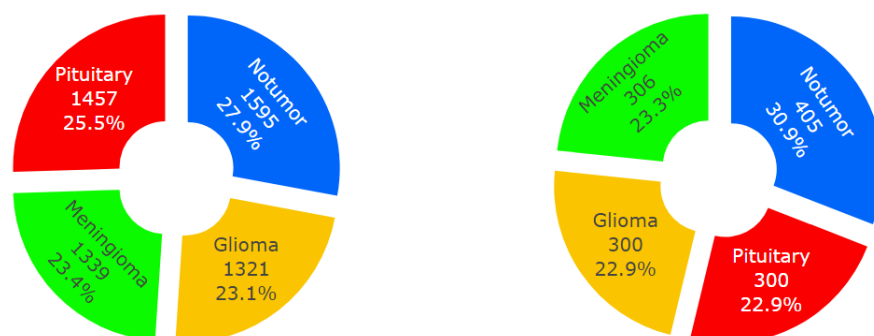


Рисунок 2.8 – Розподіл на класи у навчальній(зліва) й тестовій(справа) вибірці

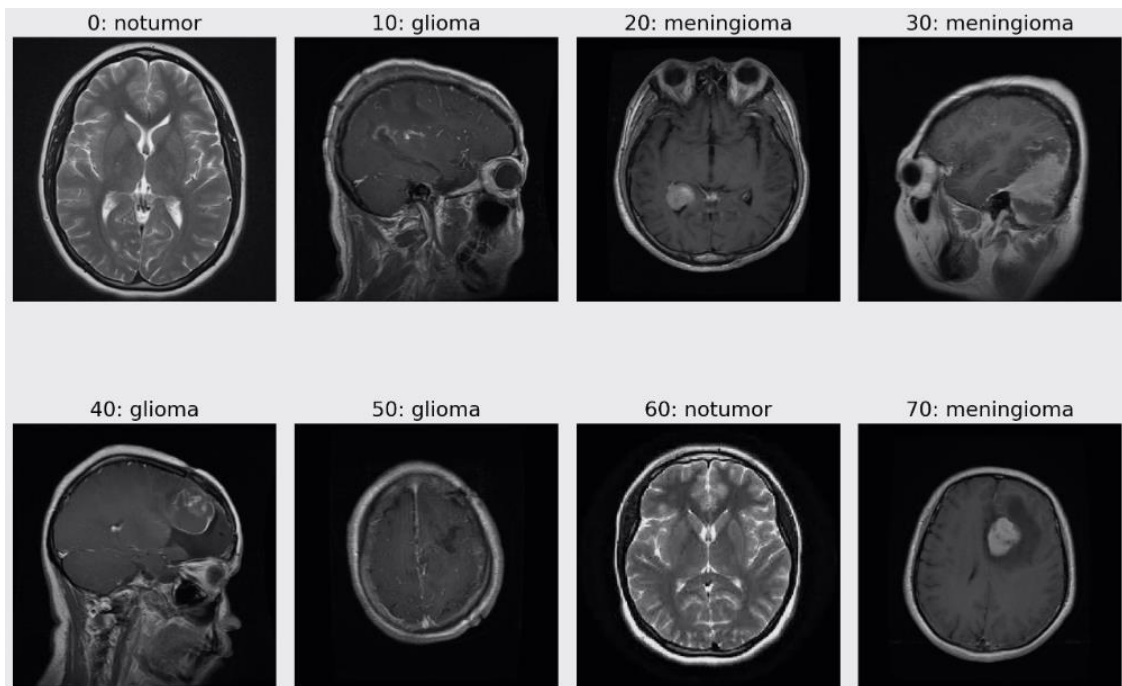


Рисунок 2.9 – Приклади зображень з набору даних

2.3 Метод діагностування пухлин головного мозку

Задача діагностування полягає у визначенні наявності об'єкта (пухлини) на зображенні. На вхід подаються зображення у вигляді масивів пікселів (значень інтенсивності кожного каналу зображення), які описуються цілими числами у діапазоні від 0 до 255. На виході повертається номер класу діагнозу з наступного переліку:

- 0 – glioma;
- 1 – meningioma;
- 2 – no tumor;
- 3 – pituitary.

Перелік діагнозів для розпізнавання може бути розширений за наявності відповідних зображень у вибірці даних.

Для вирішення поставленої задачі виділено наступні етапи:

- формування вибірки даних;
- побудова моделі нейронної мережі;
- навчання моделі;

- перевірка точності розпізнавання моделі.

На першому етапі треба сформувати та підготувати вибірку даних. Для цього використовуються генератори даних, що забезпечують динамічне завантаження та обробку зображень. Це дозволяє ефективно управляти пам'яттю та, за необхідністю, збільшувати обсяг навчальної вибірки за допомогою аугментації даних.

Для аугментації використовуються наступні параметри:

- зміна масштабу (rescale);
- випадкове обертання (rotation_range);
- зміна яскравості (brightness_range);
- зсув по ширині та висоті (width_shift_range, height_shift_range);
- зсув по зсуву (shear_range);
- горизонтальне відображення (horizontal_flip) .

На другому етапі потрібно побудувати модель нейронної мережі. Було прийнято рішення модифікувати архітектуру AlexNet. Основні зміни включають:

- зменшення кількості згорткових шарів. Модифікована архітектура має три згорткових шара порівняно з п'ятьма в оригінальній AlexNet;
- використання більших ядер (12x12, 6x6) та більших кроків (5x5, 2x2) у перших шарах зроблено для швидшого захоплення особливостей зображення;
- використання додаткових шарів нормалізації (BatchNormalization) для стабілізації та прискорення навчання;
- використання меншої кількості нейронів у повнозв'язних шарах. Замість двох шарів з 4096 нейронами, модифікована модель використовує один шар з 1028 нейронами;
- зменшення загальної кількості параметрів. Модифікована модель має 888,024 параметри порівняно з оригінальною AlexNet, що значно зменшує обчислювальні ресурси, необхідні для її тренування.

На рис. 2.10 наведена структура модифікованої моделі.

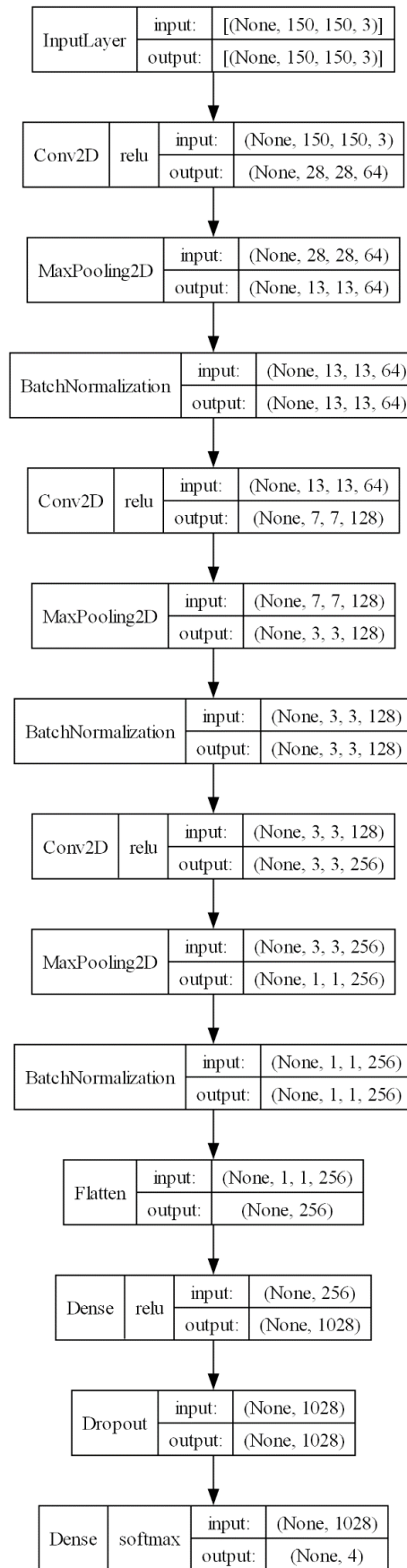


Рисунок 2.10 – Структура модифікованої моделі

На третьому етапі проводиться початкове навчання моделі. Модель навчається за допомогою оптимізатора Adam, з використанням категоріальної кросентропії як функції втрат. Процес навчання включає кілька епох, в кожній з яких модель оновлює свої ваги на основі градієнтів, обчислених на навчальних даних.

Для підвищення ефективності навчання та уникнення перенавчання застосовуються наступні зворотні виклики (callbacks):

- EarlyStopping – для ранньої зупинки навчання якщо точність не збільшується тривалий час;
- ReduceLROnPlateau – зменшення швидкості навчання, якщо помилка на валідаційній вибірці не зменшується тривалий час.

Після завершення навчання модель оцінюється на тестовій вибірці для перевірки її точності. Детальніше про результати тестування описано в пункті 5.4.

Після оцінки моделі вона може бути використана для діагностування пухлин головного мозку за наступним алгоритмом:

- лікар завантажує зображення МРТ головного мозку у систему;
- зображення проходить попередню обробку: змінюється його розмір, нормалізується та перетворюється у багатомірний масив;
- підготовлений масив передається у модель нейронної мережі, яка повертає номер класу, до якого належить зображення;
- за отриманим номером визначається назва діагнозу;
- отриманий діагноз з відповідним зображенням виводиться на екран для лікаря.

2.4 Висновки до другого розділу

У даному розділі було розглянуто існуючі архітектури згорткових нейронних мереж та виконано порівняння основних характеристик цих архітектур.

Для розробки програмного застосунку було вирішено модифікувати архітектуру AlexNet, додаючи нові шари та застосовуючи техніки регуляризації. Для тренування та тестування моделі вирішено використовувати набір даних з відкритого джерела Kaggle, що містить МРТ-знімки різних видів пухлин головного мозку.

Наукова новизна роботи полягає у запропонованому методі діагностування пухлин головного мозку, який характеризується використанням модифікованої архітектури AlexNet. Ця модифікація дозволяє значно зменшити кількість налаштовуваних параметрів, що спрощує вимоги до навчання моделі, при цьому забезпечуючи покращення точності діагностування.

3 ПРОЄКТУВАННЯ ПРОГРАМИ

В даному розділі має бути розглянуто основні особливості програмних засобів, за допомогою яких буде розроблятися програмне забезпечення.

3.1 Вибір мови програмування

Python – найкращий вибір для побудови моделей машинного навчання завдяки простоті використання, великій бібліотеці фреймворків та легкості використання [19].

Python надає високу потужність та універсальність середовища машинного навчання. Простий синтаксис мови спрощує перевірку даних і впорядковує процеси вилучення, обробки, уточнення, очищення, впорядкування та аналізу, тим самим роблячи співпрацю з іншими програмістами меншою перешкодою [19].

Python також надає широкую екосистему бібліотек, які позбавляють розробників монотонної рутинної роботи з написання функцій, звільняючи їх від необхідності писати вже відомі й розібрані алгоритми, зменшуючи ймовірність помилок під час програмування й дозволяючи зосередитись на реалізації нової функціональності програмного проєкту. Для нейронних мереж існують бібліотеки TensorFlow, Keras, для веброзробки Flask та Django. Вони являють собою потужні інструменти для розробки поставленого проєкту.

Python може працювати на будь-якій операційній системі, включаючи найпопулярніші: Windows, macOS, Linux. Також важливим фактором є величезна спільнота Python розробників, що робить процес пошуку нової інформації й розв'язання можливих проблем зручнішим.

У табл. 3.1 наведено порівняння обраної мови програмування з іншими найпопулярнішими мовами програмування.

Таблиця 3.1 – Порівняння мов програмування

Критерій порівняння	Python	JavaScript	Java	C#
Складність вивчення	Низька	Середня	Середня	Висока
Швидкість виконання	Середня	Середня	Висока	Висока
Підтримка та спільнота	Дуже висока	Дуже висока	Висока	Висока
Підтримка нейронних мереж	Висока (TensorFlow, Keras, PyTorch)	Середня (TensorFlow.js)	Середня (DeepLearning4J)	Середня (ML.NET)
Підтримка вебфреймворків	Висока (Flask, Django)	Висока (Express.js)	Висока (Spring, Vaadin)	Висока (ASP.NET)
Кросплатформність	Висока	Висока	Висока	Висока

3.2 Вибір середовища розробки

Visual Studio Code, також відомий як VSCode – це редактор вихідного коду, розроблений компанією Microsoft для Windows, Linux та macOS [20]. Його можливості включають в себе підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, рефакторинг коду та вбудований Git. У даного редактора присутня можливість змінювати тему, комбінації клавіш, налаштування та встановлювати величезну кількість розширень, які додають функціональність.

В опитуванні розробників Stack Overflow 2023 Visual Studio Code був визнаний найпопулярнішим інструментом середовища розробників серед 86 544 респондентів, 73,71% з яких повідомили, що використовують його. Опитування також показало, що Visual Studio Code більше використовують ті, хто вчиться програмувати, ніж професійні розробники (78% проти 74%) [20].

Jupyter notebook – це інструмент з відкритим вихідним кодом, який є важливою частиною науки про дані. Він забезпечує інтерактивне середовище і полегшує інтерактивне дослідження даних, сприяючи відтворюваності

досліджень, а також співпраці та обміну знаннями. Він пропонує єдиний документ, який містить математичні рівняння, описовий текст, візуалізації та будь-які інші мультимедійні матеріали. Це дозволяє користувачам зробити свою роботу легко зрозумілою та доступною для інших, розробляючи та візуалізуючи результати, додаючи інформацію, діаграми та формули. Клітинки блокнота дозволяють писати та виконувати фрагменти коду, візуалізувати дані та документувати свої міркування в режимі реального часу, що полегшує швидке експериментування з різними алгоритмами ML та налаштуваннями гіперпараметрів [21].

У табл. 3.2 наведено порівняння популярних IDE для розробки на Python.

Таблиця 3.2 – Порівняння середовищ розробки

Критерій порівняння	VSCode	PyCharm	JupyterNotebook
Підтримка розробки на різних мовах програмування	Висока	Середня	Низька
Легкість налаштування	Висока	Середня	Висока
Підтримка та спільнота	Висока	Висока	Висока
Кросплатформність	Висока	Висока	Висока
Інтеграція з системами контролю версій	Висока	Висока	Низька
Підтримка розширень та плагінів	Висока	Висока	Низька

Основним середовищем розробки було обрано VSCode, проте, на етапі розробки нейронної мережі й експериментів для покращення результатів буде використано середовище розробки JupyterNotebook через його можливість в виконання різних комірок коду зі збереженням результату між виконаннями.

3.3 Вибір вебфреймворку

У табл. 3.3 наведено порівняння двох найпопулярніших вебфреймворків Django та Flask для розробки вебінтерфейсу на Python.

Таблиця 3.3 – Порівняння вебфреймворків Django та Flask

Критерій порівняння	Django	Flask
Рівень складності	Висока	Середня
Масштабованість	Висока	Середня
Швидкість розробки	Низька	Висока
Інтеграція з ORM	Django ORM	SQLAlchemy
Вбудовані можливості	Велика кількість вбудованих функцій, включаючи аутентифікацію, адмін панель тощо	Мінімальні вбудовані можливості, що дозволяють використовувати тільки те, що потрібно
Ресурсоемність	Висока	Низька
Спільнота та підтримка	Висока	Висока

Для розробки вебінтерфейсу проєкту було обрано вебфреймворк Flask, оскільки цей фреймворк надає велику гнучкість без додаткової складної функціональності, яка часто є частиною більш великих фреймворків, таких як Django. Flask дозволяє легко інтегрувати необхідні бібліотеки для роботи з нейронними мережами, такі як TensorFlow чи PyTorch, не обтяжуючи розробку великою кількістю непотрібних залежностей. Завдяки своїй мінімалістичній природі, Flask забезпечує більш високу швидкість відгуку застосунку, що є важливим для функцій, які передбачають обробку обчислювано складних алгоритмів в реальному часі.

Flask також сприяє швидкому розвитку проєкту, надаючи можливість додавати нові функції без переосмислення або значних змін вже існуючої архітектури. Ці особливості розглянутого вебфреймворку стали ключовими при виборі. Такий підхід дозволяє зосередитися на суті задачі, не вдаючись до надмірної складності, яка може бути характерною для більш великих фреймворків, при цьому залишаючись гнучкою, та здатною до подальшого розширення.

3.4 Вибір бази даних

У табл. 3.4 наведено порівняння SQLite, MySQL та PostgreSQL.

Таблиця 3.4 – Порівняння баз даних

Критерій порівняння	SQLite	MySQL	PostgreSQL
Легкість налаштування	Висока	Низька	Середня
Продуктивність	Низька (для великих даних)	Висока	Висока
Масштабованість	Низька	Висока	Висока
Гнучкість у зміні схеми	Середня	Низька	Висока
Вимоги до ресурсів	Низькі	Середні	Середні
Підтримка ORM	Висока (SQLAlchemy)	Висока (SQLAlchemy)	Висока (SQLAlchemy)

Для розробки програмного продукту було обрано SQLite як основну базу даних. Основними перевагами цього вибору є легкість використання та налаштування, що дозволяє швидко розпочати розробку та тестування без необхідності складної конфігурації. SQLite не потребує встановлення окремого серверного програмного забезпечення, що зменшує накладні витрати на розгортання. Крім того, використання SQLAlchemy як ORM дозволяє зберегти абстракцію від конкретної бази даних, забезпечуючи можливість легкої міграції до інших баз даних (наприклад, MySQL або PostgreSQL) шляхом зміни конфігураційного URI. Це забезпечує гнучкість та масштабованість системи в майбутньому, якщо виникне потреба в зміні бази даних для обробки більшого обсягу даних або підвищення продуктивності.

3.5 Проєктування структури бази даних

Для зберігання даних даного проєкта було спроектовано БД, що

складається з наступних таблиць:

- Users містить основну інформацію про всіх користувачів системи, незалежно від їх ролі (лікар чи пацієнт) (табл. 3.5);
- Doctors містить інформацію про лікарів (табл. 3.6);
- Patients містить інформацію про пацієнтів (табл. 3.7);
- MRIScans містить інформацію про завантажені знімки (табл. 3.8);
- Conclusion містить висновки до знімків MPT (табл. 3.9);
- ErrorLog містить записи усіх помилок, які виникли (табл. 3.10).

Таблиця 3.5 – Сутність «Users»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
username	STRING	Унікальне ім'я користувача
password	STRING	Хешований пароль
email	STRING	Унікальна електронна пошта
role	ENUM	Роль користувача ('admin', 'doctor', 'patient')
created_at	DATETIME	Дата та час створення акаунту
is_active	BOOLEAN	Чи активний акаунт цього користувача

Таблиця 3.6 – Сутність «Doctors»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
user_id	INT	Зовнішній ключ до таблиці Users
first_name	STRING	Ім'я лікаря
last_name	STRING	Прізвище лікаря
birth_date	DATETIME	Дата народження
specialty	STRING	Спеціалізація лікаря

Таблиця 3.7 – Сутність «Patients»

Поле	Тип даних	Опис
1	2	3
id	INT	Первинний ключ, автоінкремент
user_id	INT	Зовнішній ключ до таблиці Users
first_name	STRING	Ім'я пацієнта

Продовження таблиці 3.7

1	2	3
last_name	STRING	Прізвище пацієнта
birth_date	DATETIME	Дата народження

Таблиця 3.8 – Сутність «MRIScans»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
patient_id	INT	Зовнішній ключ до таблиці Patients
file_name	STRING	Назва збереженого файлу
upload_date	DATETIME	Дата завантаження знімку
diagnosis	STRING	Результат діагностики
diagnosed_by	INT	Зовнішній ключ до таблиці Doctor

Таблиця 3.9 – Сутність «Conclusion»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
mri_scan_id	INT	Зовнішній ключ до таблиці MRIScans
doctor_id	INT	Зовнішній ключ до таблиці Doctors
conclusion	STRING	Висновок
created_at	DATETIME	Дата та час створення висновку

Таблиця 3.10 – Сутність «ErrorLog»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
message	STRING	Повідомлення про помилку
timestamp	DATETIME	Дата та час виникнення помилки

Зв'язки між таблицями:

- Users має зв'язок один до одного з таблицями Doctors та Patients через поле user_id;
- Patients має зв'язок один до багатьох з таблицею MRIScans через поле patient_id;
- Doctors має зв'язок один до багатьох з таблицею MRIScans через поле diagnosed_by;

- MRIScans має зв'язок один до багатьох з таблицею Conclusion через поле mri_scan_id;
- Doctors має зв'язок один до багатьох з таблицею Conclusion через поле doctor_id.

Схему бази даних та зв'язки між сутностями наведено на рис. 3.1.

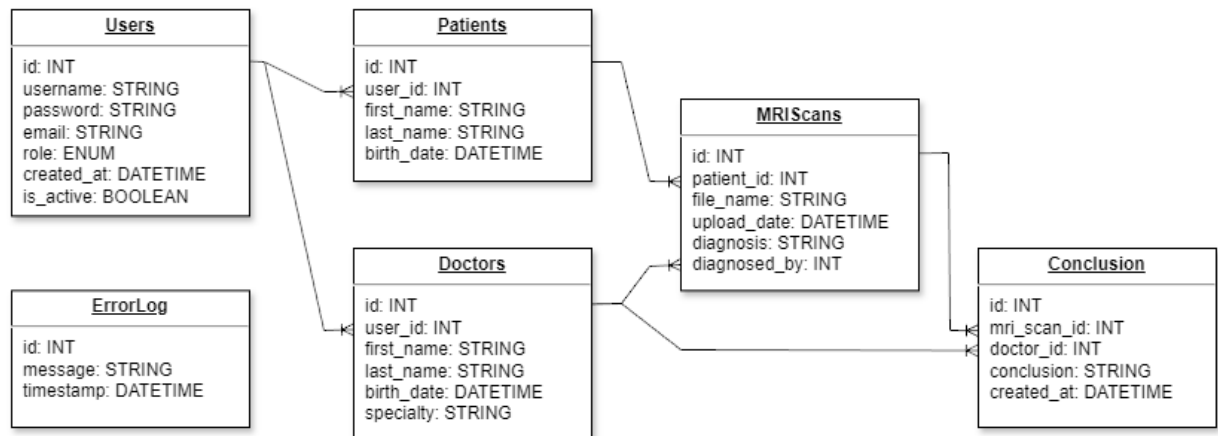


Рисунок 3.1 – Схема бази даних та зв'язки між сутностями

3.6 Архітектура програмного забезпечення

Для розробки програмного продукту буде використано архітектурний шаблон MVC, який підтримується вебфреймворком Flask. Використання архітектури MVC забезпечує модульність й зручність у розробці та тестуванні. Архітектурний шаблон складається з трьох компонентів:

а) Model (Модель):

1) опис: Модель відповідає за управління даними застосунку. Вона включає логіку для роботи з базою даних, здійснює маніпуляції даними та бізнес-логіку;

2) реалізація: У нашому випадку модель буде включати нейронну мережу, розроблену за допомогою TensorFlow, яка виконує класифікацію знімків МРТ. Знімки й дані про них зберігаються в базі даних;

б) View (Подання):

1) опис: Подання відповідає за відображення інформації користувачу. Воно отримує дані від Контролера і відображає їх у зручному для сприйняття вигляді;

2) реалізація: Вебінтерфейс, який буде створено за допомогою HTML, CSS та JavaScript, що буде дозволяти лікарям завантажувати знімки МРТ та переглядати результати класифікації;

в) Controller (Контролер):

1) опис: Контролер відповідає за обробку запитів від користувачів, взаємодію з моделлю та вибір відповідного подання для відображення;

2) реалізація: У Flask контролери реалізуються у вигляді маршрутів (routes). Кожен маршрут відповідає за обробку певного типу запитів, викликає необхідні методи моделі для обробки даних і визначає, яке подання слід використовувати для відповіді.

На рис. 3.2 зображено архітектуру застосунку на основі шаблону проєктування MVC.

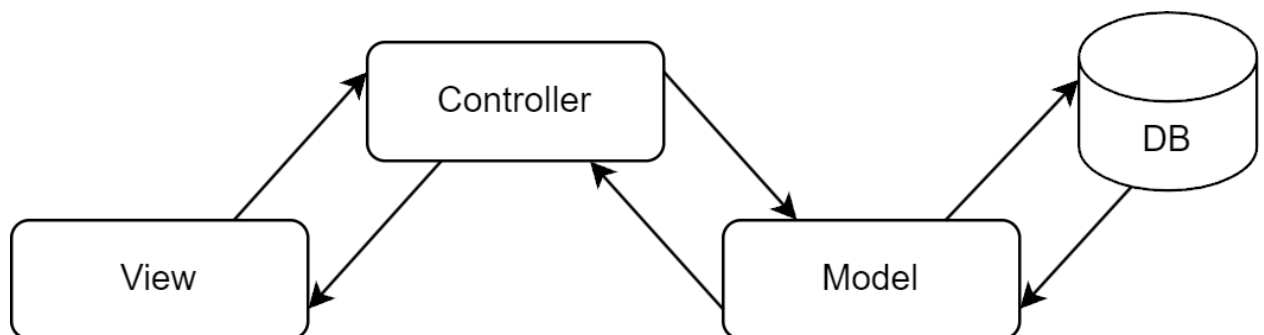


Рисунок 3.2 – Архітектура застосунку

3.7 Визначення функціональних вимог до застосунку

Для програмного забезпечення було визначено наступні функціональні вимоги.

Вимоги лікаря:

– авторизація;

- перегляд МРТ знімків пацієнтів;
- пошук МРТ знімків по пацієнту;
- діагностика знімку;
- пакетна діагностика знімків;
- завантаження МРТ знімку з комп'ютера;
- пакетне завантаження МРТ знімків;
- розпізнавання знімків;
- редагування діагнозу МРТ знімка;
- збереження МРТ знімку в БД;
- формування висновків до МРТ знімків;
- вихід з особистого кабінету.

Вимоги пацієнта:

- реєстрація;
- авторизація;
- перегляд власних МРТ знімків;
- перегляд висновків лікаря до знімку;
- вихід з особистого кабінету.

Вимоги адміністратора:

- авторизація;
- перегляд всіх користувачів;
- редагування даних користувача;
- видалення користувача;
- реєстрація нових лікарів;
- перегляд журналу помилок;
- ініціювати перенавчання нейронної мережі;
- зміна архітектури й ваг моделі для класифікації;
- вихід з панелі адміністратора.

Діаграма прецедентів для лікаря наведена на рис. 3.3.

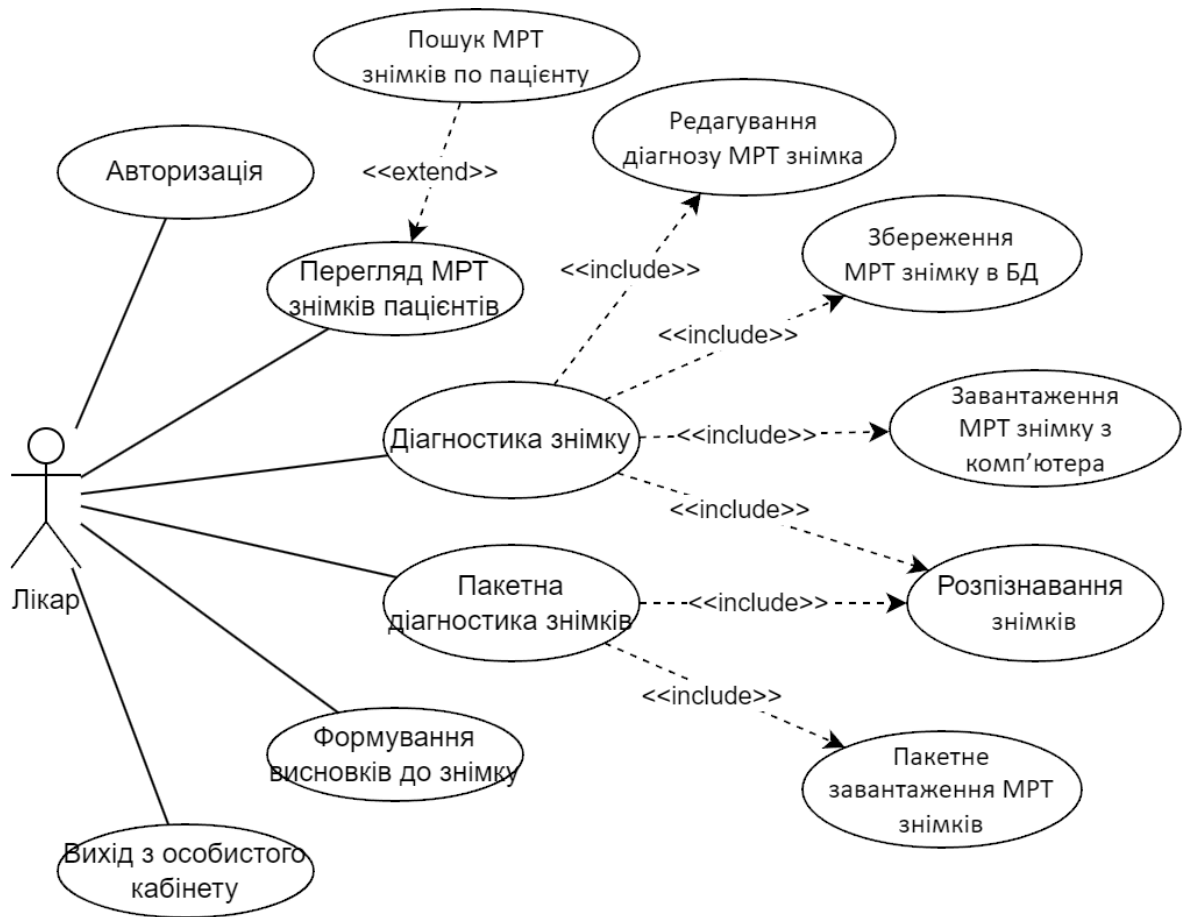


Рисунок 3.3 – Діаграма прецедентів для лікаря

Діаграма прецедентів для пацієнта наведена на рис. 3.4.

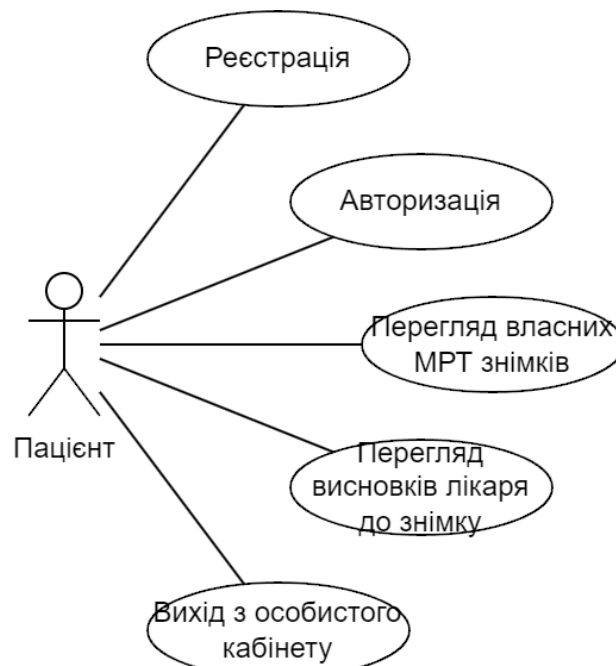


Рисунок 3.4 – Діаграма прецедентів для пацієнта

Діаграма прецедентів для адміністратора наведена на рис. 3.5.



Рисунок 3.5 – Діаграма прецедентів для адміністратора

3.8 Висновки до третього розділу

У даному розділі було розглянуто обрані засоби, інструменти й методи для розробки програмного забезпечення, а саме:

- було обрано мову програмування Python та бібліотеки Tensorflow, Keras для реалізації нейромережних методів;
- було обрано середовище розробки VSCode для основної розробки програмного застосунку та JupyterNotebook для розробки нейромережних методів й експериментів для підвищення результатів;

- було обрано вебфреймворк Flask для розробки вебінтерфейсу проєкту;
- для роботи з базою даних було визначено СКБД SQLite та ORM SQLAlchemy для збереження абстракції й полегшення подальшої міграції до іншої СКБД при необхідності.

Також було розглянуто структуру бази даних та визначено функціональні вимоги до застосунку.

4 РОЗРОБКА ПРОГРАМИ

4.1 Алгоритм функціонування застосунку

На рис. 4.1 зображено схему функціонування застосунку, яка побудована на основі функціональних вимог, визначених у попередньому розділі.

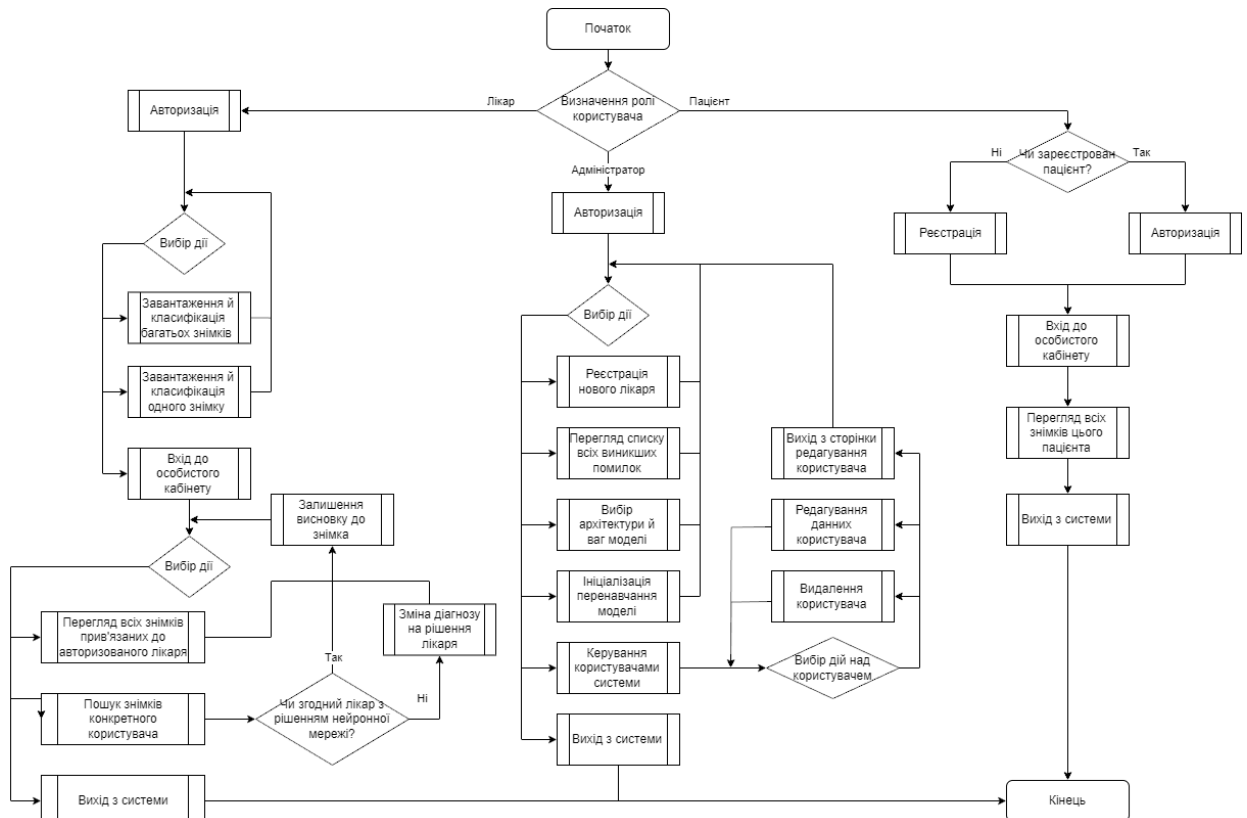


Рисунок 4.1 – Схему функціонування застосунку

Спочатку роботи з системою користувач, якщо він зареєстрований, повинен перейти до сторінки авторизації та ввести свої дані для входу в систему. Якщо користувач є пацієнтом та не є зареєстрованим, йому необхідно пройти процедуру реєстрації, створюючи новий обліковий запис.

Після успішної авторизації система визначає роль користувача: лікар, пацієнт або адміністратор.

Якщо користувач є пацієнтом, він потрапляє до свого особистого кабінету, де може переглядати всі свої знімки.

Якщо користувач є лікарем, то йому доступний такий вибір дій:

- лікар може завантажити один знімок. При виборі знімку також вказується якого пацієнта цей знімок. Після завантаження знімку система з допомогою визначеної нейронної мережі класифікує, чи є пухлина на знімку, відображає результат лікарю, та зберігає його в базу даних. Лікар може одразу після цього залишити свій висновок до знімку;

- лікар може завантажити багато знімків одночасно. Після завантаження система з допомогою визначеної нейронної мережі класифікує, чи є пухлина на цих знімках, й відображає результат для кожного знімку лікарю;

- лікар може увійти до свого особистого кабінету, де він може переглянути знімки всіх його пацієнтів, чи виконати пошук конкретного пацієнта, й переглядати тільки його знімки. Знайшовши необхідний знімок, лікар може або видалити його, або написати висновок для пацієнта. При написанні висновка в лікаря є можливість змінити діагноз з того, що розпізнала нейронна мережа, на той, що він вважає правильним.

Якщо користувач є адміністратором, то йому доступний такий вибір дій:

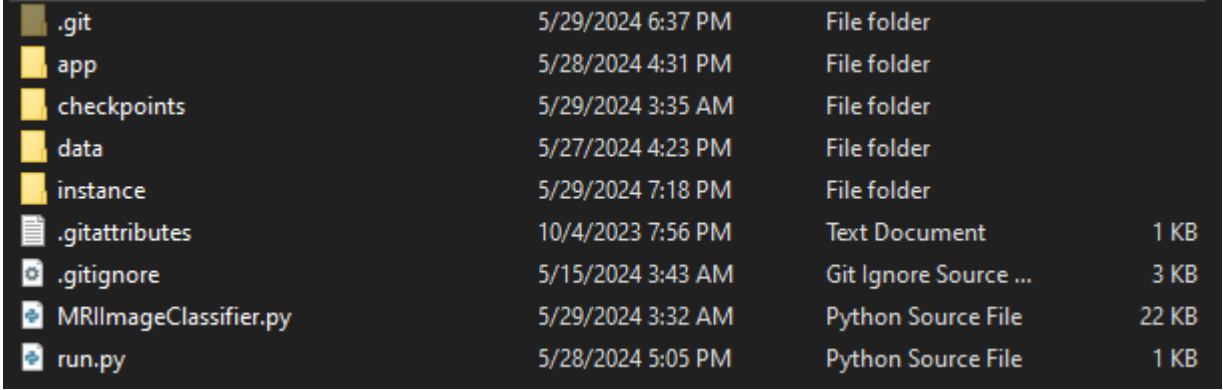
- реєстрація нового лікаря в системі;
- перегляд списку всіх помилок, які виникли у системі;
- керування користувачами системи. Адміністратор може редагувати дані користувача, такі як ім'я, прізвище, email та username. Також в адміністратора є можливість або деактивувати акаунт користувача, або видалити його з системи;

- ініціалізувати перенавчання обранної нейронної мережі;
- обирати архітектуру й ваги моделі нейронної мережі в системі.

Усі сторінки вебзастосунку поєднані між собою, тому користувач може легко переходити від однієї сторінки до іншої. Робота з системою закінчується, коли користувач вийшов з облікового запису.

4.2 Структура програмного застосунку

Коренева директорія проєкту наведена на рис. 4.2.



.git	5/29/2024 6:37 PM	File folder	
app	5/28/2024 4:31 PM	File folder	
checkpoints	5/29/2024 3:35 AM	File folder	
data	5/27/2024 4:23 PM	File folder	
instance	5/29/2024 7:18 PM	File folder	
.gitattributes	10/4/2023 7:56 PM	Text Document	1 KB
.gitignore	5/15/2024 3:43 AM	Git Ignore Source ...	3 KB
MRImageClassifier.py	5/29/2024 3:32 AM	Python Source File	22 KB
run.py	5/28/2024 5:05 PM	Python Source File	1 KB

Рисунок 4.2 – Коренева директорія проєкту

Директорія `.git` містить всю інформацію, яка необхідна для проєкту з боку управління версіями, а також всю інформацію про комміти, адресу віддаленого сховища і т.д.

Директорія `data` зберігає навчальні та тестові дані, що використовуються для тренування та тестування нейронної мережі.

Директорія `checkpoints` зберігає збережені моделі (чекпойнти), які можуть бути використані для завантаження та продовження навчання або класифікації.

Директорія `instance` використовується для зберігання бази даних.

Файл `.gitattributes` – Git файл, який потрібен щоб вказати кодування файлів як у сховищі Git, так і в робочому каталозі.

Файл `.gitignore` – визначає навмисно невідстежувані файли, які Git має ігнорувати.

Файл `run.py` – основний скрипт для запуску Flask-застосунку.

Файл `MRImageClassifier` містить однойменний клас, призначений для класифікації зображень МРТ головного мозку за допомогою різних архітектур глибоких нейронних мереж. Цей клас містить такі методи:

- `__init__()` – конструктор класу, що ініціалізує параметри та налаштовує оточення;
- `setup_environment()` – метод для налаштування параметрів Tensorflow, дозволяючи йому динамічно виділяти пам'ять на GPU, що забезпечує ефективне використання ресурсів;
- `set_network_name()` – метод для встановлення назви мережі;
- `load_model_from_checkpoint()` – метод для завантаження ваг моделі з файлу чекпоінта;
- `classify_image()` – метод для класифікації зображення за допомогою навченої моделі;
- `train()` – метод для тренування моделі;
- `plot_training_history()` – метод для візуалізації історії тренування моделі;
- `plot_confusion_matrix()` – метод для візуалізації матриці помилок;
- `plot_data_distribution()` – метод для візуалізації розподілу даних;
- `_get_model()` – метод, що ініціалізує модель нейронної мережі за її назвою.

Директорія `app` містить усю вебчастину проєкту. На рис. 4.3 наведено її структуру.

	<code>static</code>	5/14/2024 8:33 AM	File folder	
	<code>templates</code>	5/25/2024 2:35 PM	File folder	
	<code>uploads</code>	5/29/2024 6:17 PM	File folder	
	<code>__init__.py</code>	5/14/2024 8:32 AM	Python Source File	1 KB
	<code>config.py</code>	5/28/2024 4:37 PM	Python Source File	1 KB
	<code>forms.py</code>	5/29/2024 6:32 PM	Python Source File	6 KB
	<code>globals.py</code>	5/29/2024 3:32 AM	Python Source File	1 KB
	<code>models.py</code>	5/29/2024 6:30 PM	Python Source File	4 KB
	<code>routes.py</code>	5/29/2024 6:31 PM	Python Source File	22 KB

Рисунок 4.3 – Структура папки `app`

Директорія `static` містить файли стилів.

Директорія templates містить HTML-шаблони для різних сторінок вебзастосунку. Кожен шаблон відповідає за відображення відповідних даних та взаємодію з користувачами.

Директорія uploads відповідає за зберігання завантажених МРТ-знімків.

Файл __init__.py призначений для ініціалізації Flask-застосунка.

Файл config.py містить налаштування конфігурацій застосунка, включаючи налаштування бази даних та секретні ключі.

Файл forms.py містить визначення форм. Цей файл містить класи для таких форм:

- UploadForm – форма, що дозволяє користувачам завантажувати один файл зображення для класифікації та вказувати інформацію про пацієнта, щоб зберегти його зображення у системі;
- BatchUploadForm – форма, що дозволяє користувачам одночасно завантажувати кілька файлів зображень для класифікації;
- SearchForm – форма, що дозволяє здійснювати пошук пацієнтів за їхнім іменем та прізвищем;
- ConclusionForm – форма, що дозволяє лікарям вводити висновки до знімків МРТ;
- DiagnosisForm – форма, що дозволяє лікарям змінювати діагноз пацієнта;
- LoginForm – форма призначена для входу в систему і вимагає від користувачів введення електронної пошти та пароля;
- RegisterPatientForm – форма призначена для реєстрації нових пацієнтів у системі та вимагає введення особистих та облікових даних;
- RegisterDoctorForm – форма призначена для реєстрації нових лікарів у системі та вимагає введення особистих даних, облікових даних та спеціальності;

– UserForm – форма, що дозволяє адміністратору оновлювати дані користувачів у системі, такі як ім'я, прізвище та електронна пошта, а також активувати або деактивувати їхні облікові записи.

Файл `models.py` визначає таблиці бази даних для зберігання інформації. Розроблені таблиці були описані в підрозділі 3.5.

Файл `routes.py` визначає маршрути для обробки HTTP-запитів. Кожен маршрут відповідає за рендеринг відповідних шаблонів та обробку даних від користувачів. Цей файл містить такі маршрути:

- «/» – перенаправляє на головну сторінку;
- «/home» – відображає головну сторінку застосунка;
- «/register» – дозволяє пацієнтам реєструватися у системі;
- «/login» – дозволяє користувачам увійти в систему;
- «/logout» – вихід з системи;
- «/profile» – відображає різний профіль користувача, залежно від ролі користувача в системі, пацієнт чи лікар;
- «/profile/<filename>» – відображає одне зображення МРТ та дозволяє лікарям додавати висновок та змінювати діагноз до нього;
- «/upload» – дозволяє лікарю завантажувати одне зображення МРТ та проводити його класифікацію;
- «/batch_upload» – дозволяє лікарю завантажувати кілька зображень МРТ одночасно та проводити їх класифікацію;
- «/admin» – відображає панель адміністратора;
- «/admin/users» – дозволяє адміністратору переглядати та управляти користувачами;
- «/admin/users/update/<int:user_id>» – дозволяє адміністратору оновлювати інформацію про користувача;
- «/admin/users/delete/<int:user_id>» – дозволяє адміністратору видаляти користувача з системи;
- «/admin/register_doctor» – дозволяє адміністратору реєструвати нових лікарів у системі;

- «/autocomplete» – надає автозаповнення для поля пошуку на сторінці профілю;
- «/mri_scans/<int:mri_scan_id>/conclusions» – надає висновки лікарів щодо конкретного зображення МРТ;
- «/delete_image» – дозволяє лікарю видаляти завантажене зображення;
- «/trigger_error» – викликає тестову помилку для перевірки обробки помилок;
- «/admin/errors» – дозволяє адміністратору переглядати помилки у системі, що виникли;
- «/admin/retrain» – дозволяє адміністратору почати перенавчання моделі нейронної мережі;
- «/admin/start-retrain» – шлях, який ініціалізує перенавчання моделі нейронної мережі;
- «/progress» – запит на прогрес перенавчання моделі нейронної мережі;
- «/retrain-results» – запит на результати перенавчання моделі;
- «/admin/select-checkpoint» – дозволяє адміністратору обирати архітектуру й ваги моделі нейронної мережі.

4.3 Висновки до четвертого розділу

У даному розділі було визначено алгоритм функціонування розроблюваного програмного застосунку.

Також було детально розглянуто основні папки, класи, та методи, які було створено для реалізації програмного застосунку, стосовно створеної структури.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис застосування програми

Розроблене програмне забезпечення призначене для медичних закладів та фахівців у галузі онкології та неврології. Воно пропонує автоматизовану підтримку лікарям щодо класифікації зображень МРТ для виявлення пухлин головного мозку.

5.2 Умови виконання програми

Для розгортання програмного забезпечення необхідно мати наявними такі компоненти:

- процесор Intel Core i7 або еквівалент;
- мінімум 8 Гб оперативної пам'яті, рекомендовано 16 й більше;
- відеокарта: рекомендована відеокарта з підтримкою CUDA;
- операційна система: Windows 10 або Ubuntu 20.04 LTS;
- інтерпретатор Python версії 3.8 або вище;
- встановлений фреймворк Flask та бібліотека Tensorflow;
- база даних SQLite.

Для коректної роботи програми на стороні клієнта (лікарів та пацієнтів) потрібно мати:

- процесор від 1.0 ГГц;
- обсяг оперативної пам'яті від 4 ГБ;
- веббраузер (Google Chrome, Mozilla Firefox, OperaGX та ін.).

5.3 Інструкція по експлуатації програми

При заході на сайт користувач потрапляє на головну сторінку (рис. 5.1). На цій сторінці розміщено невеликий текст, присвячений темі пухлин

головного мозку, який має на меті підтримати користувачів та надати їм впевненість у можливостях сучасної медицини.

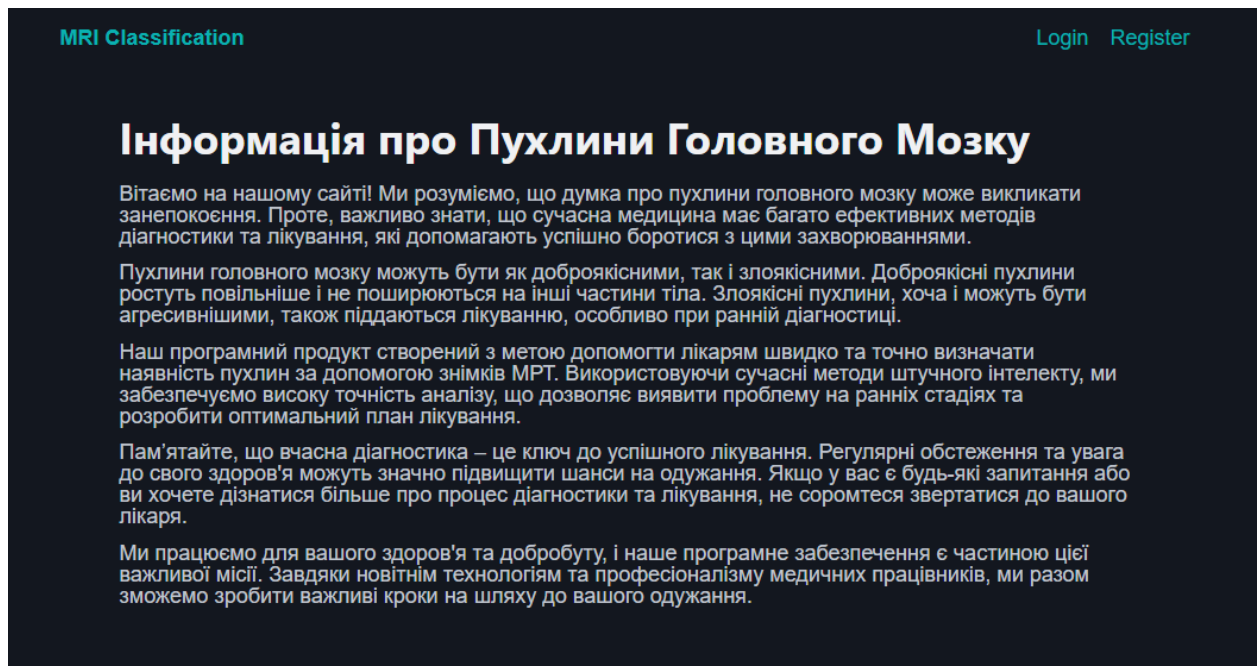


Рисунок 5.1 – Головна сторінка вебсайту

Якщо користувач є лікарем, і не є зареєстрованим, йому потрібно звернутися до адміністратора для реєстрації, оскільки реєструвати лікарів може виключно адміністратор. Це зроблено для того, щоб забезпечити належний контроль за доступом до системи і гарантувати, що лише кваліфіковані медичні працівники мають доступ до відповідальних відповідальних функцій.

Для пацієнтів, які ще не зареєстровані в системі, у верхній правій частині навігаційної панелі розміщено посилання «Register». Переходячи за цим посиланням, користувач потрапляє на сторінку реєстрації (рис. 5.2), де може заповнити необхідні дані для створення свого власного облікового запису.

The screenshot shows a dark-themed web interface for the 'MRI Classification' system. At the top left is the text 'MRI Classification' and at the top right are links for 'Login' and 'Register'. The main heading is 'Register Form'. Below it are several input fields: 'Email', 'Username', 'Password', 'Confirm Password', 'First Name', 'Last Name', and 'Birth Date'. The 'Birth Date' field includes a placeholder 'mm/dd/yyyy' and a calendar icon. At the bottom is a large teal button labeled 'Register'.

Рисунок 5.2 – Сторінка реєстрації пацієнта

Після успішного заповнення та відправки реєстраційної форми, програма автоматично перенаправляє користувача на сторінку «Login» (рис. 5.3), де він може ввести свої реєстраційні дані для входу в систему. І лікарі, і пацієнти, і адміністратор використовують цю сторінку для авторизації. Після введення своїх облікових даних користувачі отримують доступ до своїх профілів та функцій, відповідно до їх ролей у системі.

The screenshot shows the 'Login Form' in the 'MRI Classification' system. At the top left is 'MRI Classification' and at the top right are links for 'Login' and 'Register'. The main heading is 'Login Form'. Below it are two input fields: 'Email' and 'Password'. At the bottom is a large teal button labeled 'Login'.

Рисунок 5.3 – Сторінка авторизації

5.3.1 Інструкція з експлуатації програми для лікаря

Після успішного входу в систему лікар потрапляє на сторінку свого особистого кабінету (рис. 5.4). У верхній частині інтерфейсу розміщено чотири основних посилання для переходу до сторінок з відповідними назвами: «Profile», «Upload», «BatchUpload» та «Logout». За допомогою посилання «Profile» лікар може з будь-якої іншої сторінки повернутися до свого особистого кабінету.

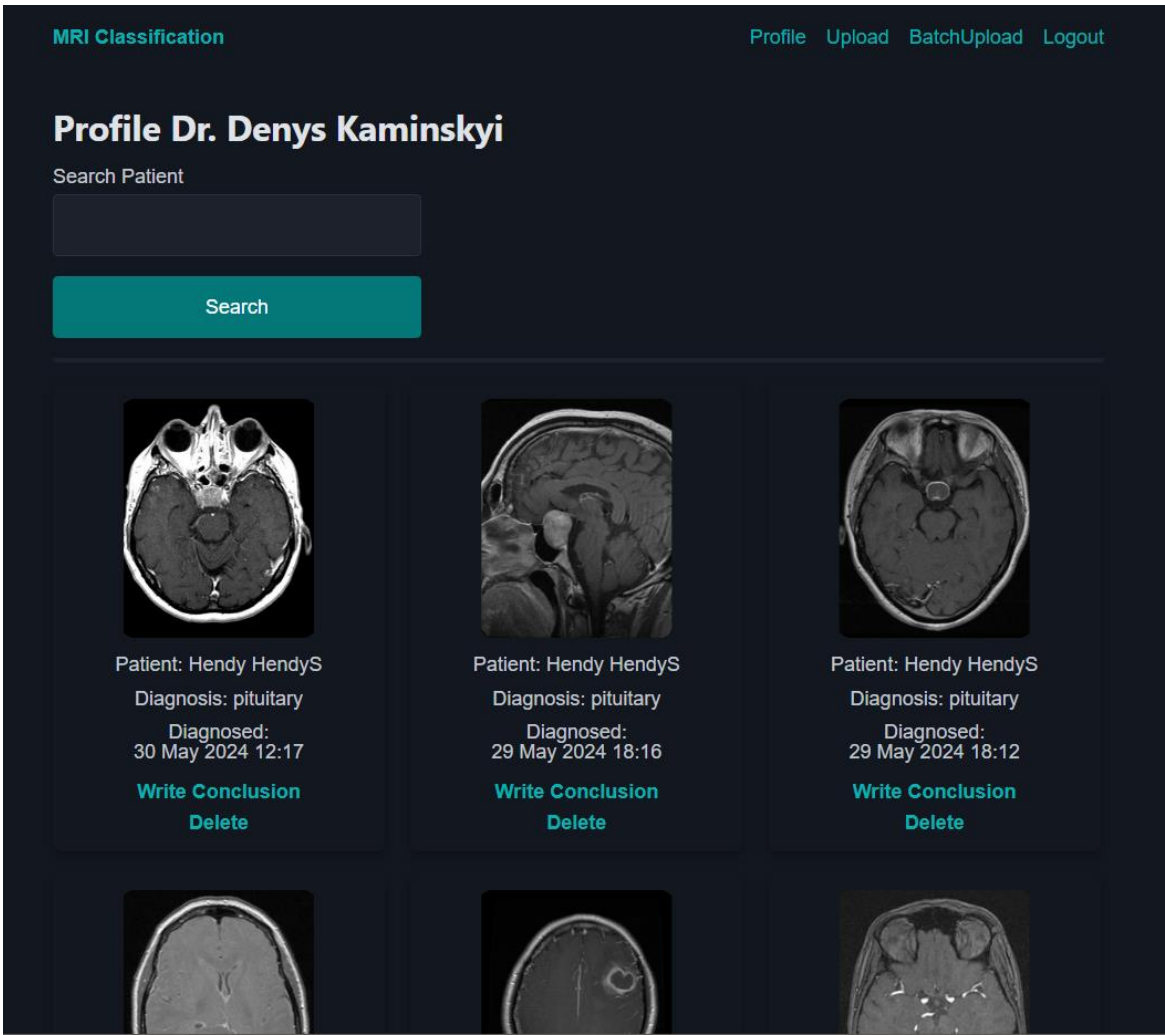


Рисунок 5.4 – Особистий кабінет лікаря

Особистий кабінет лікаря – це центральна сторінка для перегляду та керування знімками МРТ пацієнтів, які були завантажені в систему. Вгорі сторінки розміщена форма пошуку, що дозволяє знайти знімки конкретного

пацієнта за допомогою введення його імені або прізвища. Система підтримує функцію автозаповнення, що значно спрощує процес пошуку: при введенні декількох символів відображаються всі відповідні пацієнти (рис. 5.5).

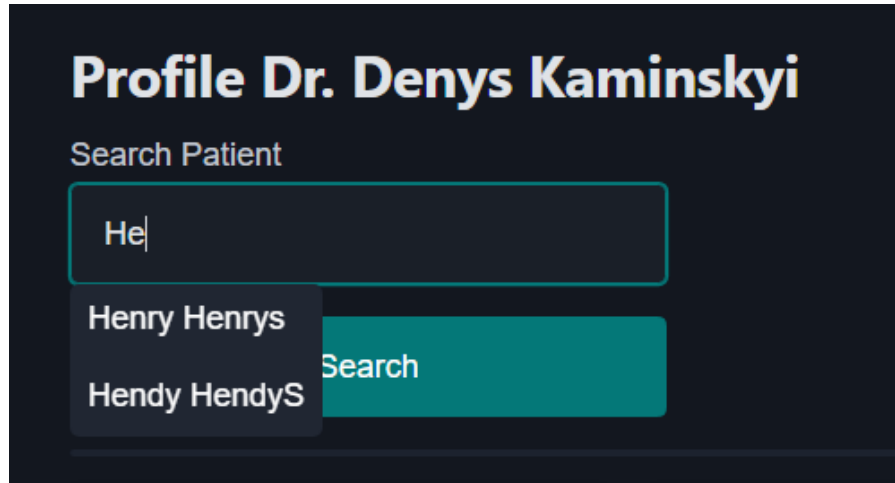


Рисунок 5.5 – Автозаповнення пошуку

На цій сторінці під кожним знімком МРТ міститься інформація про пацієнта: його ім'я та прізвище, діагноз, а також час встановлення діагнозу. Доступні дві основні дії: «Write conclusion» і «Delete». При натисканні кнопки «Delete» з'являється діалогове вікно з підтвердженням, що запитує лікаря, чи дійсно він бажає видалити знімок (рис. 5.6). У разі підтвердження знімок повністю видаляється з системи.

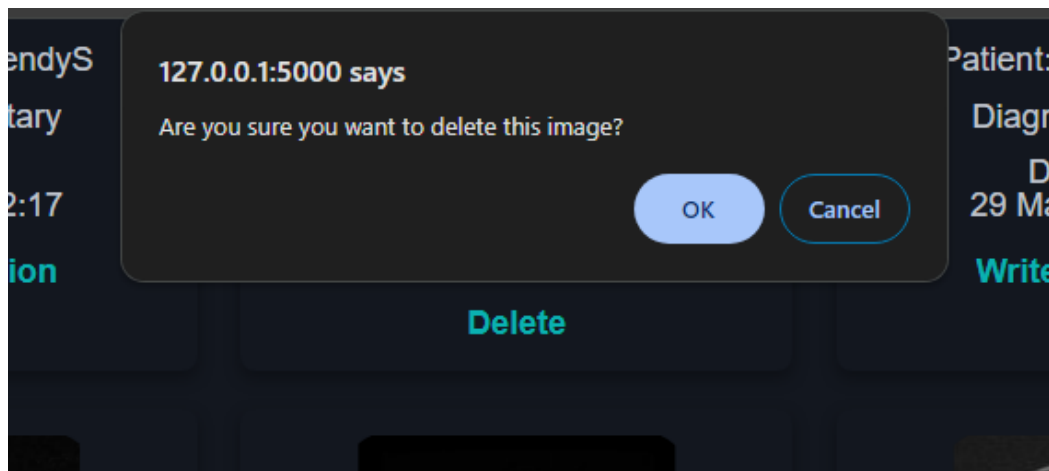
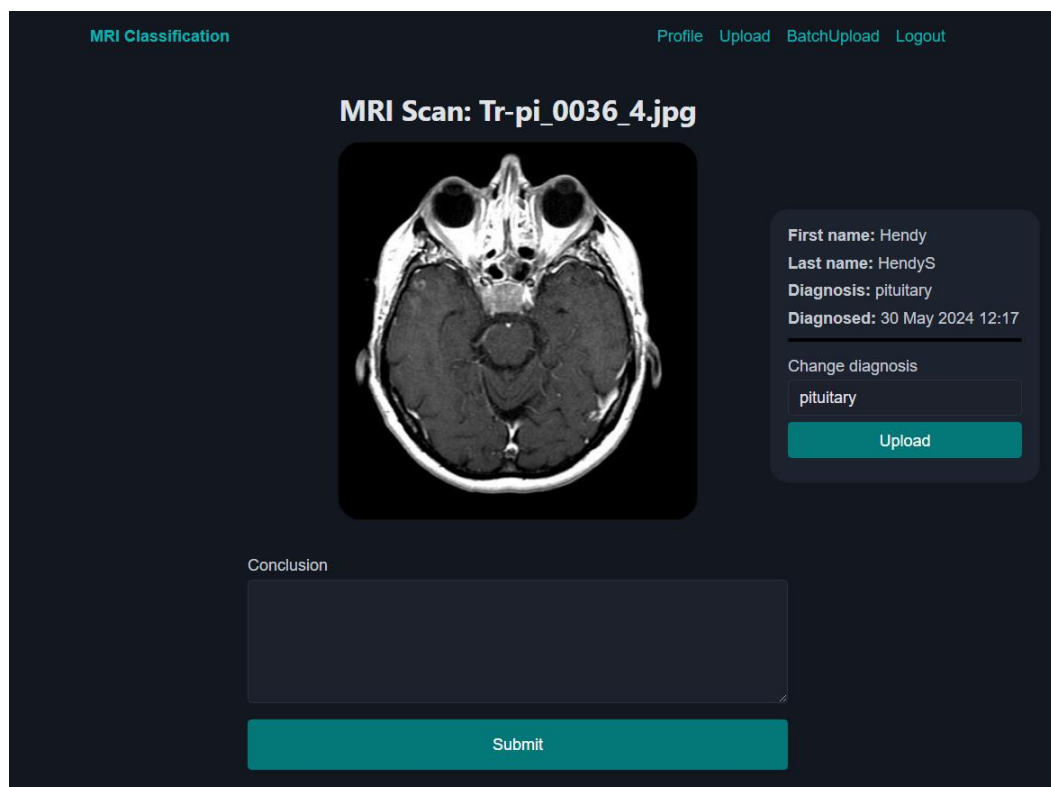


Рисунок 5.6 – Діалогове вікно з підтвердженням видалення знімку

Кнопка «Write conclusion» відкриває нову сторінку (рис. 5.7) зі збільшеним зображенням МРТ, де лікар може детально переглянути знімок. Праворуч від зображення розміщується панель з основною інформацією про пацієнта, включаючи ім'я та прізвище, діагноз та час діагнозу. В цій панелі також є поле для зміни діагнозу, якщо лікар вважає, що висновок нейронної мережі помилковий. За замовчування в це поле вписано діагноз, що збережено в базі даних. Після внесення змін лікар може натиснути кнопку «Upload», щоб зберегти новий діагноз у системі. Крім того, на цій сторінці знаходиться форма для залишення висновків та рекомендацій щодо серйозності пухлини та інших медичних аспектів. Після залишення висновку й натиснення кнопки «Submit» лікарю відобразиться його висновок з датою, коли його було залишено (рис.5.8)



The screenshot shows a web application interface titled "MRI Classification". At the top right, there are navigation links: "Profile", "Upload", "BatchUpload", and "Logout". The main heading is "MRI Scan: Tr-pi_0036_4.jpg". Below this is a large image of an axial MRI scan of a brain. To the right of the image is a panel containing patient information: "First name: Hendy", "Last name: HendyS", "Diagnosis: pituitary", and "Diagnosed: 30 May 2024 12:17". Below this information is a "Change diagnosis" section with a text input field containing "pituitary" and an "Upload" button. At the bottom of the page, there is a "Conclusion" section with a large text area for writing and a "Submit" button.

Рисунок 5.7 – Сторінка для залишення висновків до знімку

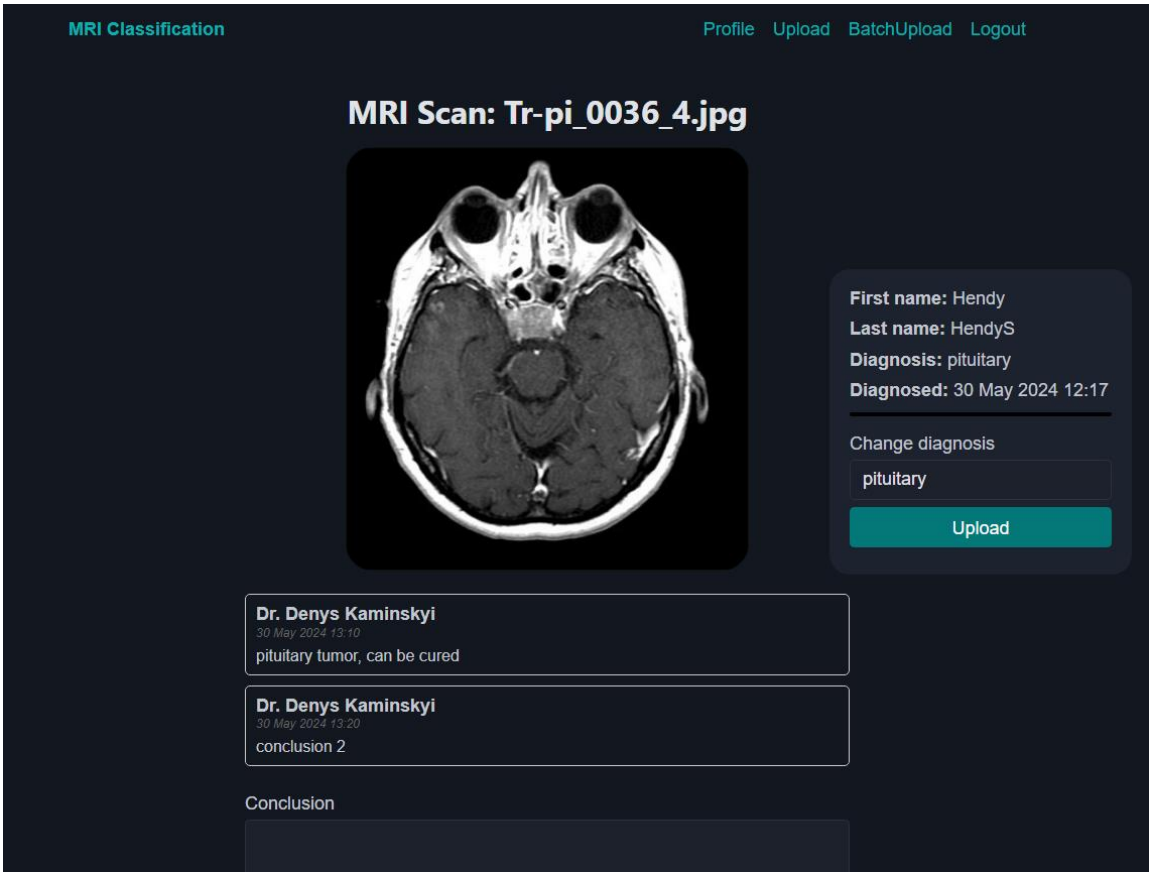


Рисунок 5.8 – Відображення висновку, який залишив лікар

Сторінка «Upload» (рис. 5.9) призначена для завантаження нових знімків МРТ пацієнтів. На лівій частині цієї сторінки розміщена форма для завантаження зображень, яка включає поле для пошуку пацієнта, вибір файлу (знімка МРТ) та кнопку «Upload». Пошукове поле підтримує функцію автозаповнення (рис. 5.10), яка пропонує варіанти пацієнтів під час введення імені або прізвища.

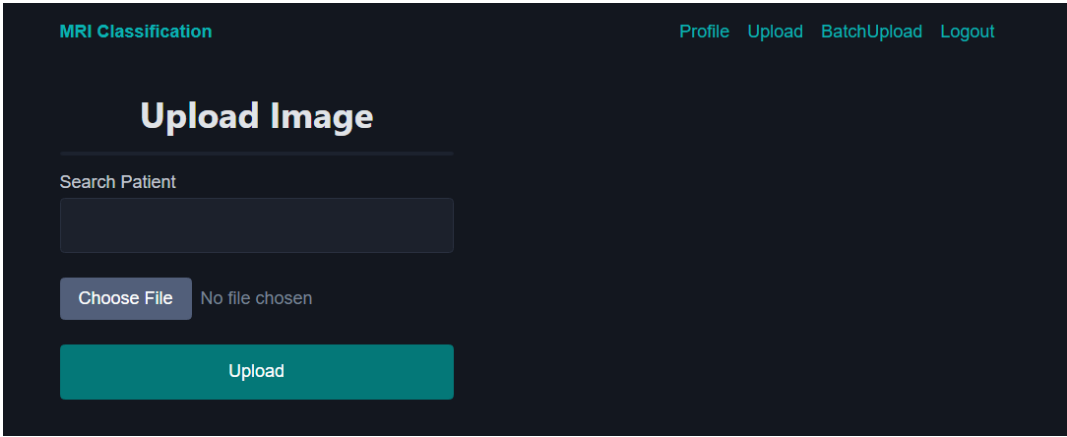
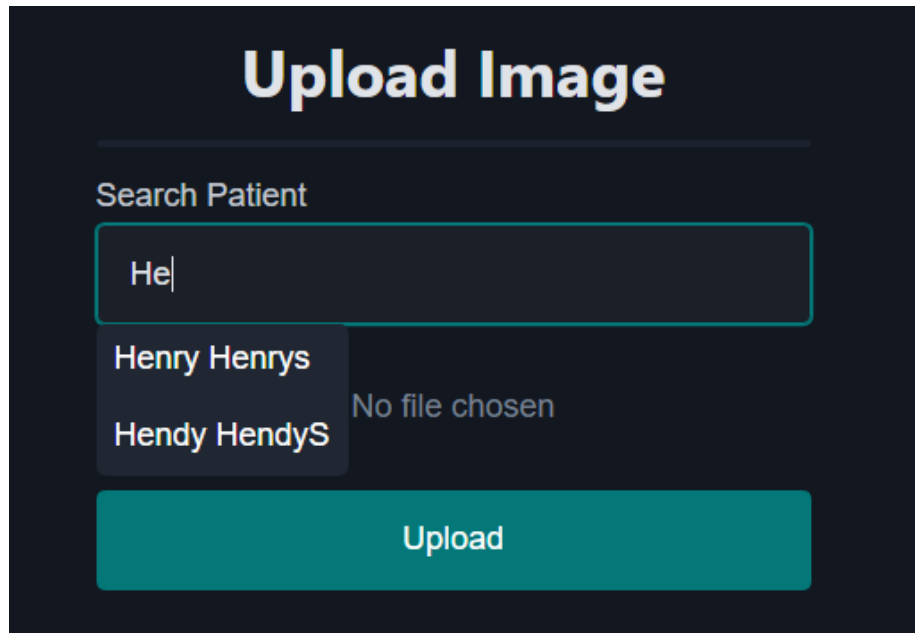


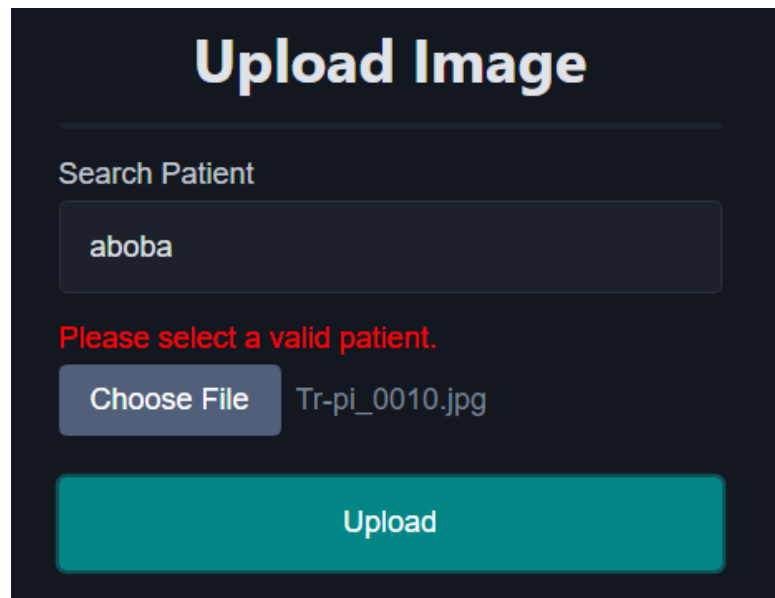
Рисунок 5.9 – Сторінка «Upload»



The screenshot shows a web interface titled "Upload Image". Below the title is a "Search Patient" section with a text input field containing "He". A dropdown menu is open, showing two suggestions: "Henry Henrys" and "Hendy HendyS". To the right of the dropdown, the text "No file chosen" is visible. Below the search section is a large teal button labeled "Upload".

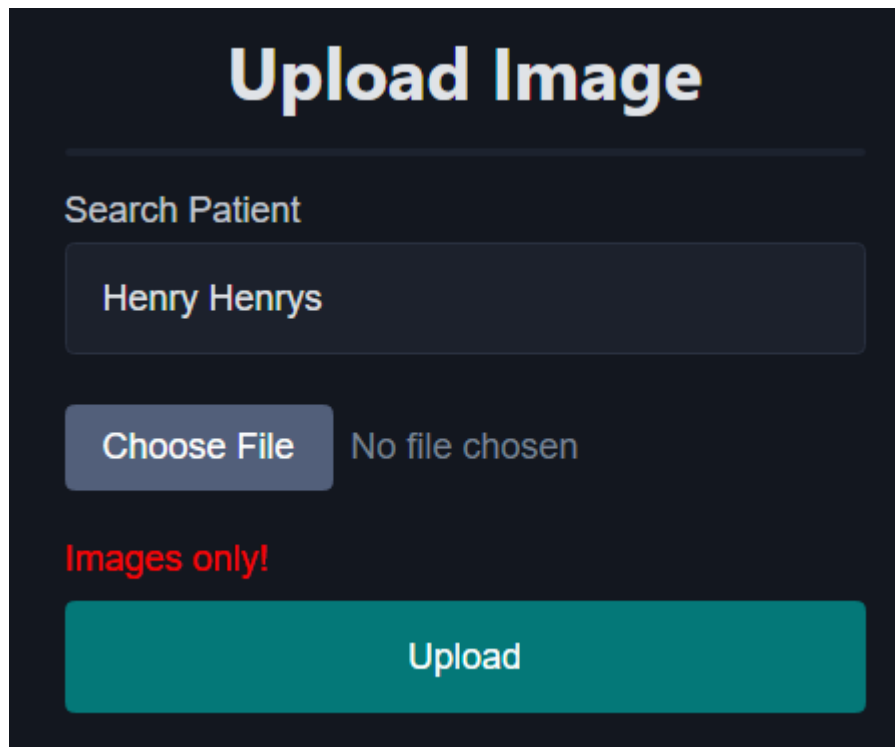
Рисунок 5.10 – Функція автозаповнення в «Upload» формі

У разі введення некоректного або неіснуючого пацієнта система видасть помилку (рис. 5.11). Якщо обрано некоректний формат файлу, система також повідомить про це (рис. 5.12).



The screenshot shows the same "Upload Image" form. The "Search Patient" input field now contains "aboba". Below the input field, a red error message reads "Please select a valid patient." Below the error message is a "Choose File" button, and to its right, the filename "Tr-pi_0010.jpg" is displayed. At the bottom is the same teal "Upload" button.

Рисунок 5.11 – Помилка при введенні некоректного імені й прізвища пацієнта



Upload Image

Search Patient

Henry Henrys

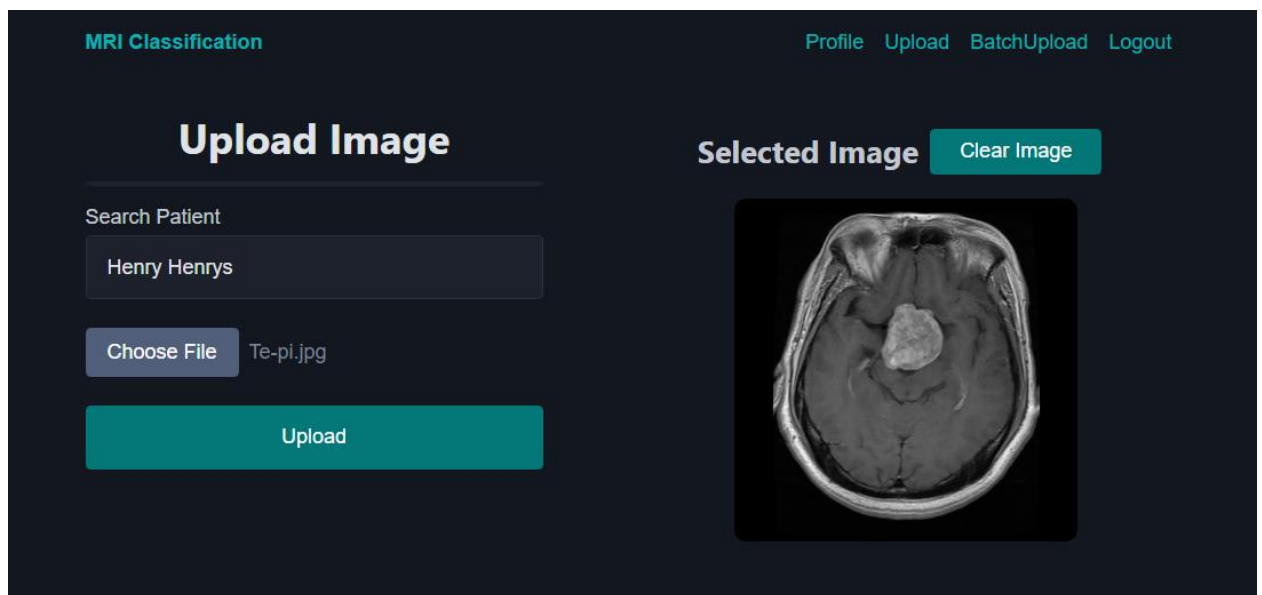
Choose File No file chosen

Images only!

Upload

Рисунок 5.12 – Помилка при виборі некоректного файлу

Після вибору файлу зображення праворуч від форми відображається його прев'ю (рис. 5.13), що дозволяє лікарю переконатися у правильності вибору. Якщо файл обрано помилково, лікар може скористатися кнопкою «Clear image» для скидання вибору.



MRI Classification Profile Upload BatchUpload Logout

Upload Image

Search Patient

Henry Henrys

Choose File Te-pi.jpg

Upload

Selected Image Clear Image

Te-pi.jpg

Рисунок 5.13 – Прев'ю обраного знімку МРТ

Після підтвердження вибору та натискання кнопки «Upload» кнопка становиться не активною та зображення відправляється на сервер для класифікації. Результат класифікації відображається на сторінці у вигляді прев'ю знімка з підписом, що вказує на розпізнаний клас пухлини (рис. 5.14). Під зображенням також є кнопка «Write conclusion» для швидкого переходу до написання висновку, якщо лікар цього бажає.

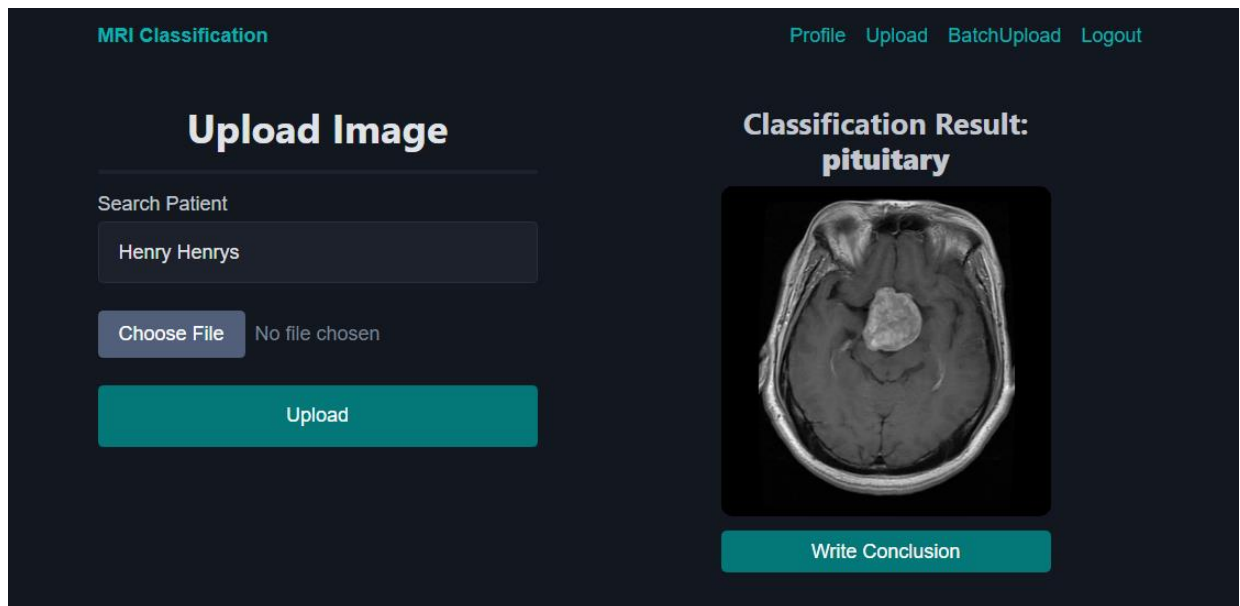


Рисунок 5.14 – Відображення результату класифікації

Лікар може продовжити обирати та завантажувати нові знімки, при цьому попередні результати залишаються видимими. Нові прев'ю з'являтимуться поруч, що дозволяє легко переглядати кілька знімків одночасно (рис. 5.15). Класифікація нових знімків відбувається після повторного натискання кнопки «Upload».

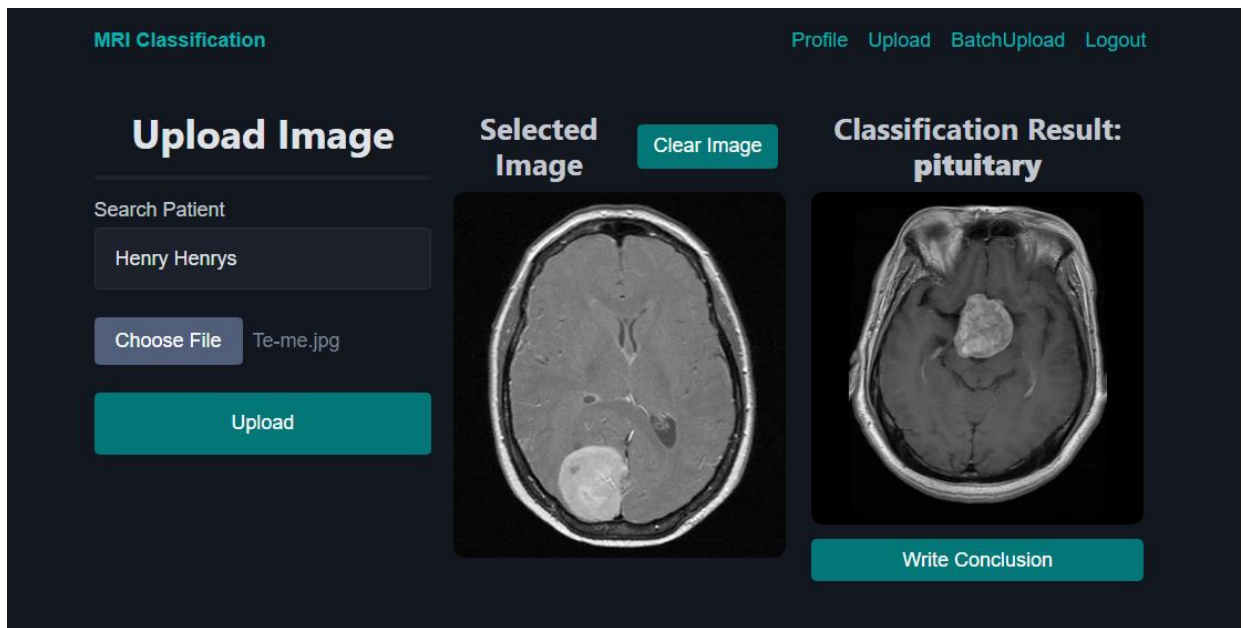


Рисунок 5.15 – Відображення нового й старого знімку

Сторінка «BatchUpload» (рис. 5.16) створена для швидкого завантаження великої кількості знімків МРТ. Ліва частина сторінки містить форму для масового завантаження зображень, яка включає вибір одного або декількох файлів та кнопку «Upload».

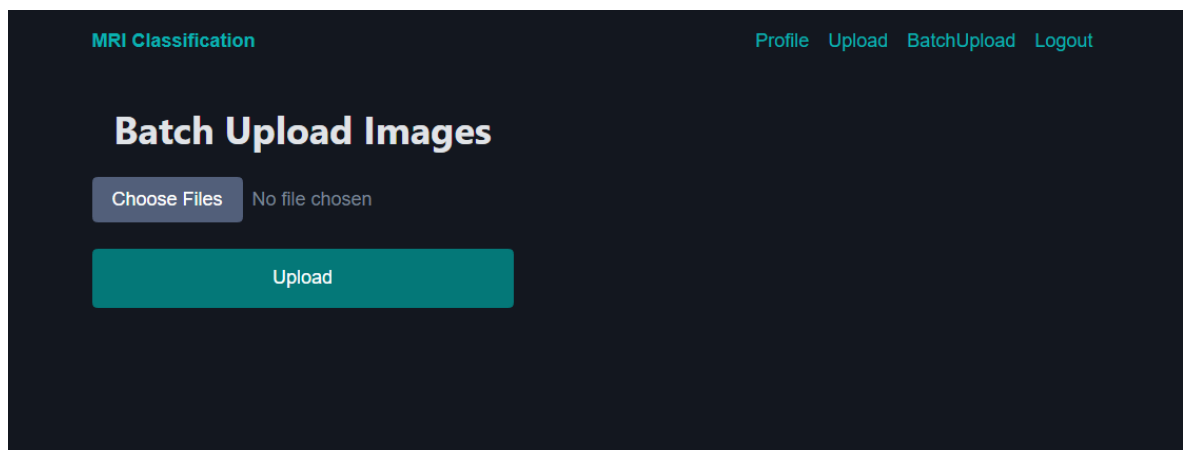


Рисунок 5.16 – Сторінка «BatchUpload»

При виборі файлів їхні прев'ю відображаються під формою (рис. 5.17). Якщо обрано некоректний формат файлу, після натиску на кнопку «Upload» система виведе помилку (рис. 5.18).

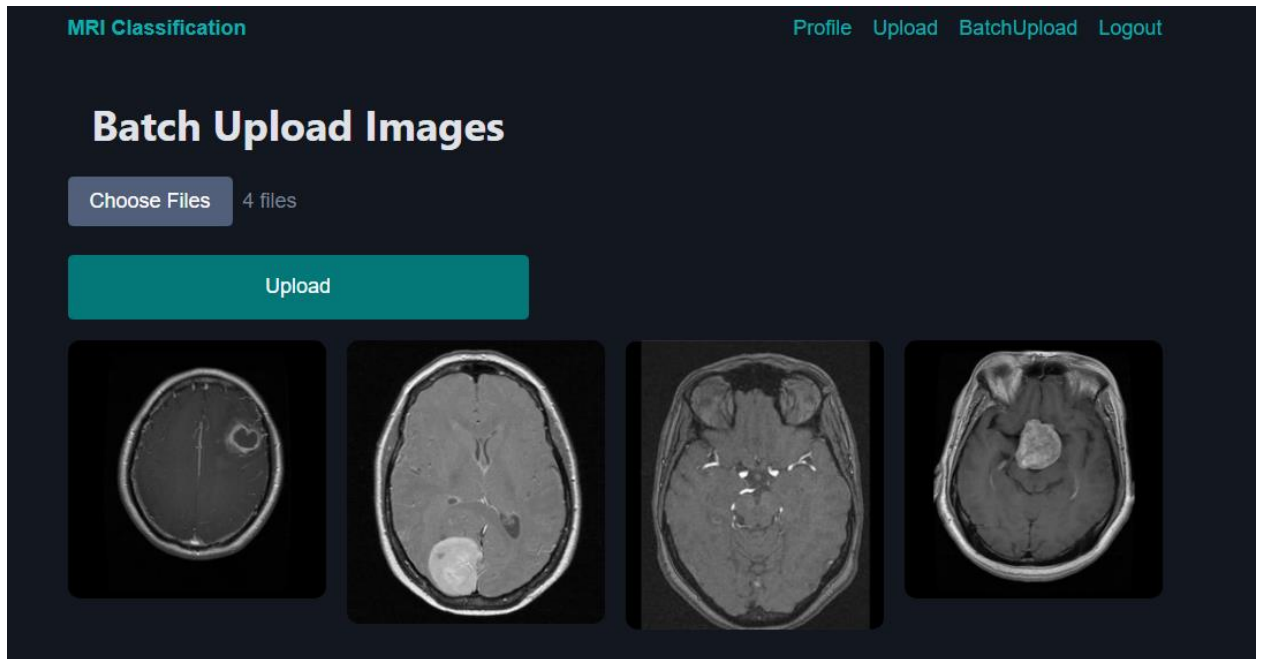


Рисунок 5.17 – Відображення прев'ю багатьох знімків

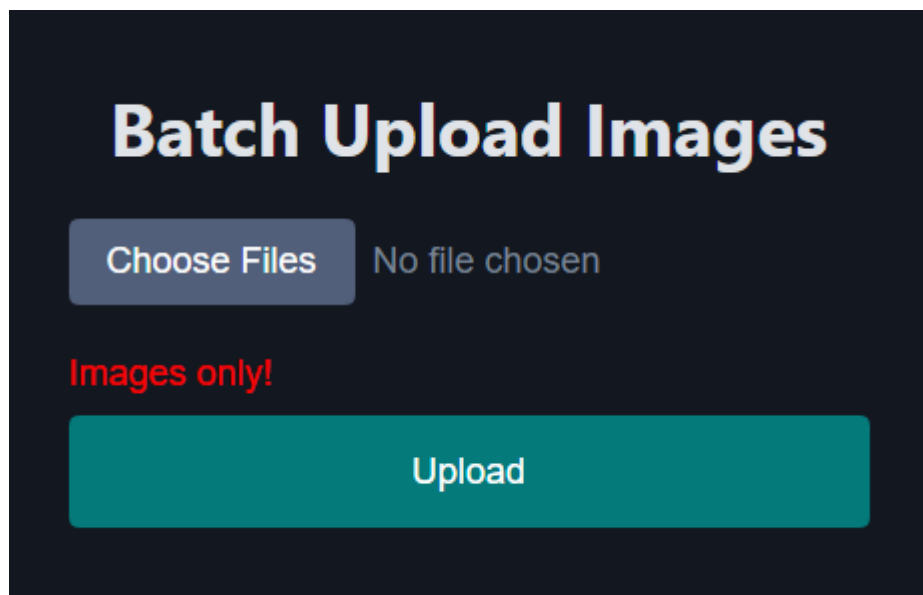


Рисунок 5.18 – Помилка при виборі некоректного файлу

Після натискання кнопки «Upload» всі обрані знімки відправляються на сервер для класифікації. Результати класифікації відображаються на сторінці у вигляді прев'ю знімків з підписами, що вказують на розпізнаний клас пухлини для кожного знімка (рис. 5.19). Важливо зазначити, що відправлені знімки таким чином не зберігаються в базі даних, оскільки не вказується конкретний пацієнт. Ця функція призначена для швидкої класифікації великої кількості знімків за кілька кліків, що значно спрощує роботу лікаря.

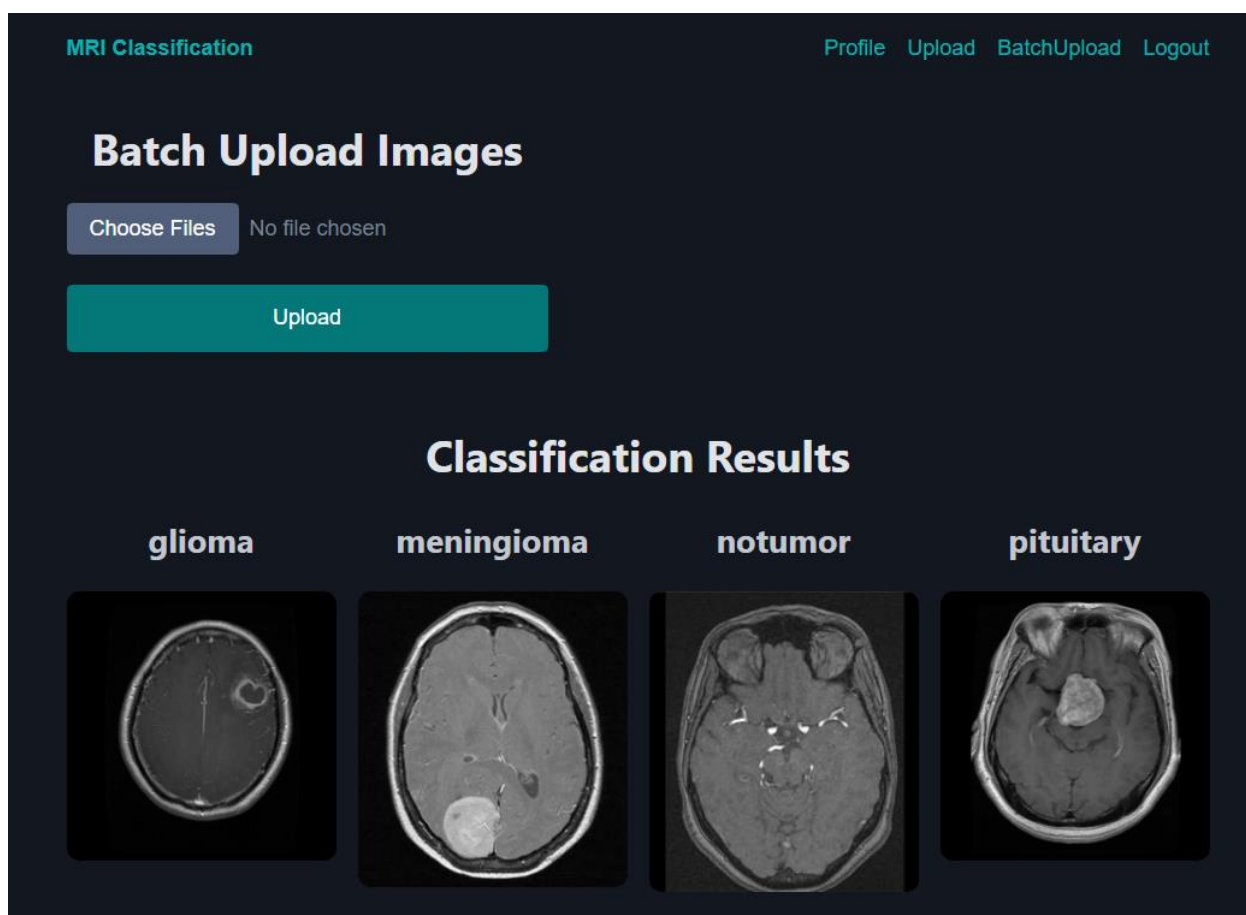


Рисунок 5.19 – Відображення результатів класифікації

Посилання «Logout» дозволяє вийти з поточного облікового запису. Після натиску користувача перенаправляє на головну сторінку вебсайту.

5.3.2 Інструкція з експлуатації програми для пацієнта

Після успішного входу в систему пацієнт має обмежений доступ до функціональності, що включає лише перегляд свого профілю та вихід з облікового запису. За замовчуванням, після входу пацієнт автоматично потрапляє на сторінку свого профілю, де відображаються всі знімки МРТ, що були завантажені до системи (рис. 5.20). На сторінці профілю під кожним знімком вказано дату його завантаження. Якщо лікар ще не залишив свого висновку, під знімком буде відображатися напис «No doctor's conclusion». Це забезпечує, що пацієнт не отримає неповну або потенційно неправильну інформацію щодо свого діагнозу, яка була сформована лише нейронною

мережею. Якщо лікар залишив свій висновок до конкретного знімка, пацієнт побачить посилання «View doctor's conclusion». Цей підхід покликаний захистити пацієнтів від можливої паніки та стресу, які можуть виникнути через неправильне розуміння автоматичного діагнозу без професійної інтерпретації лікаря. Нейронна мережа може визначити пухлину, але лише лікар може точно оцінити її серйозність та надати відповідні рекомендації щодо лікування. Пацієнт, побачивши автоматичний діагноз без коментаря лікаря, може занепокоїтися та почати шукати інформацію в інтернет, що може призвести до хибних висновків і зайвого стресу. Лікар, зі свого боку, може мати більш оптимістичний прогноз та рекомендації щодо лікування.

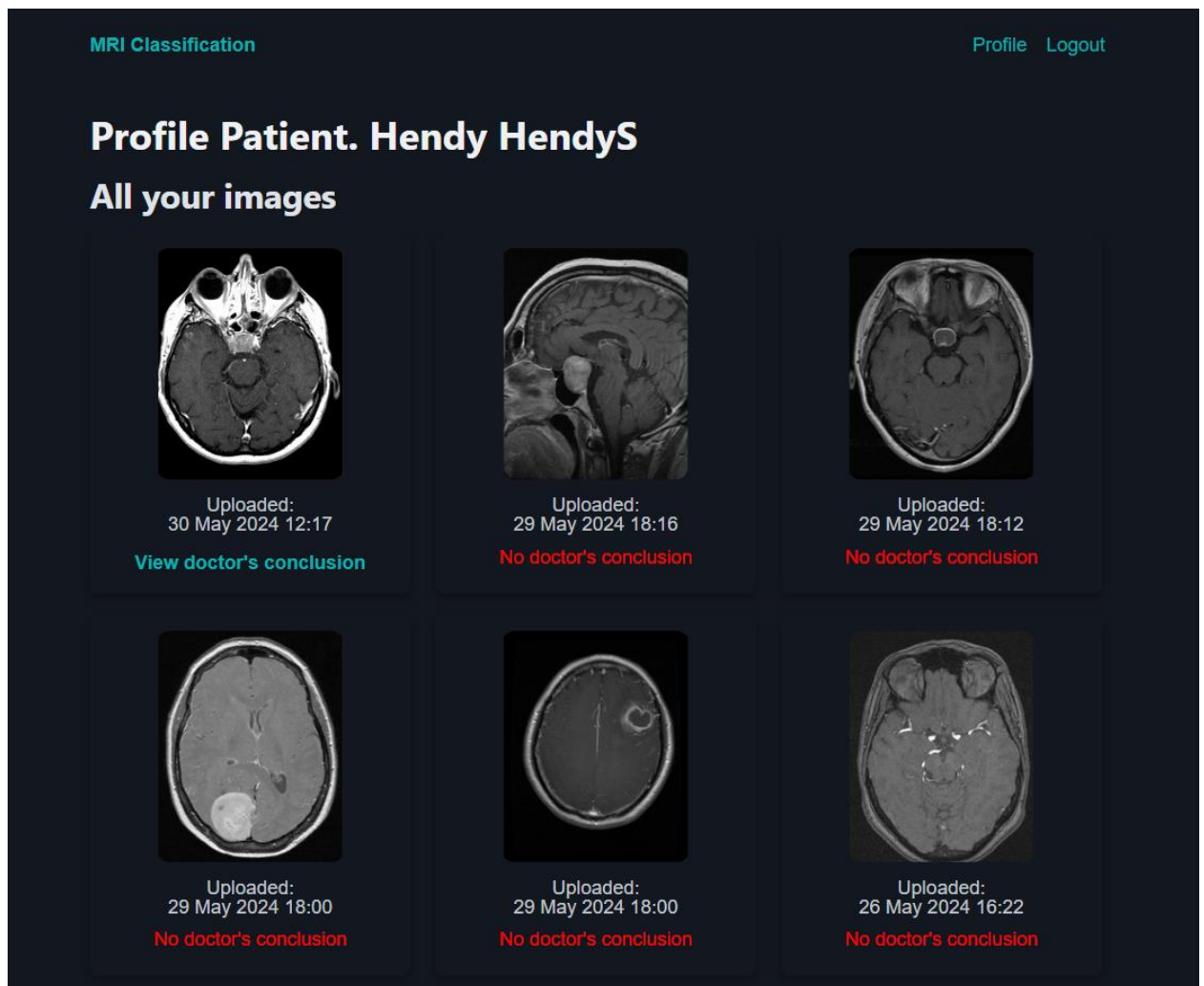


Рисунок 5.20 – Особистий кабінет пацієнта

Коли пацієнт натискає на посилання «View doctor's conclusion», на сторінці з'являється модальне вікно (рис. 5.21), що містить інформацію про знімок. У модальному вікні відображається саме зображення знімка, діагноз, перевірений лікарем, а також висновки лікаря. Ці висновки включають ім'я та прізвище лікаря, дату написання висновку та самий текст висновку. Якщо лікар залишив декілька висновків, всі вони будуть відображатися в модальному вікні, забезпечуючи повну інформацію для пацієнта.

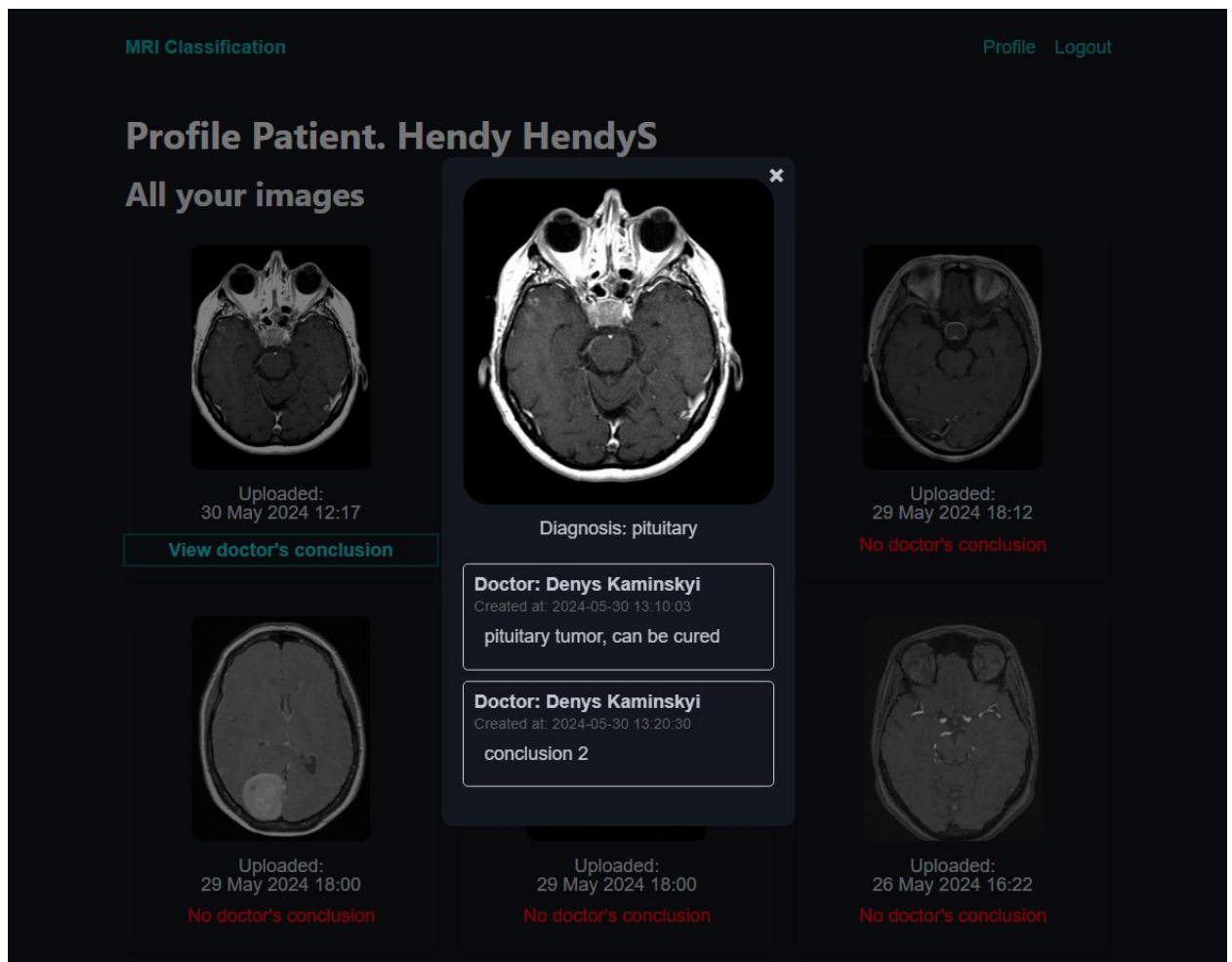


Рисунок 5.21 – Модальне вікно з інформацією про знімок

5.3.3 Інструкція з експлуатації програми для адміністратора

Після успішного входу в систему адміністратор потрапляє на панель управління з низкою основних функціональних пунктів (рис. 5.22). Ці пункти

включають: «Register New Doctor», «Manage users», «Errors logs», «Retrain NN», та «Select NN checkpoint».



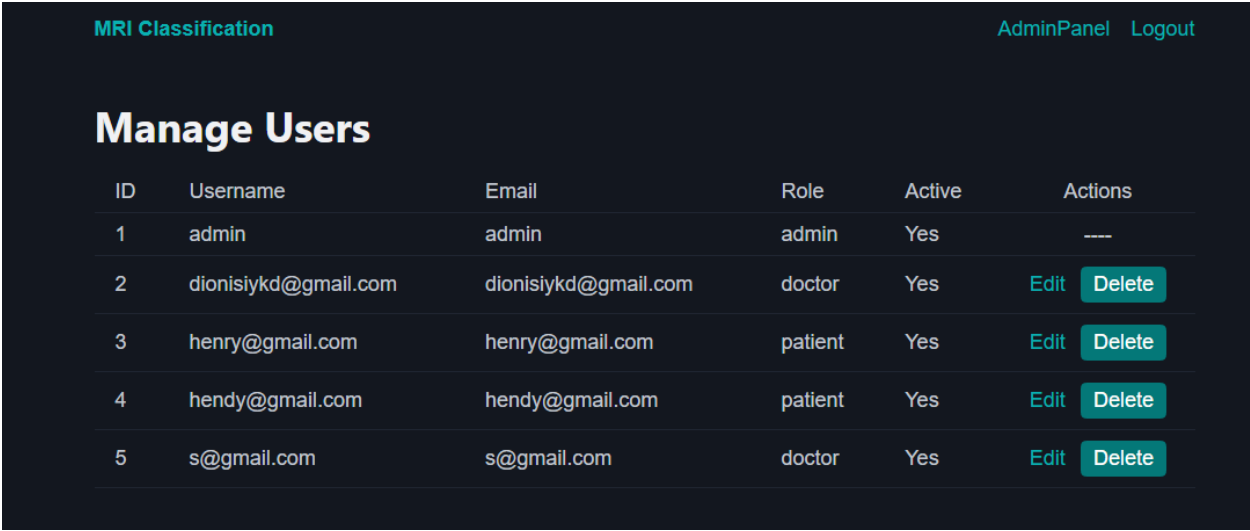
Рисунок 5.22 – Панель адміністратора

Перша вкладка, «Register New Doctor» (рис. 5.23), призначена для реєстрації нових лікарів. Щоб запобігти можливості помилкової реєстрації пацієнтів як лікарів, реєстрацію лікарів може виконувати лише адміністратор. На цій сторінці адміністратор вводить необхідні дані для створення нового облікового запису лікаря: електронну пошту, ім'я користувача, пароль, ім'я, прізвище, спеціальність та дату народження.

The image shows a web application interface with a dark theme. At the top, there is a header bar with 'MRI Classification' on the left and 'AdminPanel Logout' on the right. Below the header, the main content area is titled 'Register Form' in a bold font. The form consists of several input fields: 'Email', 'Username', 'Password', 'Confirm Password', 'First Name', 'Last Name', 'Specialty', and 'Birth Date'. The 'Birth Date' field has a placeholder 'mm/dd/yyyy' and a calendar icon. At the bottom of the form, there is a large green button labeled 'Register'.

Рисунок 5.23 – Сторінка реєстрації нового лікаря

Вкладка «Manage users» (рис. 5.24) дозволяє адміністратору керувати всіма користувачами системи. При переході на цю сторінку адміністратору відображається таблиця, яка містить інформацію про кожного користувача, включаючи ім'я користувача, електронну пошту, роль та статус облікового запису. Адміністратор може видалити користувача, натиснувши кнопку «Delete», або змінити його дані, натиснувши кнопку «Edit».



ID	Username	Email	Role	Active	Actions
1	admin	admin	admin	Yes	---
2	dionisiykd@gmail.com	dionisiykd@gmail.com	doctor	Yes	Edit Delete
3	henry@gmail.com	henry@gmail.com	patient	Yes	Edit Delete
4	hendy@gmail.com	hendy@gmail.com	patient	Yes	Edit Delete
5	s@gmail.com	s@gmail.com	doctor	Yes	Edit Delete

Рисунок 5.24 – Сторінка «Manage users»

При редагуванні відкривається форма зі значеннями, які вже збережені в базі даних (рис. 5.25). Після внесення змін та натискання кнопки «Update» інформація оновлюється, і адміністратор перенаправляється на сторінку «Manage users», де можна побачити оновлені дані.

На рис. 5.26 наведено приклад видалення користувача з email s@gmail.com та змін даних користувача henry@gmail.com на denry@gmail.com разом з відключенням його акаунту.

MRI Classification

AdminPanel Logout

Edit User henry@gmail.com

Username

henry@gmail.com

Email

henry@gmail.com

First name

Henry

Last name

Henrys

Active

☒

Update

Рисунок 5.25 – Сторінка з формою для змін даних користувача

MRI Classification

AdminPanel Logout

Manage Users

ID	Username	Email	Role	Active	Actions	
1	admin	admin	admin	Yes	----	
2	dionisiykd@gmail.com	dionisiykd@gmail.com	doctor	Yes	Edit	Delete
3	denry@gmail.com	denry@gmail.com	patient	No	Edit	Delete
4	hendy@gmail.com	hendy@gmail.com	patient	Yes	Edit	Delete

Рисунок 5.26 – Приклад змінених даних користувачів

Якщо цей пацієнт спробує увійти в систему під цими даними, йому виведеться відповідне повідомлення (рис. 5.27).

Login Form

[User is not active.]

Login

Рисунок 5.27 – Повідомлення про неактивний акаунт

Вкладка «Errors logs» (рис. 5.28) надає адміністратору доступ до журналу помилок системи. На цій сторінці відображаються всі помилки з міткою часу їх виникнення. Це дозволяє адміністратору мати повне уявлення про стан системи та оперативно вирішувати виникаючі проблеми.

MRI ClassificationAdminPanelLogout

Error Log

#	Error Message	Timestamp
650	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 14:00:56.646093
649	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 14:00:51.951524
648	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:54:45.082836
647	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:54:39.176654
646	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:54:37.257714
645	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:54:36.285204
644	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:51:13.460213
643	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:50:59.694977
642	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:49:58.311059
641	404 Not Found: The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.	2024-05-30 13:49:55.701450

Рисунок 5.28 – Журнал помилок системи

Вкладка «Retrain NN» містить форму для перенавчання нейронної мережі (рис. 5.29), якщо, наприклад, був доповнений набір даних великою кількістю нових знімків МРТ. За замовчуванням у формі зазначені базові гіперпараметри, які адміністратор може налаштовувати. Адміністратор також має вказати назву архітектури нейронної мережі, яка реалізована на серверній частині. За замовчуванням використовується модифікована архітектура AlexNet. Після натискання кнопки «Retrain model» система запитує підтвердження дії (рис. 5.30), оскільки навчання моделі є ресурсоємним процесом, і нове навчання не може бути розпочато до завершення поточного.

The screenshot shows a web interface for 'MRI Classification'. At the top, there are links for 'AdminPanel' and 'Logout'. The main heading is 'Retrain Model'. Below it, a message says 'Click the button below to initiate model retraining.' The form contains several input fields: 'Network name' with a dropdown menu showing 'OwnV2', 'Epochs' with the value '20', 'Batch Size' with '16', 'Learning Rate' with '0.001', 'Beta 1' with '0.9', and 'Beta 2' with '0.999'. At the bottom of the form is a large green button labeled 'Retrain Model'.

Рисунок 5.29 – Форма перенавчання нейронної мережі

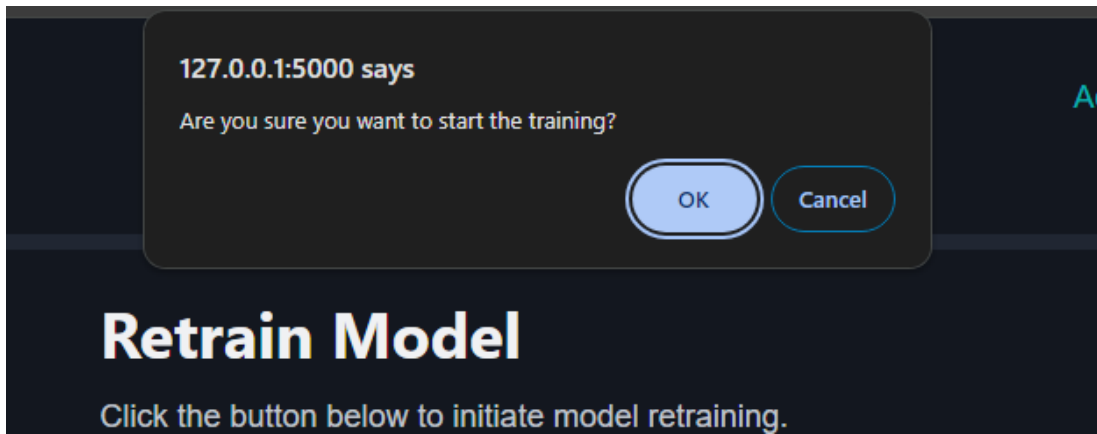


Рисунок 5.30 – Підтвердження запиту на перенавчання моделі

Після підтвердження запиту навчання починається, і адміністратор отримує повідомлення про успішний старт процесу (рис. 5.31). Кнопка стає неактивною до завершення навчання. Прогрес навчання відображається за допомогою індикатора прогресу (`progress_bar`), який оновлюється в реальному часі. Після завершення навчання система відправляє запит на отримання результатів, які відображаються у панелі праворуч (рис. 5.32).

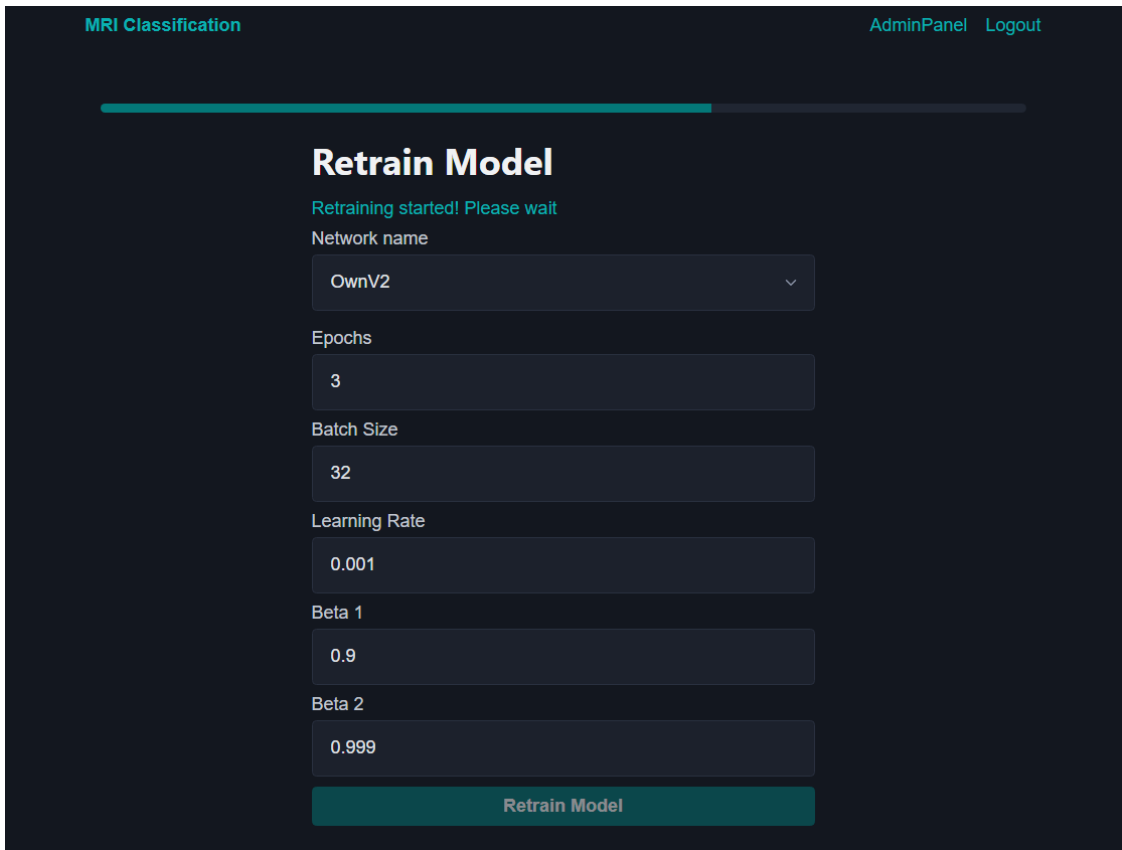


Рисунок 5.31 – Стан сторінки після початку перенавчання

Рисунок 5.32 – Стан сторінки після отримання результатів перенавчання

Вкладка «Select NN checkpoint» (рис. 5.33) дозволяє адміністратору вибрати поточну нейронну мережу та її ваги. На цій сторінці є два випадających списки: «select checkpoint», де відображаються всі збережені чекпоінти (рис.5.34), та «select neural network», де можна вибрати архітектуру нейронної мережі (рис. 5.35), до якої належить обраний чекпоінт. В модулі з нейронною мережею використовується callback для збереження найкращих ваг (чекпоінтів) нейронної мережі після кожної епохи навчання. Ці чекпоінти зберігаються в окремій папці для подальшого використання.

Після натискання кнопки «load checkpoint» система глобально змінює архітектуру нейронної мережі та її ваги на обрані.

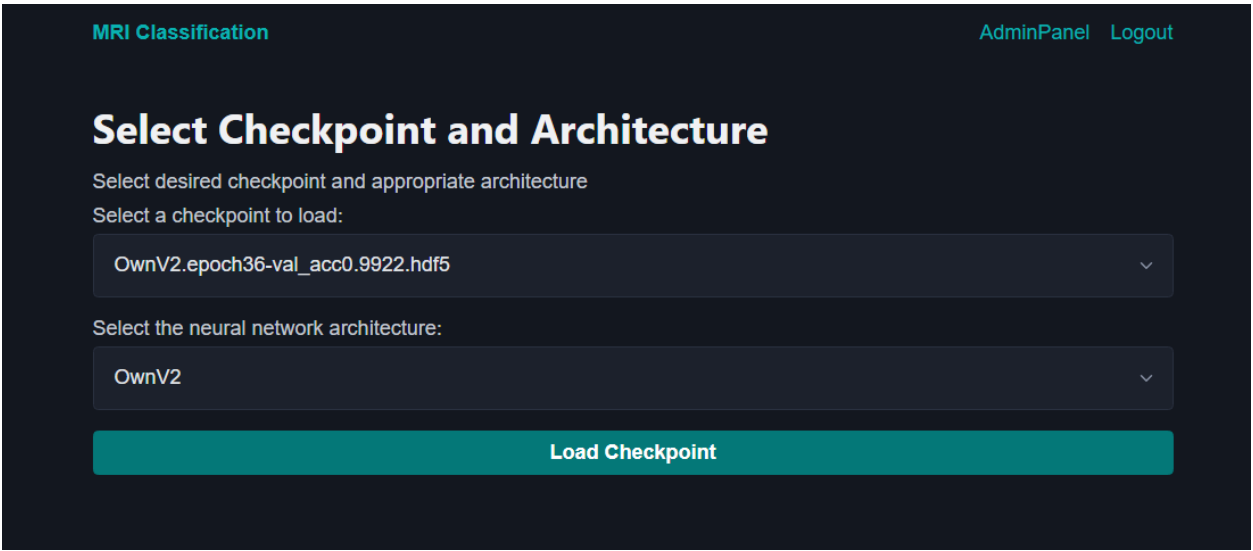


Рисунок 5.33 – Сторінка «Select NN checkpoint»

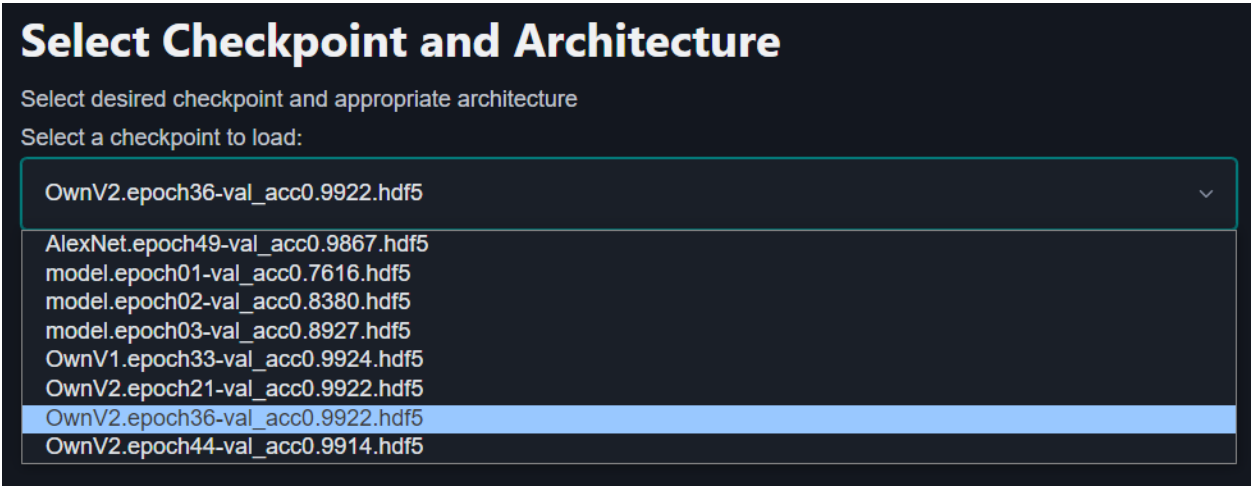


Рисунок 5.34 – Випадаючий список збережених чекпоінтів

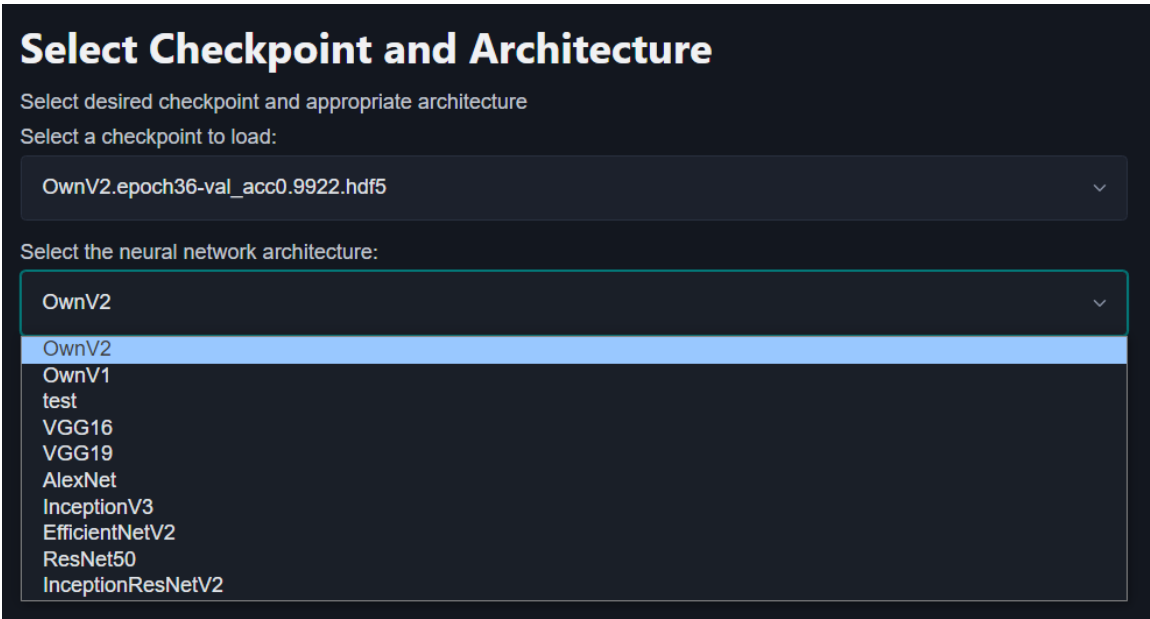


Рисунок 5.35 – Випадаючий список реалізованих нейронних мереж

5.4 Методика та результати тестування

Для навчання та оцінки точності нейронних мереж було використано наступні значення гіперпараметрів:

- кількість епох – 50;
- розмір партії (батчу) – 32;
- оптимізатор – Adam;
- швидкість навчання – 0,001.

Для більшості архітектур для порівняння результатів було використано Keras.applications модуль з реалізованими архітектурами нейронних мереж. Після базового шару обраної архітектури було вставлено GlobalAveragePooling2D-шар, який перетворює високорозмірний вихід базової моделі з формою (висота, ширина, канали) у більш керований одношаровий тензор з формою (канали). Це зменшення є важливим для зменшення вимірності, забезпечуючи сумісність виходу з наступним Dense-шаром, який вимагає двовимірного вводу. За допомогою усереднення кожної карти ознак, шар GlobalAveragePooling2D також зменшує обчислювальне навантаження та кількість параметрів й підвищує стійкість моделі, зберігаючи найбільш значущі ознаки.

На рис. 5.36 наведено тестування архітектури ResNet50 на заданій вибірці даних.

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 5, 5, 2048)	23587712
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 4)	8196
Total params: 23,595,908 Trainable params: 8,196 Non-trainable params: 23,587,712		
Epoch 1/60		
178/178 [=====] - ETA: 0s - loss: 1.2410 - acc: 0.4692		
Epoch 1: val_acc improved from -inf to 0.52578, saving model to ./checkpoints\model.epoch01-val_acc0.5258.hdf5		
178/178 [=====] - 35s 178ms/step - loss: 1.2410 - acc: 0.4692 - val_loss: 1.2029 - val_acc: 0.5258		
Epoch 2/60		
178/178 [=====] - ETA: 0s - loss: 1.1051 - acc: 0.5801		
Epoch 2: val_acc improved from 0.52578 to 0.53594, saving model to ./checkpoints\model.epoch02-val_acc0.5359.hdf5		
...		
Epoch 59: val_acc did not improve from 0.70312		
178/178 [=====] - 32s 181ms/step - loss: 0.6511 - acc: 0.7468 - val_loss: 0.8035 - val_acc: 0.6953		
Epoch 59: early stopping		
40/40 [=====] - 4s 94ms/step - loss: 0.8035 - acc: 0.6953		
Output is truncated. View as a scrollable element or open in a text editor . Adjust cell output settings...		

Рисунок 5.36 – Тестування архітектури ResNet50

На рис. 5.37 наведено тестування архітектури EfficientNetV2 на заданій вибірці даних.

Layer (type)	Output Shape	Param #
efficientnetv2-b0 (Functional)	(None, 5, 5, 1280)	5919312
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense (Dense)	(None, 4)	5124
Total params: 5,924,436 Trainable params: 5,124 Non-trainable params: 5,919,312		
Epoch 1/60		
178/178 [=====] - ETA: 0s - loss: 1.3934 - acc: 0.2519		
Epoch 1: val_acc improved from -inf to 0.31641, saving model to ./checkpoints\model.epoch01-val_acc0.3164.hdf5		
178/178 [=====] - 37s 178ms/step - loss: 1.3934 - acc: 0.2519 - val_loss: 1.3793 - val_acc: 0.3164		
Epoch 2/60		
178/178 [=====] - ETA: 0s - loss: 1.3848 - acc: 0.2754		
...		
Epoch 9: val_acc did not improve from 0.31641		
178/178 [=====] - 29s 165ms/step - loss: 1.3844 - acc: 0.2771 - val_loss: 1.3769 - val_acc: 0.3164		
Epoch 9: early stopping		
40/40 [=====] - 2s 53ms/step - loss: 1.3769 - acc: 0.3164		
Output is truncated. View as a scrollable element or open in a text editor . Adjust cell output settings...		

Рисунок 5.37 – Тестування архітектури EfficientNetV2

На рис. 5.38 наведено тестування архітектури InceptionV3 на заданій вибірці даних.

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 3, 3, 2048)	21802784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 4)	8196

Total params: 21,810,980
 Trainable params: 8,196
 Non-trainable params: 21,802,784

Epoch 1/60
 178/178 [=====] - ETA: 0s - loss: 0.6258 - acc: 0.7593
 Epoch 1: val_acc improved from -inf to 0.76641, saving model to [./checkpoints\model.epoch01-val_acc0.7664.hdf5](#)
 178/178 [=====] - 35s 171ms/step - loss: 0.6258 - acc: 0.7593 - val_loss: 0.6257 - val_acc: 0.7664
 Epoch 2/60
 178/178 [=====] - ETA: 0s - loss: 0.3993 - acc: 0.8496
 Epoch 2: val_acc improved from 0.76641 to 0.81719, saving model to [./checkpoints\model.epoch02-val_acc0.8172.hdf5](#)
 ...
 Epoch 28: val_acc did not improve from 0.90781
 178/178 [=====] - 29s 161ms/step - loss: 0.1545 - acc: 0.9426 - val_loss: 0.2674 - val_acc: 0.8984
 Epoch 28: early stopping
 40/40 [=====] - 2s 48ms/step - loss: 0.2674 - acc: 0.8984
 Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

Рисунок 5.38 – Тестування архітектури InceptionV3

На рис. 5.39 наведено тестування архітектури InceptionResNetV2 на заданій вибірці даних.

Layer (type)	Output Shape	Param #
inception_resnet_v2 (Functional)	(None, 3, 3, 1536)	54336736
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1536)	0
dense (Dense)	(None, 4)	6148

Total params: 54,342,884
 Trainable params: 6,148
 Non-trainable params: 54,336,736

Epoch 1/60
 178/178 [=====] - ETA: 0s - loss: 0.6401 - acc: 0.7493
 Epoch 1: val_acc improved from -inf to 0.77266, saving model to [./checkpoints\model.epoch01-val_acc0.7727.hdf5](#)
 178/178 [=====] - 48s 211ms/step - loss: 0.6401 - acc: 0.7493 - val_loss: 0.6381 - val_acc: 0.7727
 Epoch 2/60
 178/178 [=====] - ETA: 0s - loss: 0.4377 - acc: 0.8407
 ...
 Epoch 32: val_acc did not improve from 0.89688
 178/178 [=====] - 34s 192ms/step - loss: 0.1670 - acc: 0.9401 - val_loss: 0.2773 - val_acc: 0.8961
 Epoch 32: early stopping
 40/40 [=====] - 5s 114ms/step - loss: 0.2773 - acc: 0.8961
 Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

Рисунок 5.39 – Тестування архітектури InceptionResNetV2

На рис. 5.40 наведено тестування архітектури VGG16 на заданій вибірці даних.

```

Layer (type)                 Output Shape                 Param #
-----
vgg16 (Functional)           (None, 4, 4, 512)           14714688

global_average_pooling2d (G (None, 512)                 0
lobalAveragePooling2D)

dense (Dense)                (None, 4)                   2052

=====
Total params: 14,716,740
Trainable params: 2,052
Non-trainable params: 14,714,688

Epoch 1/60
178/178 [=====] - ETA: 0s - loss: 1.0695 - acc: 0.6206
Epoch 1: val_acc improved from -inf to 0.68125, saving model to ./checkpoints\model.epoch01-val_acc0.6812.hdf5
178/178 [=====] - 41s 197ms/step - loss: 1.0695 - acc: 0.6206 - val_loss: 0.9112 - val_acc: 0.6812
Epoch 2/60
...
Epoch 45: val_acc did not improve from 0.85469
178/178 [=====] - 32s 178ms/step - loss: 0.2967 - acc: 0.8977 - val_loss: 0.4010 - val_acc: 0.8492
Epoch 45: early stopping
40/40 [=====] - 3s 85ms/step - loss: 0.4010 - acc: 0.8492
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Рисунок 5.40 – Тестування архітектури VGG16

На рис. 5.41 наведено тестування архітектури VGG19 на заданій вибірці даних.

```

Layer (type)                 Output Shape                 Param #
-----
vgg19 (Functional)           (None, 4, 4, 512)           20024384

global_average_pooling2d (G (None, 512)                 0
lobalAveragePooling2D)

dense (Dense)                (None, 4)                   2052

=====
Total params: 20,026,436
Trainable params: 2,052
Non-trainable params: 20,024,384

Epoch 1/60
178/178 [=====] - ETA: 0s - loss: 1.0841 - acc: 0.6192
Epoch 1: val_acc improved from -inf to 0.65625, saving model to ./checkpoints\model.epoch01-val_acc0.6562.hdf5
178/178 [=====] - 41s 201ms/step - loss: 1.0841 - acc: 0.6192 - val_loss: 0.9310 - val_acc: 0.6562
Epoch 2/60
...
Epoch 29: val_acc did not improve from 0.80625
178/178 [=====] - 36s 204ms/step - loss: 0.3897 - acc: 0.8546 - val_loss: 0.5384 - val_acc: 0.8062
Epoch 29: early stopping
40/40 [=====] - 5s 113ms/step - loss: 0.5384 - acc: 0.8062
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```

Рисунок 5.41 – Тестування архітектури VGG19

Структуру архітектури AlexNet було побудовано самостійно за допомогою модулю `keras.layers` (рис. 5.42).

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 35, 35, 96)	34944
max_pooling2d (MaxPooling2D)	(None, 17, 17, 96)	0
batch_normalization (Batch Normalization)	(None, 17, 17, 96)	384
conv2d_1 (Conv2D)	(None, 17, 17, 256)	614656
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 256)	0
batch_normalization_1 (Batch Normalization)	(None, 8, 8, 256)	1024
conv2d_2 (Conv2D)	(None, 8, 8, 384)	885120
conv2d_3 (Conv2D)	(None, 8, 8, 384)	1327488
conv2d_4 (Conv2D)	(None, 8, 8, 256)	884992
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 256)	0
batch_normalization_2 (Batch Normalization)	(None, 3, 3, 256)	1024
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 4096)	9441280
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4)	16388
=====		
Total params: 29,988,612		
Trainable params: 29,987,396		
Non-trainable params: 1,216		

Рисунок 5.42 – Побудована архітектура AlexNet

На рис. 5.43 наведено тестування архітектури AlexNet на заданій вибірці даних.

```

Epoch 50/60
178/178 [=====] - ETA: 0s - loss: 0.0076 - acc: 0.9977
Epoch 50: val_acc did not improve from 0.98906
178/178 [=====] - 29s 163ms/step - loss: 0.0076 - acc: 0.9977 - val_l
Epoch 51/60
178/178 [=====] - ETA: 0s - loss: 0.0077 - acc: 0.9967
Epoch 51: val_acc did not improve from 0.98906
178/178 [=====] - 29s 161ms/step - loss: 0.0077 - acc: 0.9967 - val_l
Epoch 52/60
178/178 [=====] - ETA: 0s - loss: 0.0091 - acc: 0.9967
Epoch 52: val_acc did not improve from 0.98906
178/178 [=====] - 29s 164ms/step - loss: 0.0091 - acc: 0.9967 - val_l
Epoch 52: early stopping
40/40 [=====] - 2s 40ms/step - loss: 0.0752 - acc: 0.9852

```

Рисунок 5.43 – Тестування архітектури AlexNet

Виходячи з результатів, було виявлено, що архітектура AlexNet є найкращою серед наведених (рис. 5.44). З метою подальшого покращення результатів були проведені модифікації в її архітектурі (рис. 5.45). Оригінальна AlexNet має 29,988,612 параметрів, тоді як модифікована модель має 888,024 параметри, що приблизно в 34 рази менше, значно зменшуючи обчислювальні ресурси, необхідні для її тренування. Результати тестування модифікованої архітектури наведено на рис. 5.44.

```

Epoch 50: val_acc did not improve from 0.99221
178/178 [=====] - 27s 150ms/step - loss: 0.0061 - acc: 0.9977 - val_loss: 0.0367 - val_
40/40 [=====] - 2s 39ms/step - loss: 0.0367 - acc: 0.9922
Test loss: 0.0367
Test accuracy: 0.9922
Training time: 1558.3431
PS E:\06_Coding\Python\ML\BrainTumor> 

```

Рисунок 5.44 – Тестування модифікованої архітектури

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 64)	27712
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
batch_normalization (Batch Normalization)	(None, 13, 13, 64)	256
conv2d_1 (Conv2D)	(None, 7, 7, 128)	295040
max_pooling2d_1 (MaxPooling2D)	(None, 3, 3, 128)	0
batch_normalization_1 (Batch Normalization)	(None, 3, 3, 128)	512
conv2d_2 (Conv2D)	(None, 3, 3, 256)	295168
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 256)	0
batch_normalization_2 (Batch Normalization)	(None, 1, 1, 256)	1024
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 1028)	264196
dropout (Dropout)	(None, 1028)	0
dense_1 (Dense)	(None, 4)	4116
Total params: 888,024		
Trainable params: 887,128		
Non-trainable params: 896		

Рисунок 5.45 – Структура модифікованої архітектури

За результатами модифікованої архітектури було побудовано графіки точності (рис. 5.46) та функції витрат (рис. 5.47) на протязі всього періоду навчання для тестової та валідаційної вибірок.

Training and Evaluation Accuracy every Epoch for "OwnV2"

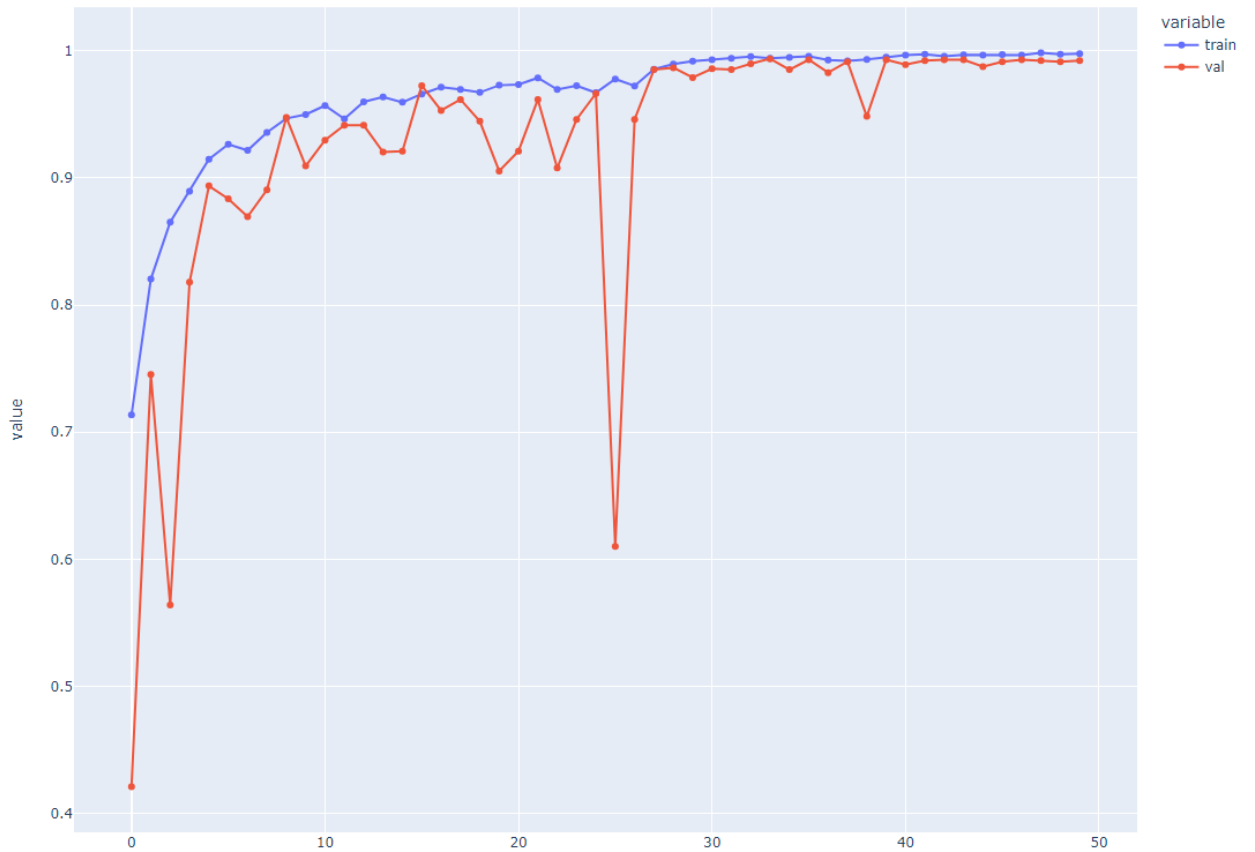


Рисунок 5.46 – Графік точності для тестової та валідаційної вибірки

Training and Evaluation Loss every Epoch

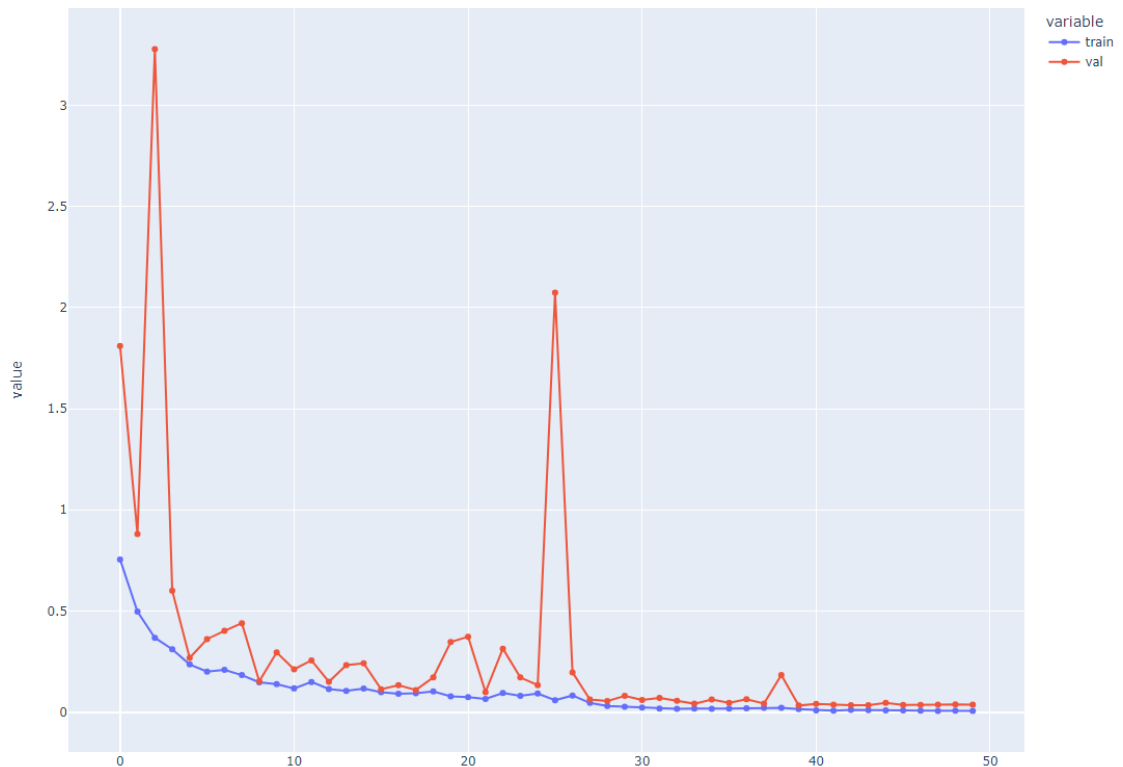


Рисунок 5.47 – Графік функції помилки для тестової та валідаційної вибірки

Результати тестування показали, що точність модифікованої моделі (99,22%), створеної на основі запропонованого методу, перевищує результати інших моделей на цьому наборі даних. Порівняння отриманих результатів представлено в табл. 5.1.

Таблиця 5.1 – Порівняння точності розпізнавання різних архітектур

Модель	Точність розпізнавання
EfficientNetV2	31,64%
ResNet50	70,31%
VGG19	80,62%
VGG16	85,46%
InceptionResNetV2	89,68%
InceptionV3	90,78%
AlexNet	98,90%
Модифікована модель AlexNet	99,22%

Деякі з даних результатів були опубліковані та представлені під час виступів на конференціях, зокрема на V міжнародній науково-практичній конференції «Інформаційні моделюючі технології, системи та комплекси (ІМТСК-2024)» (18-19 квітня 2024 р., Черкаси, Україна) [22] та на університетській науково-технічній конференції «Тиждень науки-2024» (Запоріжжя, 15-19 квітня 2024 р.) [23].

5.5 Тестування застосунку та його результати

Для перевірки якості було проведено ряд тестувань за встановленими функціональними вимогами. Для кожного тестування були створені чеклисти (рис. 5.48), де було зазначено список перевірок та результати тестування. В результаті програмний застосунок пройшов всі перевірки.

1	Вимоги лікаря	
a	Авторизація	Passed
b	Перегляд МРТ знімків пацієнтів	Passed
c	Пошук МРТ знімків по пацієнту	Passed
d	Діагностика знімку	Passed
e	Пакетна діагностика знімків	Passed
f	Завантаження МРТ знімку з комп'ютера	Passed
g	Пакетне завантаження МРТ знімків	Passed
h	Розпізнавання знімків	Passed
i	Редагування діагнозу МРТ знімка	Passed
j	Збереження МРТ знімку в БД	Passed
k	Залишення висновків до МРТ знімків	Passed
l	Вихід з особистого кабінету	Passed
2	Вимоги пацієнта	
a	Реєстрація	Passed
b	Авторизація	Passed
c	Перегляд власних МРТ знімків	Passed
d	Перегляд висновків лікаря до знімку	Passed
e	Вихід з особистого кабінету	Passed
3	Вимоги адміністратора	
a	Авторизація	Passed
b	Перегляд всіх користувачів	Passed
c	Редагування даних користувача	Passed
d	Видалення користувача	Passed
e	Реєстрація нових лікарів	Passed
f	Перегляд журналу помилок	Passed
g	Ініціювати перенавчання нейронної мережі	Passed
h	Зміна архітектури й ваг моделі для класифікації	Passed
i	Вихід з панелі адміністратора	Passed

Рисунок 5.48 – Чеклист тестування функціональних вимог

5.6 Висновки до п'ятого розділу

У даному розділі було наведено опис застосування розробленого програмного забезпечення для лікарів, пацієнтів та адміністратора. Було розглянуто основні та спеціальні сторінки для кожної ролі. Також було розглянуто процес тестування різних архітектур нейронних мереж, наведено

запропоновану модифіковану модель та наведено порівняння результатів. Тестування виявилось повністю успішним.

ВИСНОВКИ

У процесі виконання дипломної кваліфікаційної роботи було розроблено програмне забезпечення для класифікації знімків МРТ на наявність пухлини головного мозку, призначене для допомоги лікарям у діагностуванні.

Під час роботи було виконано такі завдання:

- проведено аналіз предметної області та розроблено технічного завдання;
- проведено експериментальне дослідження різних архітектур CNN;
- проведено експериментальне дослідження архітектури з найбільшою точністю для подальшого покращення результату на основі запропонованого рішення;
- спроектовано та розроблено вебінтерфейс;
- проведення тестування програмного забезпечення.

Під час аналізу предметної області були визначені основні особливості обробки знімків МРТ та проведено порівняння з існуючими аналогами. Було вирішено використовувати метод згорткових нейронних мереж, що забезпечує високу точність класифікації медичних зображень. Для реалізації нейронної мережі було обрано мову програмування Python та бібліотеки TensorFlow і Keras.

На етапі проєктування програмного забезпечення були визначені функціональні вимоги для різних груп користувачів (лікарі, пацієнти, адміністратор), спроектовано структуру бази даних та визначено архітектуру програмного забезпечення MVC.

Розроблене програмне забезпечення включає алгоритм завантаження та класифікації знімків МРТ, зручний для користувачів інтерфейс, а також функціональні можливості для управління обліковими записами користувачів та їх ролями.

Тестування програмного забезпечення показало високу ефективність роботи системи, підтвердило її коректне функціонування та відповідність встановленим вимогам.

Наукова новизна роботи полягає у запропонованому методі діагностування пухлин головного мозку, який характеризується використанням модифікованої архітектури AlexNet. Ця модифікація дозволяє значно зменшити кількість налаштовуваних параметрів, що спрощує вимоги до навчання моделі, при цьому забезпечуючи покращення точності діагностування. Програма не замінює лікаря, але значно полегшує та прискорює процес діагностування, що особливо важливо для своєчасного виявлення та лікування захворювань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Brain Tumor [Electronic resource]. – Access mode : <https://www.cancer.net/cancer-types/brain-tumor/statistics>.
2. Brain tumours [Electronic resource]. – Access mode : <https://www.nhs.uk/conditions/brain-tumours/>.
3. Brain tumor overview [Electronic resource]. – Access mode : https://www.health.harvard.edu/a_to_z/brain-tumor-overview-a-to-z.
4. Brain tumor [Electronic resource]. – Access mode : <https://www.mayoclinic.org/diseases-conditions/brain-tumor/symptoms-causes/syc-20350084>.
5. Brain tumor – primary [Electronic resource]. – Access mode : <https://medlineplus.gov/ency/article/007222.html>.
6. Glia [Electronic resource]. – Access mode : <https://en.wikipedia.org/wiki/Glia>.
7. Brain Tumors [Electronic resource]. – Access mode : <https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Brain-Tumors>.
8. Субботін, С. О. Нейронні мережі [Текст] : теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с..
9. Image classification using Support Vector Machine (SVM) in Python [Electronic resource]. – Access mode : <https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/>.
10. The Power of ViT, Vision Transformers for Image Recognition [Electronic resource]. – Access mode : <https://visionplatform.ai/vision-transformers-vit/>.
11. AlexNet [Electronic resource] : The First CNN to win Image Net. – Access mode : <https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/>.

12. ImageNet Classification with Deep Convolutional Neural Networks [Electronic resource]. – Access mode : https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
13. VGG16 – Convolutional Network for Classification and Detection [Electronic resource]. – Access mode : <https://neurohive.io/en/popular-networks/vgg16/>.
14. VGG-19-Architecture [Electronic resource]. – Access mode : https://www.researchgate.net/figure/VGG-19-Architecture-39-VGG-19-has-16-convolution-layers-grouped-into-5-blocks-After_fig5_359771670.
15. Advanced Guide to Inception v3 [Electronic resource]. – Access mode : <https://cloud.google.com/tpu/docs/inception-v3-advanced>.
16. ResNet-50 [Electronic resource] : The Basics and a Quick Tutorial. – Access mode : <https://datagen.tech/guides/computer-vision/resnet-50/#>.
17. ResNet-50 [Electronic resource]. – Access mode : <https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758>.
18. Brain Tumor MRI Dataset [Electronic resource]. – Access mode : <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset>.
19. How Is Python Used in Machine Learning? [Electronic resource]. – Access mode : <https://builtin.com/machine-learning/python-machine-learning>.
20. Visual Studio Code [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Visual_Studio_Code.
21. How Can Data Scientists Leverage The Power of GPU Jupyter Notebooks To Accelerate Deep Learning Tasks? [Electronic resource]. – Access mode : <https://www.e2enetworks.com/blog/how-can-data-scientists-leverage-the-power-of-gpu-jupyter-notebooks-to-accelerate-deep-learning-tasks#>.
22. Камінський, Д.О. Програмне забезпечення для розпізнавання пухлини головного мозку за знімком магнітно-резонансної томографії [Текст] / Д. О. Камінський, В. М. Льовкін // Інформаційні моделюючі технології, системи та комплекси (IMTСК-2024) : V міжнародна науково-

практична конференція. 18-19 квітня 2024 р., Черкаси, Україна. – Черкаси : Черкаський національний університет імені Богдана Хмельницького, 2024. – С. 168-169.

23. Камінський, Д. О. Програмне забезпечення підтримки роботи лікаря для розпізнавання пухлини головного мозку [Електрон. ресурс] / Д. О. Камінський, В. М. Льовкін // Тиждень науки-2024. Факультет комп'ютерних наук і технологій. Тези доповідей науково-технічної конференції, Запоріжжя, 15-19 квітня 2024 р. / Редкол. : Вадим ШАЛОМЄЄВ (відпов. ред.) Електрон. дані. – Запоріжжя : НУ «Запорізька політехніка», 2024. – Препринт.

ДОДАТОК А
Технічне завдання

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Дослідження та програмна реалізація нейромережних методів розпізнавання пухлини головного мозку за знімком МРТ» згідно з наказом №168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Програмне забезпечення призначене для використання у медичній сфері, зокрема для діагностування пухлин головного мозку. Система дозволяє завантажувати знімки МРТ, отримувати попередні діагнози на основі нейронних мереж, а також вести додаткові дії пов'язані з завантаженими знімками.

А.3 Вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно виконувати наступні функціональні вимоги:

- а) вимоги лікаря:
 - 1) авторизація;
 - 2) перегляд МРТ знімків пацієнтів;
 - 3) пошук МРТ знімків по пацієнту;
 - 4) діагностика знімку;
 - 5) пакетна діагностика знімків;
 - 6) пакетне завантаження МРТ знімків;
 - 7) розпізнавання знімків;
 - 8) редагування діагнозу МРТ знімка;
 - 9) збереження МРТ знімку в БД;

- 10) формування висновків до МРТ знімків;
 - 11) вихід з особистого кабінету.
- б) вимоги пацієнта:
- 1) реєстрація;
 - 2) авторизація;
 - 3) перегляд власних МРТ знімків;
 - 4) перегляд висновків лікаря до знімку;
 - 5) вихід з особистого кабінету.
- в) вимоги адміністратора:
- 1) авторизація;
 - 2) перегляд всіх користувачів;
 - 3) редагування даних користувача;
 - 4) видалення користувача;
 - 5) реєстрація нових лікарів;
 - 6) перегляд журналу помилок;
 - 7) ініціювати перенавчання нейронної мережі;
 - 8) зміна архітектури й ваг моделі для класифікації;
 - 9) вихід з панелі адміністратора.

А.3.2 Вимоги до надійності

Програмне забезпечення повинно відповідати таким вимогам безпеки:

- підтримка функцій захисту від несанкціонованого доступу (доступ до різних сторінок для лікарів, пацієнтів та адміністратора);
- обробка помилкових дій користувача з відповідними повідомленнями (валідація форм користувачів, перехід на неіснуючі сторінки);
- обробка виняткових ситуацій (наприклад, помилки при роботі з базою даних).

A.3.3 Умови експлуатації

Для розміщення розробленої програми у мережі Інтернет необхідно виділити вебвузол і встановити на ньому:

- сервер із підтримкою Python та Flask;
- бібліотека TensorFlow для реалізації нейронних мереж;
- база даних для зберігання інформації про користувачів, знімки та діагнози;
- доступ до мережі Інтернет для роботи з інтерфейсом та обробки запитів.

A.3.4 Вимоги до складу та параметрів технічних засобів

Для використання розробленої системи необхідні:

- процесор Intel Core i7 або еквівалент;
- мінімум 8 Гб оперативної пам'яті, рекомендовано 16 й більше;
- відеокарта: рекомендована відеокарта з підтримкою CUDA;
- операційна система: Windows 10 або Ubuntu 20.04 LTS.

A.4 Порядок контролю та приймання

Виконання дипломної кваліфікаційної роботи бакалавра має бути закінчено у восьми тижневий термін з розподіленням за етапами, що визначено завданням на роботу у відповідному календарному плані. Після виконання даного плану реалізується процедура захисту роботи. Захист відбувається перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Файл run.py

```
from app import app

if __name__ == "__main__":
    app.run(debug=True)
```

Б.2 Файл MRImageClassifier.py

```
import os, random, cv2, time, keras, datetime
from PIL import Image

import numpy as np
import pandas as pd
import tensorflow as tf
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from keras.optimizers import Adam, Adamax, SGD
from keras import regularizers

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.layers import BatchNormalization, MaxPooling2D, GlobalAveragePooling2D
import app.globals as globals

class MRImageClassifier:
    def __init__(self, network_name='OwnV2'):
        self.available_network_names = ['OwnV2', 'OwnV1', 'test', 'VGG16', 'VGG19', 'AlexNet', 'InceptionV3', 'EfficientNetV2', 'ResNet50', 'InceptionResNetV2']
        self.network_name = network_name
        self.model = None
        self.history = None

        self.img_size = (150, 150)
        self.img_shape = (self.img_size[0], self.img_size[1], 3)

        self.class_labels = {
            'glioma': 0,
            'meningioma': 1,
            'notumor': 2,
            'pituitary': 3,
        }
        self.num_classes = len(self.class_labels.keys())

        self.setup_environment()

    def setup_environment(self):
        """Configure GPU settings and logging level."""
```

```

physical_devices = tf.config.list_physical_devices('GPU')
if physical_devices:
    tf.config.experimental.set_memory_growth(physical_devices[0],
True)
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

def set_network_name(self, network_name):
    self.network_name = network_name

def load_model_from_checkpoint(self, checkpoint_path: str):
    """
    Load model from a checkpoint file.

    Parameters:
    - checkpoint_path (str): Path to the checkpoint file.
    """

    self._get_model()
    self.model.load_weights(checkpoint_path)

def classify_image(self, image_path: str):
    """
    Classify an image using a trained model.

    Parameters:
    - image_path (str): Path to the image file.

    Returns:
    - str: Predicted class label.
    """
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=self.img_shape)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    img_array /= 255.0

    predictions = self.model.predict(img_array)
    predicted_class_index = np.argmax(predictions[0])
    predicted_class_label = [label for label, index in
self.class_labels.items() if index == predicted_class_index][0]

    return predicted_class_label

def train(self, learning_rate=0.001, beta_1=0.9, beta_2=0.999, epochs=20,
batch_size=16, logging=True, model_summary=True, earllystop=True):
    self._get_model()
    if self.model == None:
        print('=== Incorrect network name ===')
        return

    start_time = time.time()

    seed = 1
    train_dir = './data/Training/'
    test_dir = './data/Testing/'

    test_datagen = ImageDataGenerator(rescale=1./255)
    train_datagen = ImageDataGenerator(rescale=1./255,
rotation_range=10,
brightness_range=(0.85, 1.15),
width_shift_range=0.002,

```

```

        height_shift_range=0.002,
        shear_range=12.5,
        zoom_range=0,
        horizontal_flip=True,
        vertical_flip=False,
        fill_mode="nearest"
    )

    self.test_generator = test_datagen.flow_from_directory(test_dir,
target_size=self.img_size,
batch_size=batch_size,
class_mode="categorical",
shuffle=False,
seed=seed)

    self.train_generator = train_datagen.flow_from_directory(train_dir,
target_size=self.img_size,
batch_size=batch_size,
class_mode="categorical",
seed=seed
    )

    steps_per_epoch = self.train_generator.samples // batch_size
    validation_steps = self.test_generator.samples // batch_size

    if model_summary: self.model.summary()

    self.model.compile(Adam(learning_rate=learning_rate, beta_1=beta_1,
beta_2=beta_2), loss= 'categorical_crossentropy', metrics=['acc'])

    if earllystop: model_es = EarlyStopping(monitor='val_acc',
min_delta=1e-9, patience=8, verbose=True)
    model_rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.3,
patience=5, verbose=True)
    model_cp = ModelCheckpoint(
        filepath='./checkpoints/model.epoch{epoch:02d}-
val_acc{val_acc:.4f}.hdf5',
        monitor='val_acc',
        save_freq='epoch',
        verbose=1,
        save_best_only=True,
        save_weights_only=True, )

    callbacks = [model_rlr, model_cp, model_es] if earllystop else
[model_rlr, model_cp]

    def on_epoch_end(epoch, val_acc):
        globals.progress = int((epoch + 1) / epochs * 100)

callbacks.append(tf.keras.callbacks.LambdaCallback(on_epoch_end=on_epoch_end)
)

    self.history = self.model.fit(self.train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,

```

```

        validation_data=self.test_generator,
        validation_steps=validation_steps,
        verbose=logging,
        callbacks=callbacks)

    loss, accuracy = self.model.evaluate(self.test_generator,
steps=validation_steps)
    self.training_time = time.time() - start_time

    print(f'Test loss: {loss:.4f}')
    print(f'Test accuracy: {accuracy:.4f}')
    print(f'Training time: {self.training_time:.4f}')
    return {'loss': loss, 'accuracy': accuracy, 'training_time':
self.training_time}

def plot_training_history(self):
    if self.history == None:
        print('=== No history to plot ===')
        return
    history = self.history.history
    acc_epochs = pd.DataFrame({'train': history['acc'], 'val':
history['val_acc']})
    loss_epochs = pd.DataFrame({'train': history['loss'], 'val':
history['val_loss']})

    px.line(acc_epochs, x=acc_epochs.index, y=acc_epochs.columns[0:],
title=f'Training and Evaluation Accuracy every Epoch for
"{self.network_name}"', markers=True).show()
    px.line(loss_epochs, x=loss_epochs.index, y=loss_epochs.columns[0:],
title=f'Training and Evaluation Loss every Epoch for "{self.network_name}"',
markers=True).show()

def plot_confusion_matrix(self, test_generator):
    if self.model == None:
        print('=== No model to test ===')
        return
    y_pred = self.model.predict(test_generator)
    y_pred = np.argmax(y_pred, axis=1)
    y_test = test_generator.classes
    cm = confusion_matrix(y_test, y_pred)
    print(cm)

def plot_data_distribution(self, train_labels, test_labels):
    train_class_counts = [len([x for x in train_labels if x == label]) for
label in self.class_labels.keys()]
    test_class_counts = [len([x for x in test_labels if x == label]) for
label in self.class_labels.keys()]

    fig = go.Figure()
    fig.add_trace(go.Pie(labels=[label.title() for label in
self.class_labels.keys()],
        values=train_class_counts,
        marker=dict(colors=['#FAC500', '#0BFA00',
'#0066FA', '#FA0000']),
        textinfo='percent+label+value',
        textfont=dict(size=20),
        hole=0.3,
        pull=[0.1, 0.1, 0.1, 0.1],
        domain={'x': [0, 0.3], 'y': [0.5, 1]}))

    fig.add_trace(go.Pie(labels=['Train', 'Test'],

```

```

        values=[len(train_labels), len(test_labels)],
        marker=dict(colors=['darkcyan', 'orange']),
        textinfo='percent+label+value',
        textfont=dict(size=20),
        hole=0.3,
        pull=[0.1, 0],
        domain={'x': [0.35, 0.65], 'y': [0.5, 1]})))

    fig.add_trace(go.Pie(labels=[label.title() for label in
self.class_labels.keys()],
        values=test_class_counts,
        marker=dict(colors=['#FAC500', '#0BFA00',
'#0066FA', '#FA0000']),
        textinfo='percent+label+value',
        textfont=dict(size=20),
        hole=0.3,
        pull=[0.1, 0.1, 0.1, 0.1],
        domain={'x': [0.7, 1], 'y': [0.5, 1]})))

    fig.update_layout(title='Data Distribution', grid={'rows': 1,
'columns': 3})
    fig.show()

def _get_model(self):
    """ Create a convolutional neural network model based on the specified
    architecture. """
    if self.network_name == 'test':
        self.model = Sequential([
            Conv2D(filters=16, kernel_size=3, activation='relu',
padding='same', input_shape=self.img_shape),
            MaxPooling2D(pool_size=2),
            Conv2D(filters=32, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=2),
            Flatten(),
            Dense(64, activation='relu'),

            Dense(self.num_classes, activation='softmax')
        ])
    elif self.network_name == 'VGG16':
        base_model = tf.keras.applications.VGG16(weights='imagenet',
include_top=False, input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),
        ])
    elif self.network_name == 'VGG19':
        base_model = tf.keras.applications.VGG19(weights='imagenet',
include_top=False, input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),
        ])
    elif self.network_name == 'AlexNet':
        self.model = tf.keras.Sequential([
            Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=self.img_shape),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),

```

```

        BatchNormalization(),

        Conv2D(filters=256,      kernel_size=(5,5),      strides=(1,1),
activation='relu', padding="same"),
        MaxPooling2D(pool_size=(3,3), strides=(2,2)),
        BatchNormalization(),

        Conv2D(filters=384,      kernel_size=(3,3),      strides=(1,1),
activation='relu', padding="same"),
        Conv2D(filters=384,      kernel_size=(3,3),      strides=(1,1),
activation='relu', padding="same"),
        Conv2D(filters=256,      kernel_size=(3,3),      strides=(1,1),
activation='relu', padding="same"),
        MaxPooling2D(pool_size=(3,3), strides=(2,2)),
        BatchNormalization(),

        Flatten(),
        Dense(4096,activation='relu'),
        Dropout(0.5),
        Dense(4096,activation='relu'),
        Dropout(0.5),
        Dense(self.num_classes,activation='softmax')
    ])

    elif self.network_name == 'InceptionV3':
        base_model =
tf.keras.applications.InceptionV3(weights='imagenet',      include_top=False,
input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),
        ])

    elif self.network_name == 'EfficientNetV2':
        base_model =
tf.keras.applications.EfficientNetV2B0(weights='imagenet', include_top=False,
input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),
        ])

    elif self.network_name == 'ResNet50':
        base_model = tf.keras.applications.ResNet50(weights='imagenet',
include_top=False, input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),
        ])

    elif self.network_name == 'InceptionResNetV2':
        base_model =
tf.keras.applications.inception_resnet_v2.InceptionResNetV2(weights='imagenet
', include_top=False, input_shape=self.img_shape)
        base_model.trainable = False
        self.model = tf.keras.Sequential([
            base_model,
            GlobalAveragePooling2D(),
            Dense(self.num_classes, activation='softmax'),

```

```

    ])

    elif self.network_name == 'OwnV1':
        self.model = tf.keras.Sequential([
            Conv2D(filters=256, kernel_size=(12,12), strides=(5,5),
activation='relu', input_shape=self.img_shape),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Conv2D(filters=512, kernel_size=(6,6), strides=(2,2),
activation='relu', padding="same"),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Flatten(),
            Dense(4096, activation='relu'),
            Dropout(0.5),
            Dense(self.num_classes, activation='softmax')
        ])
    elif self.network_name == 'OwnV2':
        self.model = Sequential([
            Conv2D(filters=64, kernel_size=(12,12), strides=(5,5),
activation='relu', input_shape=self.img_shape),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Conv2D(filters=128, kernel_size=(6,6), strides=(2,2),
activation='relu', padding="same"),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
            MaxPooling2D(pool_size=(3,3), strides=(2,2)),
            BatchNormalization(),

            Flatten(),
            Dense(1028, activation='relu'),
            Dropout(0.5),
            Dense(self.num_classes, activation='softmax')
        ])
    else: self.model = None

```

Б.3 Файл __init__.py

```

from flask import Flask
from app.config import Config

app = Flask(__name__)
app.config.from_object(Config)

from app import routes

```

Б.4 Файл config.py

```
class Config:
    SECRET_KEY = 'thisisasecretkey'
    SQLALCHEMY_DATABASE_URI = 'sqlite:///test.db'
    UPLOADED_IMAGES_DEST =
'E:\\06_Coding\\Python\\.ML\\BrainTumor\\app\\uploads'
```

Б.5 Файл forms.py

```
from flask_wtf import FlaskForm
from wtforms import SubmitField, StringField, PasswordField, SelectField,
DateField, TextAreaField, BooleanField
from flask_wtf.file import FileField, FileRequired, FileAllowed,
MultipleFileField
from wtforms.validators import InputRequired, Length, ValidationError, Email,
DataRequired, EqualTo
from flask_uploads import UploadSet, IMAGES
from app.models import User, Patient

mri_scans = UploadSet('images', IMAGES)

class UploadForm(FlaskForm):
    image = FileField('Upload Image',
validators=[FileAllowed(mri_scans, 'Images only!'), FileRequired('File field
should not be empty!')])
    search = StringField('Search Patient', validators=[InputRequired()])
    patient_id = StringField('Patient ID', validators=[InputRequired()])
    submit = SubmitField('Upload')

class BatchUploadForm(FlaskForm):
    images = MultipleFileField('Upload Images',
validators=[FileAllowed(mri_scans, 'Images only!'), FileRequired('File field
should not be empty!')])
    submit = SubmitField('Upload')

class SearchForm(FlaskForm):
    search = StringField('Search Patient', validators=[InputRequired()])
    patient_id = StringField('Patient ID', validators=[InputRequired()])
    submit = SubmitField('Search')

class ConclusionForm(FlaskForm):
    conclusion = TextAreaField('Conclusion', validators=[InputRequired()],
render_kw={"rows": 4})
    submit = SubmitField('Submit')

class DiagnosisForm(FlaskForm):
    diagnosis = StringField('Change diagnosis', validators=[DataRequired()])
    submit = SubmitField('Upload')

class LoginForm(FlaskForm):
    email = StringField('Email', validators=[InputRequired()],
render_kw={"placeholder": "Email"})
    password = PasswordField('Password', validators=[InputRequired(),
Length(min=5, max=50)], render_kw={"placeholder": "Password"})
    submit = SubmitField('Login')
```

```

class RegisterPatientForm(FlaskForm):
    email = StringField('Email',
        validators=[InputRequired(), Email()], render_kw={"placeholder":
        "Email"})
    username = StringField('Username',
        validators=[InputRequired(), Length(min=4, max=30)], render_kw={"placeholder":
        "Username"})
    password = PasswordField('Password',
        validators=[InputRequired(), Length(min=8, max=50)], render_kw={"placeholder":
        "Password"})
    confirm_password = PasswordField('Confirm Password',
        validators=[InputRequired(), EqualTo('password')], render_kw={"placeholder":
        "Confirm Password"})

    first_name = StringField('First Name', validators=[InputRequired(),
        Length(min=2, max=50)], render_kw={"placeholder": "First Name"})
    last_name = StringField('Last Name', validators=[InputRequired(),
        Length(min=2, max=50)], render_kw={"placeholder": "Last Name"})
    birth_date = DateField('Birth Date', validators=[InputRequired()],
        format='%Y-%m-%d')

    submit = SubmitField('Register')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user: raise ValidationError('Username already taken!')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user: raise ValidationError('That email is already taken. Please
        choose a different one.')

class RegisterDoctorForm(FlaskForm):
    email = StringField('Email',
        validators=[InputRequired(), Email()], render_kw={"placeholder":
        "Email"})
    username = StringField('Username',
        validators=[InputRequired(), Length(min=4, max=30)], render_kw={"placeholder":
        "Username"})
    password = PasswordField('Password',
        validators=[InputRequired(), Length(min=8, max=50)], render_kw={"placeholder":
        "Password"})
    confirm_password = PasswordField('Confirm Password',
        validators=[InputRequired(), EqualTo('password')], render_kw={"placeholder":
        "Confirm Password"})

    first_name = StringField('First Name', validators=[InputRequired(),
        Length(min=2, max=50)], render_kw={"placeholder": "First Name"})
    last_name = StringField('Last Name', validators=[InputRequired(),
        Length(min=2, max=50)], render_kw={"placeholder": "Last Name"})
    birth_date = DateField('Birth Date', validators=[InputRequired()],
        format='%Y-%m-%d', render_kw={"placeholder": "Birthdate YYYY-MM-DD"})
    specialty = StringField('Specialty', render_kw={"placeholder":
        "Specialty"})

    submit = SubmitField('Register')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user: raise ValidationError('Username already taken!')

```

```

def validate_email(self, email):
    user = User.query.filter_by(email=email.data).first()
    if user: raise ValidationError('That email is already taken. Please
choose a different one.')

class UserForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(),
Email()])
    role = SelectField('Role', choices=[('admin', 'Admin'), ('doctor',
'Doctor'), ('patient', 'Patient')], validators=[DataRequired()])
    first_name = StringField('First name', validators=[DataRequired()])
    last_name = StringField('Last name', validators=[DataRequired()])
    is_active = BooleanField('Active')
    submit = SubmitField('Update')
    delete = SubmitField('Delete')

```

Б.6 Файл globals.py

```

progress = 0
training_result = None

```

Б.7 Файл models.py

```

from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import datetime, timezone
import pytz

db = SQLAlchemy()
utc_plus_3 = pytz.timezone('Etc/GMT-3')

class ErrorLog(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    message = db.Column(db.Text)
    timestamp = db.Column(db.DateTime, default=datetime.now(utc_plus_3))

    def __repr__(self):
        return f"<ErrorLog {self.id}>"

class User(db.Model, UserMixin):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)

    email = db.Column(db.String(255), nullable=False, unique=True)
    username = db.Column(db.String(100), nullable=False, unique=True)
    password = db.Column(db.String(100), nullable=False)

    role = db.Column(db.Enum('admin', 'doctor', 'patient',
name='user_roles'), nullable=False)
    created_at = db.Column(db.DateTime(timezone=True), nullable=False,
default=datetime.now(timezone.utc))
    is_active = db.Column(db.Boolean, default=True)

```

```

        doctor          = db.relationship('Doctor',      back_populates='user',
uselist=False)
        patient          = db.relationship('Patient',    back_populates='user',
uselist=False)

class Doctor(db.Model):
    __tablename__ = 'doctors'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

    first_name = db.Column(db.String(255), nullable=False)
    last_name  = db.Column(db.String(255), nullable=False)
    birth_date = db.Column(db.Date, nullable=False)
    specialty  = db.Column(db.String(255))

    user       = db.relationship('User',      back_populates='doctor')
    mri_scans  = db.relationship('MRIScan',
back_populates='diagnosed_by_doctor')
    conclusions = db.relationship('Conclusion', back_populates='doctor')

class Patient(db.Model):
    __tablename__ = 'patients'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

    first_name = db.Column(db.String(255), nullable=False)
    last_name  = db.Column(db.String(255), nullable=False)
    birth_date = db.Column(db.Date, nullable=False)

    user       = db.relationship('User',      back_populates='patient')
    mri_scans  = db.relationship('MRIScan', back_populates='patient')

class MRIScan(db.Model):
    __tablename__ = 'mri_scans'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    patient_id = db.Column(db.Integer, db.ForeignKey('patients.id'),
nullable=False)

    file_name     = db.Column(db.String(255), nullable=False)
    upload_date   = db.Column(db.DateTime(timezone=True), nullable=False,
default=datetime.now(timezone.utc))

    diagnosis     = db.Column(db.String(255), nullable=False)
    diagnosed_by  = db.Column(db.Integer, db.ForeignKey('doctors.id'))

    conclusions   = db.relationship('Conclusion',
back_populates='mri_scan')
    patient       = db.relationship('Patient', back_populates='mri_scans')
    diagnosed_by_doctor = db.relationship('Doctor',
back_populates='mri_scans')

class Conclusion(db.Model):
    __tablename__ = 'conclusions'

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)

```

```

    mri_scan_id = db.Column(db.Integer, db.ForeignKey('mri_scans.id'),
                             nullable=False)

    doctor_id = db.Column(db.Integer, db.ForeignKey('doctors.id'),
                           nullable=False)
    conclusion = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime(timezone=True), nullable=False,
                           default=datetime.now(timezone.utc))

    mri_scan = db.relationship('MRI_Scan', back_populates='conclusions')
    doctor = db.relationship('Doctor', back_populates='conclusions')

```

Б.8 Файл routes.py

```

from MRIImageClassifier import MRIImageClassifier
from app import app
from app.forms import *
from app.models import *
import app.globals as globals

from flask import flash, jsonify, render_template, redirect, request, url_for,
send_from_directory
from flask_login import LoginManager, login_user, login_required, logout_user,
current_user
from flask_uploads import configure_uploads
from flask_bcrypt import Bcrypt
from werkzeug.security import generate_password_hash, check_password_hash

from datetime import datetime
from threading import Thread
import os, pytz, time

db.init_app(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
configure_uploads(app, mri_scans)
utc_plus_3 = pytz.timezone('Etc/GMT-3')

with app.app_context():
    def create_admin_user():
        admin_email = 'admin'
        admin_username = 'admin'
        admin_password = generate_password_hash('admin')
        admin_role = 'admin'

        existing_admin = User.query.filter_by(email=admin_email).first()
        if not existing_admin:
            admin_user = User(
                email=admin_email,
                username=admin_username,
                password=admin_password,
                role=admin_role
            )
            db.session.add(admin_user)
            db.session.commit()
            app.logger.info("Admin user created.")

```

```

        db.create_all()
        create_admin_user()

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.template_filter('formatdatetime')
def format_datetime(value, format='%d %B %Y %H:%M'):
    if value is None:
        return ""
    return value.strftime(format)

checkpoint_path = './checkpoints/OwnV2.epoch36-val_acc0.9922.hdf5'
model = MRIImageClassifier()
model.load_model_from_checkpoint(checkpoint_path)

@app.route('/', methods=['GET', 'POST'])
def index():
    return redirect(url_for('home'))

@app.route('/home', methods=['GET', 'POST'])
def home():
    return render_template('index.html')

@app.route('/register', methods=['GET', 'POST'])
def register_patient():
    form = RegisterPatientForm()
    if form.validate_on_submit():
        try:
            hashed_password = generate_password_hash(form.password.data)

            user = User(
                email=form.email.data,
                username=form.username.data,
                password=hashed_password,
                role='patient'
            )
            db.session.add(user)
            db.session.commit()

            patient = Patient(
                user_id=user.id,
                first_name=form.first_name.data,
                last_name=form.last_name.data,
                birth_date=form.birth_date.data
            )
            db.session.add(patient)
            db.session.commit()

            flash('Registration successful.', 'success')
            return redirect(url_for('login'))
        except Exception as e:
            db.session.rollback()
            app.logger.error(f"Error during registration: {str(e)}")
            flash('Registration failed.', 'danger')
    return render_template('register_patient.html', form=form)

@app.route('/login', methods=['GET', 'POST'])
def login():
    errors = []

```

```

form = LoginForm()
if form.validate_on_submit():
    user = User.query.filter_by(email=form.email.data).first()
    if user:
        if user.role == 'admin':
            login_user(user)
            flash('Logged in as admin.', 'success')
            return redirect(url_for('admin_profile'))
        elif check_password_hash(user.password, form.password.data):
            if user.is_active:
                login_user(user)
                flash('Logged in successfully.', 'success')
                return redirect(url_for('profile'))
            else:
                errors.append('User is not active.')
                flash('User is not active.', 'danger')
        else:
            errors.append('Invalid password.')
            flash('Invalid password.', 'danger')
    else:
        errors.append('User not found.')
        flash('User not found.', 'danger')
return render_template('login.html', form=form, errors=errors)

@app.route('/logout', methods=['GET', 'POST'])
@login_required
def logout():
    logout_user()
    flash('Logged out successfully.', 'success')
    return redirect(url_for('home'))

@app.route('/autocomplete', methods=['GET'])
def autocomplete():
    search = request.args.get('q')
    patients = Patient.query.filter(
        db.or_(Patient.first_name.ilike(f'%{search}%'),
            Patient.last_name.ilike(f'%{search}%'))
    ).all()
    results = [{ 'id': patient.id, 'name': f'{patient.first_name}
{patient.last_name}' } for patient in patients]
    return jsonify(matching_results=results)

@app.route('/mri_scans/<int:mri_scan_id>/conclusions', methods=['GET'])
def get_conclusions(mri_scan_id):
    mri_scan = MRIScan.query.get_or_404(mri_scan_id)
    conclusions = []
    for conclusion in mri_scan.conclusions:
        doctor = Doctor.query.filter_by(id=conclusion.doctor_id).first()
        conclusion_data = {
            'id': conclusion.id,
            'doctor': doctor.first_name + ' ' + doctor.last_name,
            'conclusion': conclusion.conclusion,
            'created_at': conclusion.created_at.strftime('%Y-%m-%d
%H:%M:%S'),
        }
        conclusions.append(conclusion_data)
    return jsonify(conclusions)

@app.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():

```

```

form = SearchForm()
if current_user.role == 'doctor':
    if form.validate_on_submit():
        patient_id = form.patient_id.data
        mri_scans = MRIScan.query.filter_by(diagnosed_by=current_user.doctor.id,
patient_id=patient_id).order_by(MRIScan.upload_date.desc()).all()
    else:
        mri_scans = MRIScan.query.filter_by(diagnosed_by=current_user.doctor.id).order_by(MRIScan
.upload_date.desc()).all()
        return render_template('profile_doctor.html', mri_scans=mri_scans,
user=current_user, form=form)
    else:
        mri_scans = MRIScan.query.filter_by(patient_id=current_user.patient.id).order_by(MRIScan.
upload_date.desc()).all()
        return render_template('profile_patient.html', mri_scans=mri_scans,
user=current_user)

@app.route('/profile/<filename>', methods=['GET', 'POST'])
@login_required
def image(filename):
    if current_user.role == 'doctor':
        mri_scan = MRIScan.query.filter_by(file_name=filename).first()
        if not mri_scan:
            flash("MRI scan not found.", "danger")
            return redirect(url_for('profile'))

        form = ConclusionForm()
        diagnosis_form = DiagnosisForm(diagnosis=mri_scan.diagnosis)

        if form.validate_on_submit():
            try:
                new_conclusion = Conclusion(
                    mri_scan_id=mri_scan.id,
                    doctor_id=current_user.doctor.id,
                    conclusion=form.conclusion.data,
                    created_at=datetime.now(utc_plus_3)
                )
                db.session.add(new_conclusion)
                db.session.commit()
                flash("Conclusion added successfully.", "success")
                return redirect(url_for('image', filename=filename))
            except Exception as e:
                db.session.rollback()
                app.logger.error(f"Error adding conclusion: {str(e)}")
                flash("Failed to add conclusion. Please try again.", "danger")

        if diagnosis_form.validate_on_submit():
            try:
                mri_scan.diagnosis = diagnosis_form.diagnosis.data
                db.session.commit()
                flash("Diagnosis updated successfully.", "success")
                return redirect(url_for('image', filename=filename))
            except Exception as e:
                db.session.rollback()
                app.logger.error(f"Error updating diagnosis: {str(e)}")
                flash("Failed to update diagnosis. Please try again.",
"danger")

```

```

        conclusions = Conclusion.query.filter_by(mri_scan_id=mri_scan.id).all()
        return render_template('conclusion_form.html', mri_scan=mri_scan,
                                conclusions=conclusions, form=form, diagnosis_form=diagnosis_form)
    else:
        flash('Access denied.', 'danger')
        return redirect(url_for('profile'))

@app.route('/delete_image', methods=['GET', 'POST'])
@login_required
def delete_image():
    if current_user.role != 'doctor':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    image_id = request.args.get('id')
    mri_scan = MRIScan.query.filter_by(id=image_id).first()
    if not mri_scan:
        flash("MRI scan not found.", "danger")
        return redirect(url_for('profile'))
    try:
        file_path = os.path.join(app.config['UPLOADED_IMAGES_DEST'],
mri_scan.file_name)
        os.remove(file_path)
        db.session.delete(mri_scan)
        db.session.commit()
        flash("Image deleted successfully.", "success")
    except Exception as e:
        db.session.rollback()
        app.logger.error(f"Error deleting file {mri_scan.file_name}: {str(e)}")
        flash("Failed to delete image.", 'danger')
        return redirect(url_for('profile'))

@app.route('/uploads/<filename>', methods=['GET', 'POST'])
def get_file(filename):
    try:
        return send_from_directory(app.config['UPLOADED_IMAGES_DEST'],
filename)
    except Exception as e:
        app.logger.error(f"Error sending file {filename}: {str(e)}")
        flash('Failed to retrieve file.', 'danger')
        return redirect(url_for('home'))

@app.route('/upload', methods=['GET', 'POST'])
@login_required
def upload():
    if current_user.role != 'doctor':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    form = UploadForm()
    if form.validate_on_submit():
        try:
            file_name = mri_scans.save(form.image.data)
            file_url = url_for('get_file', filename=file_name)
            patient = Patient.query.filter_by(id=form.patient_id.data).first()
            predicted_class = model.classify_image(os.path.join(app.config['UPLOADED_IMAGES_DEST'],
file_name))
            mri_scan = MRIScan(
                file_name=file_name,
                diagnosis=predicted_class,

```

```

        upload_date=datetime.now(utc_plus_3),
        patient=patient,
        diagnosed_by_doctor=current_user.doctor,
    )
    db.session.add(mri_scan)
    db.session.commit()
    flash('Image uploaded and classified successfully!', 'success')
    return render_template('upload.html', form=form,
text=predicted_class, file_url=file_url, user=current_user, mri_scan=mri_scan)
    except Exception as e:
        db.session.rollback()
        app.logger.error(f"Error uploading image: {str(e)}")
        flash("Failed to upload image.", "danger")
    return render_template('upload.html', form=form, user=current_user)

@app.route('/batch_upload', methods=['GET', 'POST'])
@login_required
def batch_upload():
    if current_user.role != 'doctor':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    form = BatchUploadForm()
    if form.validate_on_submit():
        uploaded_files = form.images.data
        results = []
        for uploaded_file in uploaded_files:
            if uploaded_file:
                try:
                    file_name = mri_scans.save(uploaded_file)
                    predicted_class =
model.classify_image(os.path.join(app.config['UPLOADED_IMAGES_DEST'],
file_name))
                    results.append((file_name, predicted_class))
                except Exception as e:
                    app.logger.error(f"Error during batch upload: {str(e)}")
                    flash("Failed to upload some images. Please try again.",
"danger")
                    flash('Images uploaded and classified successfully!', 'success')
        return render_template('batch_upload.html', form=form,
results=results, user=current_user)
    return render_template('batch_upload.html', form=form, user=current_user)

@app.route('/admin')
@login_required
def admin_profile():
    if current_user.role != 'admin':
        flash('Access unauthorized!', 'danger')
        return redirect(url_for('index'))
    return render_template('admin/admin_profile.html')

@app.route('/admin/users')
@login_required
def manage_users():
    if current_user.role != 'admin':
        flash('Access unauthorized!', 'danger')
        return redirect(url_for('index'))
    users = User.query.all()
    return render_template('admin/manage_users.html', users=users)

@app.route('/admin/users/update/<int:user_id>', methods=['GET', 'POST'])
@login_required
def update_user(user_id):

```

```

if current_user.role != 'admin':
    flash('Access unauthorized!', 'danger')
    return redirect(url_for('index'))
user = User.query.get(user_id)
if user.doctor:
    form = UserForm(obj=user, first_name=user.doctor.first_name,
last_name=user.doctor.last_name)
else:
    form = UserForm(obj=user, first_name=user.patient.first_name,
last_name=user.patient.last_name)

if form.validate_on_submit():
    user.username = form.username.data
    user.email = form.email.data
    user.role = form.role.data
    if user.doctor:
        user.doctor.first_name = form.first_name.data
        user.doctor.last_name = form.last_name.data
    else:
        user.patient.first_name = form.first_name.data
        user.patient.last_name = form.last_name.data
    user.is_active = form.is_active.data
    db.session.commit()
    flash('User updated successfully.', 'success')
    return redirect(url_for('manage_users'))
return render_template('admin/edit_user.html', form=form, user=user)

@app.route('/admin/users/delete/<int:user_id>', methods=['POST'])
@login_required
def delete_user(user_id):
    if current_user.role != 'admin':
        flash('Access unauthorized!', 'danger')
        return redirect(url_for('index'))

    user = User.query.get(user_id)
    if user:
        if user.role == 'doctor':
            doctor = Doctor.query.filter_by(user_id=user_id).first()
            if doctor:
                db.session.delete(doctor)
        elif user.role == 'patient':
            patient = Patient.query.filter_by(user_id=user_id).first()
            if patient:
                db.session.delete(patient)

        db.session.delete(user)
        db.session.commit()
        flash('User and related records deleted successfully.', 'success')
    else:
        flash('User not found.', 'danger')
    return redirect(url_for('manage_users'))

@app.route('/admin/register_doctor', methods=['GET', 'POST'])
@login_required
def register_doctor():
    if current_user.role != 'admin':
        flash('Access unauthorized!', 'danger')
        return redirect(url_for('index'))

    form = RegisterDoctorForm()
    if form.validate_on_submit():
        try:

```

```

        hashed_password = generate_password_hash(form.password.data)
        user = User(
            email=form.email.data,
            username=form.username.data,
            password=hashed_password,
            role='doctor'
        )
        db.session.add(user)
        db.session.commit()
        doctor = Doctor(
            user_id=user.id,
            first_name=form.first_name.data,
            last_name=form.last_name.data,
            specialty=form.specialty.data,
            birth_date=form.birth_date.data
        )
        db.session.add(doctor)
        db.session.commit()
        flash('Doctor registered successfully', 'success')
        return redirect(url_for('admin_profile'))
    except Exception as e:
        db.session.rollback()
        app.logger.error(f"Error during registration: {str(e)}")
        flash('Registration failed. Please try again.', 'danger')
    return render_template('register_doctor.html', form=form)

@app.errorhandler(Exception)
def handle_error(error):
    error_log = ErrorLog(message=str(error),
        timestamp=datetime.now(utc_plus_3))
    db.session.add(error_log)
    db.session.commit()
    return render_template('error.html', error=error_log), 500

@app.route('/trigger_error')
def trigger_error():
    raise Exception("This is a test error")

@app.route('/admin/errors')
@login_required
def view_errors():
    if current_user.role != 'admin':
        flash('Access unauthorized!', 'danger')
        return redirect(url_for('index'))
    errors = ErrorLog.query.order_by(ErrorLog.id.desc()).all()
    return render_template('admin/errors.html', errors=errors)

def retrain_model(architecture, epochs, batch_size, learning_rate, beta_1,
    beta_2):
    classifier = MRIImageClassifier(network_name=architecture)
    result = classifier.train(
        epochs=epochs,
        batch_size=batch_size,
        logging=True,
        learning_rate=learning_rate,
        beta_1=beta_1,
        beta_2=beta_2,
    )

    globals().training_result = result
    return result

```

```

@app.route('/progress', methods=['GET'])
@login_required
def progress_status():
    if current_user.role != 'admin':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    try:
        return jsonify(progress=globals.progress), 200
    except Exception as e:
        app.logger.error(f"Error fetching progress: {str(e)}")
        return jsonify({'error': 'Internal Server Error'}), 500

@app.route('/retrain-results', methods=['GET'])
@login_required
def retrain_results():
    if current_user.role != 'admin':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    try:
        return jsonify(result=globals.training_result), 200
    except Exception as e:
        app.logger.error(f"Error fetching retrain results: {str(e)}")
        return jsonify({'error': 'Internal Server Error'}), 500

@app.route('/admin/start-retrain', methods=['POST'])
@login_required
def start_retrain():
    if current_user.role != 'admin':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    globals.progress = 0
    globals.training_result = None
    try:
        data = request.form
        architecture = str(data['architecture'])
        epochs = int(data['epochs'])
        batch_size = int(data['batch_size'])
        learning_rate = float(data['learning_rate'])
        beta_1 = float(data['beta_1'])
        beta_2 = float(data['beta_2'])
        print(architecture)
        thread = Thread(target=retrain_model, args=(architecture, epochs,
batch_size, learning_rate, beta_1, beta_2))
        thread.start()
        return jsonify({'message': 'Retraining started! Please wait'}), 200
    except Exception as e:
        app.logger.error(f"Error starting retraining: {str(e)}")
        return jsonify({'message': 'Internal Server Error'}), 500

@app.route('/admin/retrain', methods=['GET', 'POST'])
@login_required
def retrain():
    if current_user.role != 'admin':
        flash('Access denied.', 'danger')
        return redirect(url_for('home'))
    architectures = model.available_network_names
    return render_template('admin/retrain.html', architectures=architectures)

@app.route('/admin/select_checkpoint', methods=['GET', 'POST'])
@login_required
def select_checkpoint():

```

```

if current_user.role != 'admin':
    flash('Access denied.', 'danger')
    return redirect(url_for('home'))
checkpoint_path = './checkpoints/OwnV2.epoch36-val_acc0.9922.hdf5'
checkpoints_dir = './checkpoints'
architectures = model.available_network_names
checkpoint_files = [f for f in os.listdir(checkpoints_dir) if
os.path.isfile(os.path.join(checkpoints_dir, f))]

if request.method == 'POST':
    selected_checkpoint = request.form.get('checkpoint')
    selected_architecture = request.form.get('architecture')
    message = None
    error = None
    if selected_checkpoint and selected_architecture:
        checkpoint_path = os.path.join(checkpoints_dir,
selected_checkpoint)
        try:
            model.network_name = selected_architecture
            model.load_model_from_checkpoint(checkpoint_path)
            flash(f'Successfully loaded checkpoint: {selected_checkpoint}
with architecture: {selected_architecture}', 'success')
            message = 'Checkpoint loaded successfully.'
        except Exception as e:
            app.logger.error(f"Error loading checkpoint
{selected_checkpoint} with architecture {selected_architecture}: {str(e)}")
            flash(f'Failed to load checkpoint: {selected_checkpoint} with
architecture: {selected_architecture}', 'danger')
            error = 'Failed to load checkpoint.'
        return render_template('admin/select_checkpoint.html',
checkpoint_files=checkpoint_files, architectures=architectures,
default_checkpoint=selected_checkpoint,
default_architecture=selected_architecture, message=message if message else
None, error=error if error else None)

    return render_template('admin/select_checkpoint.html',
checkpoint_files=checkpoint_files, architectures=architectures,
default_checkpoint=checkpoint_path, default_architecture='OwnV2')

```

Б.9 Файл base.html

```

<!DOCTYPE html>
<html lang="en" data-theme="dark">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/reset.css') }}" >
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@picocss/pico@2/css/pico.cyan.min.css"/>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css')
}}" >

    {% block head %}{% endblock %}
</head>
<body>
    <header class="container">

```

```

        <nav>
            <ul>
                <li><a href="{{url_for('home')}}"><strong>MRI
Classification</strong></a></li>
            </ul>
            <ul>
                {% if current_user.is_authenticated %}
                    {% if current_user.role == 'admin' %}
                        <li><a
href="{{url_for('admin_profile')}}">AdminPanel</a></li>
                    {% else %}
                        <li><a href="{{url_for('profile')}}">Profile</a></li>
                    {% endif %}
                {% endif %}

                {% if current_user.is_authenticated and current_user.doctor %}
                    <li><a href="{{url_for('upload')}}">Upload</a></li>
                    <li><a
href="{{url_for('batch_upload')}}">BatchUpload</a></li>
                {% endif %}

                {% if current_user.is_authenticated %}
                    <li><a href="{{url_for('logout')}}">Logout</a></li>
                {% endif %}

                {% if not current_user.is_authenticated %}
                    <li><a href="{{url_for('login')}}">Login</a></li>
                    <li><a
href="{{url_for('register_patient')}}">Register</a></li>
                {% endif %}
            </ul>
        </nav>
    </header>

    <main class="container">
        {% block content %}{% endblock %}
    </main>

    <footer class="container">
        {% block footer %}{% endblock %}
        <small>
            Kaminskyi Denys
        </small>
    </footer>
    {% block scripts %}{% endblock %}
</body>
</html>

```

Б.10 Файл batch_upload.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Batch Upload</title>
{% endblock %}

{% block content %}
    <div class="upload-container">

```

```

        <form id="uploadForm" method="POST" enctype="multipart/form-data"
onsubmit="handleSubmit()">
            {{ form.hidden_tag() }}

            <h2>Batch Upload Images</h2>

            {{ form.images(id="imageInput", onchange="previewImages(event)")
}}<br>
            {% for error in form.images.errors %}
                <span style="color: red;">{{ error }}</span>
            {% endfor %}

            {{ form.submit(id="submitButton") }}
        </form>

        <div class="grid grid-batch image-container preview"
id="previewContainer"> </div>

        {% if results %}
            <h2>Classification Results</h2>
            <div class="grid grid-batch">
                {% for file_name, predicted_class in results %}
                    <div class="grid-item image-container">
                        <div class="image-container-header">
                            <h3>{{ predicted_class }}</h3>
                        </div>
                        
                    </div>
                {% endfor %}
            </div>
        {% endif %}
    </div>
{% endblock %}

{% block scripts %}
    <script>
        function previewImages(event) {
            var files = event.target.files;
            var previewContainer =
document.getElementById('previewContainer');
            previewContainer.innerHTML = '';

            if (files) {
                Array.from(files).forEach(file => {
                    var reader = new FileReader();
                    reader.onload = function(e) {
                        var img = document.createElement('img');
                        img.src = e.target.result;
                        img.className = 'preview-item';
                        previewContainer.appendChild(img);
                    }
                    reader.readAsDataURL(file);
                });
            }
        }

        function clearImages() {
            document.getElementById('imageInput').value = "";
            document.getElementById('previewContainer').innerHTML = '';
        }
    </script>

```

```

        function handleSubmit() {
            var submitButton = document.getElementById('submitButton');
            submitButton.disabled = true;
        }
    </script>
{% endblock %}

```

Б.11 Файл conclusion_form.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Profile</title>
{% endblock %}

{% block content %}
    <div class="container conclusion-container">
        <div class="conclusion-container-item conclusion-container-header">
            <h2>MRI Scan: {{ mri_scan.file_name }}</h2>
            

            <aside class="conclusion-container-aside">
                <p><strong>First name: </strong>{{ mri_scan.patient.first_name
}}</p>
                <p><strong>Last name: </strong>{{ mri_scan.patient.last_name
}}</p>
                <p><strong>Diagnosis: </strong>{{ mri_scan.diagnosis }}</p>
                <p><strong>Diagnosed: </strong>{{ mri_scan.upload_date |
formatdatetime }}</p>
                <div class="line" style="border-color: black;"></div>
                <form
                    class="change-diagnosis-form"
                    method="POST"
enctype="multipart/form-data">
                    {{ diagnosis_form.hidden_tag() }}

                    {{ diagnosis_form.diagnosis.label }}
                    {{ diagnosis_form.diagnosis(class="change-diagnosis-input")
}}

                    {{ diagnosis_form.submit(class="change-diagnosis-submit") }}
                </form>
            </aside>

            <ul class="conclusion-container-item conclusions-list">
                {% for conclusion in conclusions %}
                    <li class="conclusion">
                        <div
                            class="doctor-name">Dr.
                                {{ conclusion.doctor.first_name }} {{ conclusion.doctor.last_name }}</div>
                        <div class="conclusion-date">{{ conclusion.created_at |
formatdatetime }}</div>
                        <div class="conclusion-content">{{ conclusion.conclusion
}}</div>
                    </li>
                {% endfor %}
            </ul>

            <form class="conclusion-container-item" method="POST">
                {{ form.hidden_tag() }}

```

```

        {{ form.conclusion.label }}
        {{ form.conclusion }}

        {{ form.submit }}
    </form>
</div>
{% endblock %}

```

Б.12 Файл error.html

```

{% extends "base.html" %}

{% block head %}
    <title>Error</title>
{% endblock %}

{% block content %}
    <h2>Error Details</h2>
    <table class="table">
        <thead>
            <tr>
                <th>#</th>
                <th>Error Message</th>
                <th>Timestamp</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>{{ error.id }}</td>
                <td>{{ error.message }}</td>
                <td>{{ error.timestamp }}</td>
            </tr>
        </tbody>
    </table>
{% endblock %}

```

Б.13 Файл index.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Main page</title>
{% endblock %}

{% block content %}
    <div class="index-container container">
        <h1>Інформація про Пухлини Головного Мозку</h1>
        <p>Вітаємо на нашому сайті! Ми розуміємо, що думка про пухлини головного мозку може викликати занепокоєння. Проте, важливо знати, що сучасна медицина має багато ефективних методів діагностики та лікування, які допомагають успішно боротися з цими захворюваннями.</p>
        <p>Пухлини головного мозку можуть бути як доброякісними, так і злоякісними. Доброякісні пухлини ростуть повільніше і не поширюються на інші частини тіла.
    </div>

```

Злоякісні пухлини, хоча і можуть бути агресивнішими, також піддаються лікуванню, особливо при ранній діагностиці.</p>
 <p>Наш програмний продукт створений з метою допомогти лікарям швидко та точно визначати наявність пухлин за допомогою знімків МРТ. Використовуючи сучасні методи штучного інтелекту, ми забезпечуємо високу точність аналізу, що дозволяє виявити проблему на ранніх стадіях та розробити оптимальний план лікування.</p>
 <p>Пам'ятайте, що вчасна діагностика – це ключ до успішного лікування. Регулярні обстеження та увага до свого здоров'я можуть значно підвищити шанси на одужання. Якщо у вас є будь-які запитання або ви хочете дізнатися більше про процес діагностики та лікування, не соромтеся звертатися до вашого лікаря.</p>
 <p>Ми працюємо для вашого здоров'я та добробуту, і наше програмне забезпечення є частиною цієї важливої місії. Завдяки новітнім технологіям та професіоналізму медичних працівників, ми разом зможемо зробити важливі кроки на шляху до вашого одужання.</p>
 </div>
 {% endblock %}

Б.14 Файл login.html

```
{% extends 'base.html' %}

{% block head %}
  <title>Login</title>
{% endblock %}

{% block content %}
  <div class="container form-container">
    <form method="POST">
      {{ form.hidden_tag() }}

      <h2>Login Form</h2>

      {% for error in errors %}
        <span style="color: red;">[{{ error }}]</span>
      {% endfor %}

      {{ form.email }}
      {% for error in form.email.errors %}
        <span style="color: red;">[{{ error }}]</span>
      {% endfor %}

      {{ form.password }}
      {% for error in form.password.errors %}
        <span style="color: red;">[{{ error }}]</span>
      {% endfor %}

      {{ form.submit }}
    </form>
  </div>
{% endblock %}
```

Б.15 Файл profile_doctor.html

```
{% extends 'base.html' %}

{% block head %}
    <title>Doctor profile</title>
{% endblock %}

{% block content %}
    <div class="profile-container container">
        <h2>Profile Dr. {{ current_user.doctor.first_name }} {{
current_user.doctor.last_name }}</h2>

        <form class="profile-search-form" method="POST" autocomplete="off">
            {{ form.hidden_tag() }}

            {{ form.search.label }}
            {{ form.search(id="patient-search") }}
            <ul id="patient-list" class="suggestions-list"></ul>
            {{ form.patient_id(id="patient-id", type="hidden") }}
            {{ form.submit }}
        </form>

        <div class="line"></div>

        <div class="grid cards">
            {% for mri_scan in mri_scans %}
            <div class="cards-item">
                
                <p>Patient: {{ mri_scan.patient.first_name }} {{
mri_scan.patient.last_name }}</p>
                <p>Diagnosis: {{ mri_scan.diagnosis }}</p>
                <p>Diagnosed: <br>{{ mri_scan.upload_date | formatdatetime
}}</p>
                <a href="{{ url_for('image', filename=mri_scan.file_name) }}"
class="btn">Write Conclusion</a>
                <a href="#" class="btn delete-btn" data-id="{{ mri_scan.id
}}">Delete</a>
            </div>
            {% endfor %}
        </div>
    </div>
{% endblock %}

{% block scripts %}
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            const deleteButtons = document.querySelectorAll('.delete-btn');

            deleteButtons.forEach(button => {
                button.addEventListener('click', function(event) {
                    event.preventDefault();
                    const imageId = this.getAttribute('data-id');
                    if (confirm('Are you sure you want to delete this image?'))
                    {
                        window.location.href = "{{ url_for('delete_image') }}"
+ "?id=" + imageId;
                    }
                });
            });
        });
    </script>
{% endblock %}
```

```

    });
  });

  document.addEventListener('DOMContentLoaded', function() {
    const searchInput = document.getElementById('patient-search');
    const patientList = document.getElementById('patient-list');
    const patientIdInput = document.getElementById('patient-id');

    searchInput.addEventListener('input', function() {
      const query = searchInput.value;
      if (query.length > 0) {
        fetch(`/autocomplete?q=${query}`)
          .then(response => response.json())
          .then(data => {
            patientList.innerHTML = '';
            if (data.matching_results.length > 0) {
              data.matching_results.forEach(patient => {
                const li = document.createElement('li');
                li.className = 'suggestion-item';
                li.textContent = patient.name;
                li.dataset.id = patient.id;
                patientList.appendChild(li);
              });
              patientList.style.display = 'block';
            } else {
              patientList.style.display = 'none';
            }
          });
      } else {
        patientList.innerHTML = '';
        patientList.style.display = 'none';
      }
    });

    patientList.addEventListener('click', function(event) {
      if (event.target.classList.contains('suggestion-item')) {
        const selectedPatient = event.target;
        searchInput.value = selectedPatient.textContent;
        patientIdInput.value = selectedPatient.dataset.id;
        patientList.innerHTML = '';
        patientList.style.display = 'none';
      }
    });

    document.addEventListener('click', function(event) {
      if (!searchInput.contains(event.target) &&
!patientList.contains(event.target)) {
        patientList.innerHTML = '';
        patientList.style.display = 'none';
      }
    });
  });
</script>
{% endblock %}

```

Б.16 Файл profile_patient.html

```

{% extends 'base.html' %}
{% block head %}

```

```

    <title>Patient profile</title>
{% endblock %}

{% block content %}
    <div class="profile-container container">
        <h1>Profile Patient. {{ current_user.patient.first_name }} {{
current_user.patient.last_name }} </h1>
        <h2>All your images</h2>

        <div class="grid cards">
            {% for mri_scan in mri_scans %}
            <div class="cards-item">
                
                <p>Uploaded: <br>{{ mri_scan.upload_date | formatdatetime
}}</p>

                {% if mri_scan.conclusions %}
                <a href="#" class="btn view-details-btn"
                    data-image="{{ url_for('get_file',
filename=mri_scan.file_name) }}"
                    data-conclusions="{{ url_for('get_conclusions',
mri_scan_id=mri_scan.id) }}"
                    data-diagnosis="{{ mri_scan.diagnosis }}"
                    >View doctor's conclusion</a>
                {% else %}
                <p style="color: red;">No doctor's conclusion</p>
                {% endif %}
            </div>
            {% endfor %}
        </div>

        <div id="modal-container" class="modal-container">
            <div class="modal-content">
                <span class="close-btn" onclick="closeModal()">&times;</span>
                
                <p>Diagnosis: <span id="modal-diagnosis"></span></p>
                <ul id="modal-conclusions-list"></ul>
            </div>
        </div>
    </div>
{% endblock %}

{% block scripts %}
    <script>
        document.addEventListener('DOMContentLoaded', function() {
            const viewDetailsButtons = document.querySelectorAll('.view-
details-btn');

            viewDetailsButtons.forEach(button => {
                button.addEventListener('click', function(event) {
                    event.preventDefault();
                    const imageSrc = this.getAttribute('data-image');
                    const conclusionsUrl = this.getAttribute('data-
conclusions');

                    const diagnosis = this.getAttribute('data-diagnosis');
                    fetch(conclusionsUrl)
                        .then(response => response.json())
                        .then(conclusions => {
                            console.log(conclusions);
                            openModal(imageSrc, conclusions, diagnosis);
                        })
                        .catch(error => {

```

```

                                console.error('Error fetching conclusions:',
error);
                                });
                            });
                        });

function openModal(imageSrc, conclusions, diagnosis) {
    const modalContainer = document.getElementById('modal-container');
    const modalImage = document.getElementById('modal-image');
    const modalConclusionsList = document.getElementById('modal-
conclusions-list');
    const modalDiagnosis = document.getElementById('modal-diagnosis');

    modalImage.src = imageSrc;
    modalConclusionsList.innerHTML = '';
    modalDiagnosis.textContent = diagnosis;

    conclusions.forEach(function(conclusion) {
        var listItem = document.createElement("li");
        listItem.classList.add("conclusion");

        var conclusionInfo = document.createElement("span");
        conclusionInfo.classList.add("conclusion-info");
        conclusionInfo.textContent = "Doctor: " + conclusion.doctor;
        listItem.appendChild(conclusionInfo);
        listItem.appendChild(document.createElement("br"));

        var created_at = document.createElement("span");
        created_at.classList.add("conclusion-created-at");
        created_at.textContent = "Created at: " +
conclusion.created_at;
        listItem.appendChild(created_at);
        listItem.appendChild(document.createElement("br"));

        var conclusionText = document.createElement("span");
        conclusionText.classList.add("conclusion-text");
        conclusionText.textContent = conclusion.conclusion;
        listItem.appendChild(conclusionText);

        modalConclusionsList.appendChild(listItem);
    });

    modalContainer.style.display = 'block';
}

function closeModal() {
    const modalContainer = document.getElementById('modal-container');
    modalContainer.style.display = 'none';
}
</script>
{% endblock %}

```

Б.17 Файл register_doctor.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Register doctor</title>

```

```

{% endblock %}

{% block content %}
    <div class="container form-container">
        <form method="POST">
            {{ form.hidden_tag() }}

            <h2>Register Form</h2>

            {{ form.email }}
            {% for error in form.email.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.username }}
            {% for error in form.username.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.password }}
            {% for error in form.password.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.confirm_password }}
            {% for error in form.confirm_password.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            <div class="line"></div>

            {{ form.first_name }}
            {% for error in form.first_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.last_name }}
            {% for error in form.last_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.specialty }}
            {% for error in form.specialty.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.birth_date.label(id="birthday-label") }}
            {{ form.birth_date }}
            {% for error in form.birth_date.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.submit() }}
        </form>
    </div>
{% endblock %}

```

Б.18 Файл register_patient.html

```
{% extends 'base.html' %}

{% block head %}
    <title>Register patient</title>
{% endblock %}

{% block content %}
    <div class="container form-container">
        <form method="POST">
            {{ form.hidden_tag() }}

            <h2>Register Form</h2>

            {{ form.email }}
            {% for error in form.email.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.username }}
            {% for error in form.username.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.password }}
            {% for error in form.password.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.confirm_password }}
            {% for error in form.confirm_password.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            <div class="line"></div>

            {{ form.first_name }}
            {% for error in form.first_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.last_name }}
            {% for error in form.last_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.birth_date.label(id="birthday-label") }}
            {{ form.birth_date }}
            {% for error in form.birth_date.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.submit }}
        </form>
    </div>
{% endblock %}
```

Б.19 Файл upload.html

```
{% extends 'base.html' %}

{% block head %}
    <title>Upload</title>
{% endblock %}

{% block content %}
    <div class="upload-container grid">
        <form method="POST" enctype="multipart/form-data" onsubmit="return
handleSubmit()" autocomplete="off">
            {{ form.hidden_tag() }}

            <h2>Upload Image</h2>

            <div class="line"></div>

            {{ form.search.label }}
            {{ form.search(id="patient-search") }}
            <ul id="patient-list" class="suggestions-list"></ul>
            <div id="patient-search-error" style="color: red; display:
none;">Please select a valid patient.</div>

            {{ form.patient_id(id="patient-id", type="hidden") }}
            {{ form.image(id="imageInput", onchange="previewImage(event)") }}
        <br>
        {% for error in form.image.errors %}
            <span style="color: red;">{{ error }}</span>
        {% endfor %}

        {{ form.submit(id="submitButton") }}
    </form>

    <div class="grid-item upload-image-container preview"
id="imagePreview" style="display: none;">
        <div class="image-container-header">
            <h3 class="preview-item">Selected Image</h3><br>
            <button type="button" onclick="clearImage()">Clear
Image</button>
        </div>
        
    </div>

    {% if text %}
    <div class="grid-item upload-image-container">
        <div class="image-container-header">
            <h3>Classification Result: <br><strong>{{ text
}}</strong></h3>
        </div>
        <div>
            
            <a href="{{ url_for('image', filename=mri_scan.file_name) }}"
class="btn upload-conclusion">Write Conclusion</a>
        </div>
    </div>
    {% endif %}
</div>
{% endblock %}
```

```

{% block scripts %}
<script>
    function previewImage(event) {
        var input = event.target;
        var preview = document.getElementById('preview');
        var previewDiv = document.getElementById('imagePreview');

        if (input.files && input.files[0]) {
            var reader = new FileReader();

            reader.onload = function (e) {
                preview.src = e.target.result;
            }

            reader.readAsDataURL(input.files[0]);
            previewDiv.style.display = 'flex';
        }
    }

    function clearImage() {
        document.getElementById('imageInput').value = "";
        document.getElementById('imagePreview').style.display = 'none';
    }

    function handleSubmit() {
        const patientIdInput = document.getElementById('patient-id');
        const patientSearchError = document.getElementById('patient-
search-error');

        if (!patientIdInput.value) {
            patientSearchError.style.display = 'block';
            return false;
        } else {
            patientSearchError.style.display = 'none';
        }

        return true;
    }

    document.addEventListener('DOMContentLoaded', function() {
        const searchInput = document.getElementById('patient-search');
        const patientList = document.getElementById('patient-list');
        const patientIdInput = document.getElementById('patient-id');

        searchInput.addEventListener('input', function() {
            const query = searchInput.value;
            if (query.length > 0) {
                fetch(`/autocomplete?q=${query}`)
                    .then(response => response.json())
                    .then(data => {
                        patientList.innerHTML = '';
                        if (data.matching_results.length > 0) {
                            data.matching_results.forEach(patient => {
                                const li = document.createElement('li');
                                li.className = 'suggestion-item';
                                li.textContent = patient.name;
                                li.dataset.id = patient.id;
                                patientList.appendChild(li);
                            });
                            patientList.style.display = 'block';
                        } else {
                            patientList.style.display = 'none';
                        }
                    });
            }
        });
    });
}

```

```

        }
    });
    } else {
        patientList.innerHTML = '';
        patientList.style.display = 'none';
    }
});

patientList.addEventListener('click', function(event) {
    if (event.target.classList.contains('suggestion-item')) {
        const selectedPatient = event.target;
        searchInput.value = selectedPatient.textContent;
        patientIdInput.value = selectedPatient.dataset.id;
        patientList.innerHTML = '';
        patientList.style.display = 'none';
    }
});

document.addEventListener('click', function(event) {
    if (!searchInput.contains(event.target) &&
!patientList.contains(event.target)) {
        patientList.innerHTML = '';
        patientList.style.display = 'none';
    }
});
});
</script>
{% endblock %}

```

Б.20 Файл admin_profile.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Admin panel</title>
{% endblock %}

{% block content %}
    <h1>Admin panel</h1>

    <h2>Actions</h2>
    <ul class="admin-actions">
        <li><a class="btn" href="{{ url_for('register_doctor') }}">Register
New Doctor</a></li>
        <li><a class="btn" href="{{ url_for('manage_users') }}">Manage
users</a></li>
        <li><a class="btn" href="{{ url_for('view_errors') }}">Errors
logs</a></li>
        <li><a class="btn" href="{{ url_for('retrain') }}">Retrain NN</a></li>
        <li><a class="btn" href="{{ url_for('select_checkpoint') }}">Select NN
checkpoint</a></li>
    </ul>
{% endblock %}

```

Б.21 Файл edit_user.html

```
{% extends 'base.html' %}

{% block head %}
    <title>Manage Users</title>
{% endblock %}

{% block content %}
    <div class="form-container container">
        <h1>Edit User {{ user.username }}</h1>
        <form class="edit-user-form" method="post">
            {{ form.hidden_tag() }}

            {{ form.username.label }}
            {{ form.username(size=32) }}
            {% for error in form.username.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.email.label }}
            {{ form.email(size=32) }}
            {% for error in form.email.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.first_name.label }}
            {{ form.first_name(size=32) }}
            {% for error in form.first_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            {{ form.last_name.label }}
            {{ form.last_name(size=32) }}
            {% for error in form.last_name.errors %}
                <span style="color: red;">[{{ error }}]</span>
            {% endfor %}

            <div class="checkbox">
                {{ form.is_active.label }}
                {{ form.is_active }}
                {% for error in form.is_active.errors %}
                    <span style="color: red;">[{{ error }}]</span>
                {% endfor %}
            </div>

            {{ form.submit() }}
        </form>
    </div>
{% endblock %}
```

Б.22 Файл errors.html

```
{% extends "base.html" %}
{% block head %}
```

```

    <title>Error Log</title>
{% endblock %}

{% block content %}
    <h2>Error Log</h2>
    <table class="table">
        <thead>
            <tr>
                <th>#</th>
                <th>Error Message</th>
                <th>Timestamp</th>
            </tr>
        </thead>
        <tbody>
            {% for error in errors %}
            <tr>
                <td>{{ error.id }}</td>
                <td>{{ error.message }}</td>
                <td>{{ error.timestamp }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}

```

Б.23 Файл manage_users.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Manage Users</title>
{% endblock %}

{% block content %}
    <h1>Manage Users</h1>
    <table>
        <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Email</th>
            <th>Role</th>
            <th>Active</th>
            <th style="text-align: center;">Actions</th>
        </tr>
        {% for user in users %}
        <tr>
            <td>{{ user.id }}</td>
            <td>{{ user.username }}</td>
            <td>{{ user.email }}</td>
            <td>{{ user.role }}</td>
            <td>{{ 'Yes' if user.is_active else 'No' }}</td>
            <td class="action-cell">
                {% if user.role == 'admin' %}
                <p style="margin: 0; ">----</p>
                {% else %}
                <a href="{% url_for('update_user', user_id=user.id) %}">Edit</a>
                <form action="{% url_for('delete_user', user_id=user.id) %}" method="post" class="manage-user-delete">

```

```

        <button type="submit">Delete</button>
    </form>
    {% endif %}

</td>
</tr>
{% endfor %}
</table>
{% endblock %}

```

Б.24 Файл retrain.html

```

{% extends "base.html" %}

{% block head %}
    <title>Retrain</title>
{% endblock %}

{% block content %}
    <div id="progress-bar" class="progress" style="margin: 15px;">
        <progress class="progress-bar" value="0" max="100"/>
    </div>
    <form id="retrain-form" class="form-container retrain-form container"
method="POST">
        <h1>Retrain Model</h1>
        <p id="message">Click the button below to initiate model
retraining.</p>

        <div class="form-group">
            <label for="architecture">Network name</label>
            <select name="architecture" id="architecture" class="form-
control">
                {% for arch in architectures %}
                    <option value="{{ arch }}" {% if arch ==
default_architecture %}selected{% endif %}>{{ arch }}</option>
                {% endfor %}
            </select>
        </div>
        <div class="form-group">
            <label for="epochs">Epochs</label>
            <input type="number" id="epochs" name="epochs" class="form-
control" value="20" required />
        </div>
        <div class="form-group">
            <label for="batch_size">Batch Size</label>
            <input type="number" id="batch_size" name="batch_size"
class="form-control" value="16" required />
        </div>
        <div class="form-group">
            <label for="learning_rate">Learning Rate</label>
            <input type="number" id="learning_rate" name="learning_rate"
step="0.0001" class="form-control" value="0.001" required />
        </div>
        <div class="form-group">
            <label for="beta_1">Beta 1</label>
            <input type="number" id="beta_1" name="beta_1" step="0.0001"
class="form-control" value="0.9" required />
        </div>
        <div class="form-group">

```



```

        } else {
            setTimeout(updateResults, 1500);
        }
    });
}

function updateResults() {
    fetch('/retrain-results')
    .then(response => response.json())
    .then(data => {
        const resultsPanel = document.getElementById("results-
panel");
        const resultText = document.getElementById("training-
result");
        const trainButton = document.getElementById("train-
button");
        const message = document.getElementById("message");
        const selectCheckpoint = document.querySelector('.select-
checkpoint');

        if (data.result) {
            resultsPanel.style.display = "block";
            resultText.innerHTML = `<p>Test      loss:
${data.result.loss.toFixed(4)}</p>
                                <p>Test      accuracy:
${data.result.accuracy.toFixed(4)}</p>
                                <p>Training    time:
${data.result.training_time.toFixed(4)} seconds</p>`;
            trainButton.disabled = false;
            message.innerHTML = "Training complete!";
            message.classList.remove("color-primary");
            selectCheckpoint.style.display = "block";
        } else {
            setTimeout(updateResults, 100);
        }
    });
}
</script>
{% endblock %}

```

Б.25 Файл select_checkpoint.html

```

{% extends 'base.html' %}

{% block head %}
    <title>Select Checkpoint</title>
{% endblock %}

{% block content %}
    <div class="container">
        <h1>Select Checkpoint and Architecture</h1>
        {% if message %}
            <p id="message" class="color-primary">{{ message }}</p>
        {% endif %}
        {% if error %}
            <p id="message" style="color: red;">{{ error }}</p>
        {% endif %}
        {% if not error and not message %}

```

```

        <p id="message" >Select desired checkpoint and appropriate
architecture</p>
        {% endif %}

        <form method="post">
            <div class="form-group">
                <label for="checkpoint">Select a checkpoint to load:</label>
                <select name="checkpoint" id="checkpoint" class="form-
control">
                    {% for file in checkpoint_files %}
                        <option value="{{ file }}" {% if file ==
default_checkpoint.split('/')[ -1] %}selected{% endif %}>{{ file }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="form-group">
                <label for="architecture">Select the neural network
architecture:</label>
                <select name="architecture" id="architecture" class="form-
control">
                    {% for arch in architectures %}
                        <option value="{{ arch }}" {% if arch ==
default_architecture %}selected{% endif %}>{{ arch }}</option>
                    {% endfor %}
                </select>
            </div>
            <button type="submit" class="btn btn-primary">Load
Checkpoint</button>
        </form>
    </div>
{% endblock %}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота бакалавра

Дослідження та програмна реалізація нейромережних методів розпізнавання
пухлини головного мозку за знімком МРТ

Research and Software Implementation of Neural Network Methods for
Recognition of Brain Tumor in MRI Images

Виконав
Студент групи КНТ-220

Денис КАМІНСЬКИЙ

Керівник роботи
к.т.н, доцент

Валерій ЛЬОВКІН

2024

Рисунок В.1 – Слайд 1

Мета роботи, об'єкт та предмет дослідження, задачі роботи

Мета роботи – дослідження та розробка програмного забезпечення з використання нейронних мереж для діагностування наявності пухлини мозку за знімком МРТ пацієнта.

Об'єкт дослідження – процес діагностування наявності пухлини мозку.

Предмет дослідження – методи та програмні засоби діагностування наявності пухлини мозку за знімком МРТ.

Задачі роботи:

- аналіз предметної області та розробка технічного завдання;
- експериментальне дослідження різних архітектур CNN;
- експериментальне дослідження архітектури з найбільшою точністю для подальшого покращення результату на основі запропонованого рішення;
- розробка вебінтерфейсу;
- проведення тестування програмного забезпечення.

Рисунок В.2 – Слайд 2

Вибір мови програмування

Критерій порівняння	Python	JavaScript	Java	C#
Складність вивчення	Низька	Середня	Середня	Висока
Швидкість виконання	Середня	Середня	Висока	Висока
Підтримка та спільнота	Дуже висока	Дуже висока	Висока	Висока
Підтримка нейронних мереж	Висока (TensorFlow, Keras, PyTorch)	Середня (TensorFlow.js)	Середня (DeepLearning4J)	Середня (ML.NET)
Підтримка вебфреймворків	Висока (Flask, Django)	Висока (Express.js)	Висока (Spring, Vaadin)	Висока (ASP.NET)
Кроссплатформеність	Висока	Висока	Висока	Висока

Рисунок В.3 – Слайд 3

Вибір вебфреймворку

Критерій порівняння	Django	Flask
Рівень складності	Висока	Середня
Масштабованість	Висока	Середня
Швидкість розробки	Низька	Висока
Інтеграція з ORM	Django ORM	SQLAlchemy
Вбудовані можливості	Велика кількість вбудованих функцій, включаючи аутентифікацію, адмін панель тощо	Мінімальні вбудовані можливості, що дозволяють використовувати тільки те, що потрібно
Ресурсоемність	Висока	Низька
Спільнота та підтримка	Висока	Висока

Рисунок В.4 – Слайд 4

Вибір бази даних

Критерій порівняння	SQLite	MySQL	PostgreSQL
Легкість налаштування	Висока	Низька	Середня
Продуктивність	Низька (для великих даних)	Висока	Висока
Масштабованість	Низька	Висока	Висока
Гнучкість у зміні схеми	Середня	Низька	Висока
Вимоги до ресурсів	Низькі	Середні	Середні
Підтримка ORM	Висока (SQLAlchemy)	Висока (SQLAlchemy)	Висока (SQLAlchemy)

Рисунок В.5 – Слайд 5

Функціональні вимоги

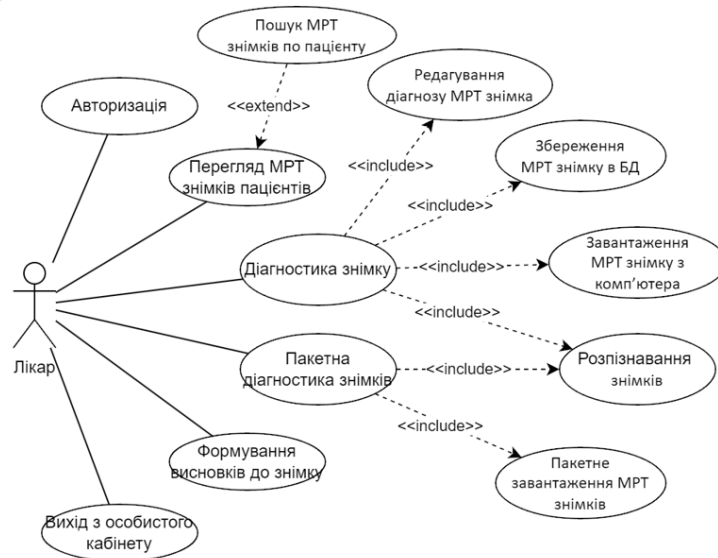


Рисунок В.6 – Слайд 6



Рисунок В.7 – Слайд 7



Рисунок В.8 – Слайд 8

Архітектура моделі

```

graph TD
    subgraph Left_Path [ ]
        direction TB
        L1[InputLayer  
input: [(None, 150, 150, 3)]  
output: [(None, 150, 150, 3)]] --> L2[Conv2D  
input: (None, 150, 150, 3)  
output: (None, 28, 28, 64)]
        L2 --> L3[MaxPooling2D  
input: (None, 28, 28, 64)  
output: (None, 13, 13, 64)]
        L3 --> L4[BatchNormalization  
input: (None, 13, 13, 64)  
output: (None, 13, 13, 64)]
        L4 --> L5[Conv2D  
input: (None, 13, 13, 64)  
output: (None, 7, 7, 128)]
        L5 --> L6[MaxPooling2D  
input: (None, 7, 7, 128)  
output: (None, 3, 3, 128)]
        L6 --> L7[BatchNormalization  
input: (None, 3, 3, 128)  
output: (None, 3, 3, 128)]
    end

    subgraph Right_Path [ ]
        direction TB
        R1[Conv2D  
input: (None, 3, 3, 128)  
output: (None, 3, 3, 256)] --> R2[MaxPooling2D  
input: (None, 3, 3, 256)  
output: (None, 1, 1, 256)]
        R2 --> R3[BatchNormalization  
input: (None, 1, 1, 256)  
output: (None, 1, 1, 256)]
        R3 --> R4[Flatten  
input: (None, 1, 1, 256)  
output: (None, 256)]
        R4 --> R5[Dense  
input: (None, 256)  
output: (None, 1028)]
        R5 --> R6[Dropout  
input: (None, 1028)  
output: (None, 1028)]
        R6 --> R7[Dense  
input: (None, 1028)  
output: (None, 4)]
    end

    L7 --> R7

```

Рисунок В.10 – Слайд 10

Результати
експериментального дослідження

Модель	Точність розпізнавання
EfficientNetV2	31.64%
ResNet50	70.31%
VGG19	80.62%
VGG16	85.46%
InceptionResNetV2	89.68%
InceptionV3	90.78%
AlexNet	98.90%
Модифікована модель AlexNet	99.22%

Рисунок В.11 – Слайд 11

Інтерфейс програми зі сторони лікаря

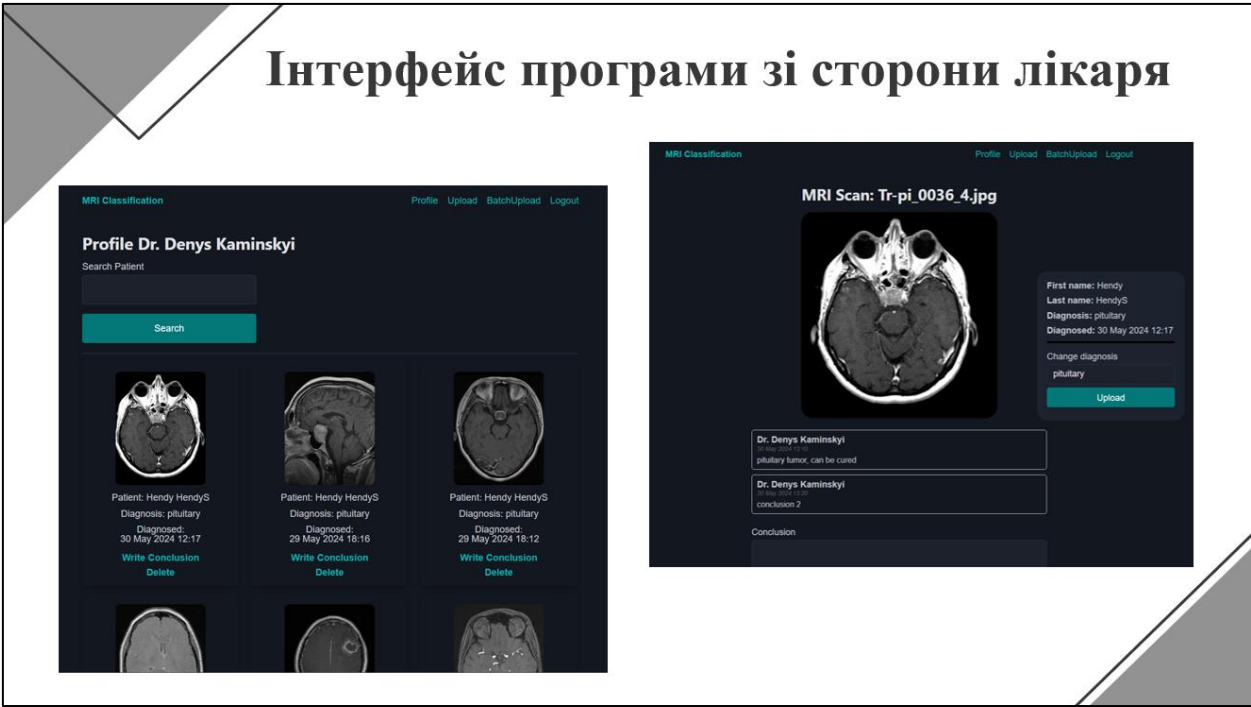


Рисунок В.12 – Слайд 12

Інтерфейс програми зі сторони лікаря

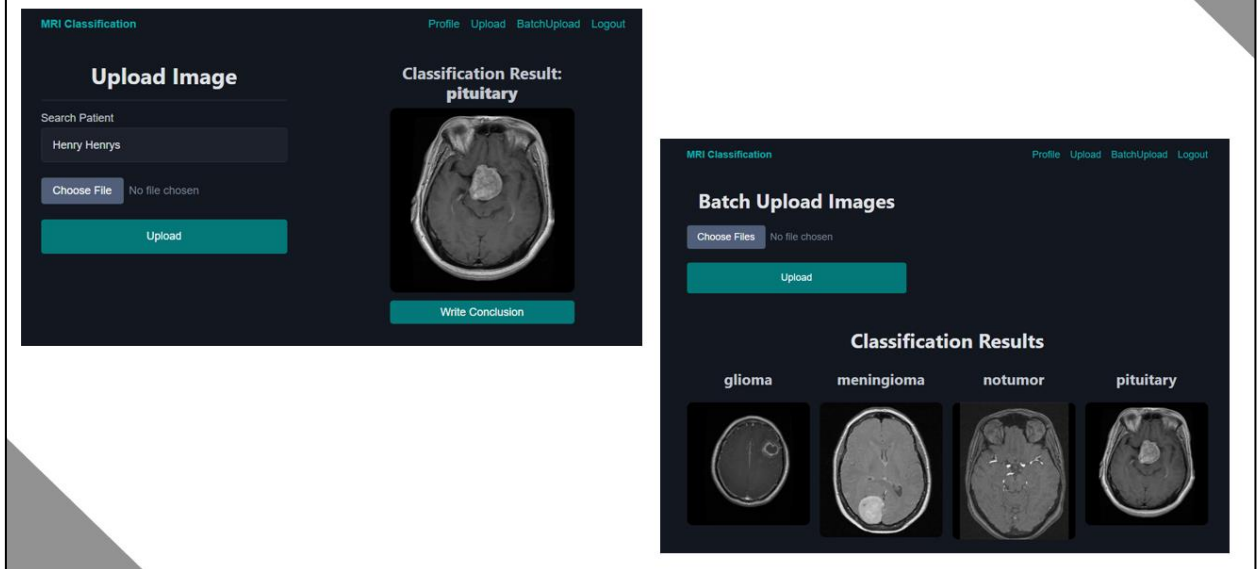


Рисунок В.13 – Слайд 13

Інтерфейс програми зі сторони пацієнта



Рисунок В.14 – Слайд 14

Інтерфейс програми зі сторони адміністратора

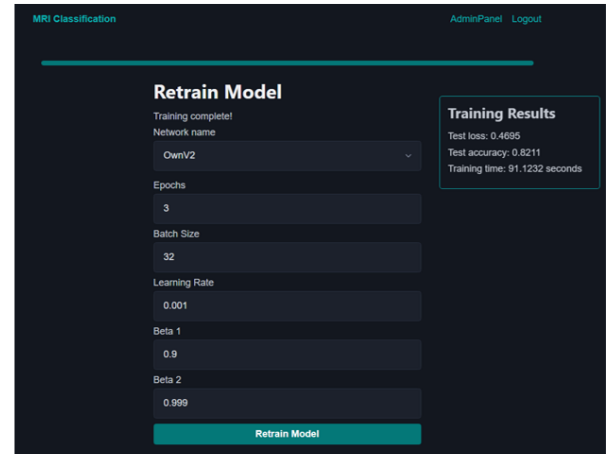
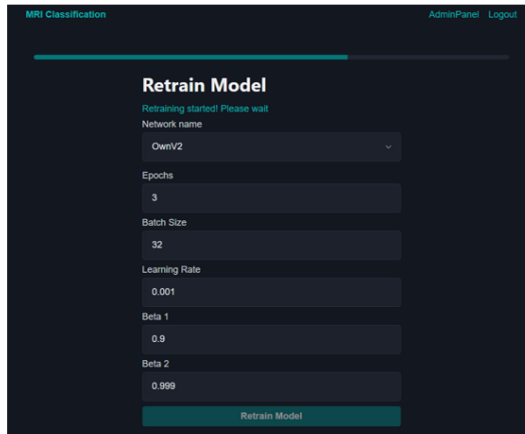


Рисунок В.15 – Слайд 15

Висновки

У процесі виконання дипломної кваліфікаційної роботи було розроблено програмне забезпечення для класифікації знімків МРТ на наявність пухлини головного мозку, призначене для допомоги лікарям у діагностуванні.

Розроблене програмне забезпечення має значну практичну цінність, оскільки спрощує процес діагностики пухлин головного мозку, надаючи лікарям додатковий інструмент для підтримки прийняття рішень. Програма не замінює лікаря, але значно полегшує та прискорює процес діагностування, що особливо важливо для своєчасного виявлення та лікування захворювань.

Наукова новизна роботи полягає у запропонованому методі діагностування пухлин головного мозку, який характеризується використанням модифікованої архітектури AlexNet. Ця модифікація дозволяє значно зменшити кількість налаштовуваних параметрів, що спрощує вимоги до навчання моделі, при цьому забезпечуючи покращення точності діагностування.

Рисунок В.16 – Слайд 16

Публікації

1. Камінський, Д.О. Програмне забезпечення для розпізнавання пухлини головного мозку за знімком магнітно-резонансної томографії [Текст] / Д. О. Камінський, В. М. Льовкін // Інформаційні моделюючі технології, системи та комплекси (ІМТСК-2024) : V міжнародна науково-практична конференція. 18-19 квітня 2024 р., Черкаси, Україна. – Черкаси : Черкаський національний університет імені Богдана Хмельницького, 2024. – С. 168-169.

2. Камінський, Д. О. Програмне забезпечення підтримки роботи лікаря для розпізнавання пухлини головного мозку [Електрон. ресурс] / Д. О. Камінський, В. М. Льовкін // Тиждень науки-2024. Факультет комп'ютерних наук і технологій. Тези доповідей науково-технічної конференції, Запоріжжя, 15-19 квітня 2024 р. / Редкол. : Вадим ШАЛОМЄЄВ (відпов. ред.) Електрон. дані. – Запоріжжя : НУ «Запорізька політехніка», 2024. – Препринт.

Рисунок В.17 – Слайд 17