НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

НН Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Моделі виявлення та сегментація пухлин головного мозку на знімках МРТ за допомогою глибинних нейронних мереж»

Виконав:
студент IV курсу, групи КА-93
Шевченко Владислав Вадимович _____

Керівник:
старший викладач кафедри ШІ, к.т.н. Шаповал Н. В. _____

Консультант з економічного розділу:
доцент, кандидат економічних наук, Рощина Н. В. _____

Рецензент:
доцент кафедри ІКТС ІТС НТУУ "КПІ",
к.т.н. Астраханцев А. А. _____

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

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
НН Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» травня 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Шевченко Владиславу Вадимовичу

1. Тема роботи «Моделі виявлення та сегментація пухлин головного мозку на знімках МРТ за допомогою глибинних нейронних мереж», керівник роботи Шаповал Наталія Віталіївна, кандидат технічних наук, старший викладач кафедри штучного інтелекту, затверджені наказом по університету від «__» травня 2023 р. № _____

2. Термін подання студентом роботи __.06.2023.

3. Вихідні дані до роботи – набір зображень знімків МРТ головного мозку

4. Зміст роботи – аналіз предметної області, огляд існуючих підходів, навчання моделей, створення програмного продукту, оцінка результатів.

5. Перелік ілюстративного матеріалу – презентація для захисту дипломної роботи, що складається з 13 слайдів, які містять постановку задачі, актуальність, аналіз побудованих моделей та результатів роботи програмного продукту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування тематики дослідження	15.04.2023 - 21.04.2023	Виконано
2	Огляд існуючих підходів для вирішення задачі	15.04.2023 - 28.04.2023	Виконано
3	Збір інформації, пошук набору даних	21.04.2023 - 28.04.2023	Виконано
4	Написання програми	28.04.2023 - 05.05.2023	Виконано
5	Оформлення пояснювальної записки	05.05.2023 - 12.05.2023	Виконано
6	Створення презентації для захисту	05.05.2023 - 19.05.2023	Виконано
7	Попередній захист дипломної роботи	19.05.2023 - 26.05.2023	Виконано
8	Захист дипломної роботи	19.05.2023 - 26.05.2023	Виконано

Студент Владислав ШЕВЧЕНКО

Керівник Наталія ШАПОВАЛ

РЕФЕРАТ

Дипломна робота: 97 с., 9 табл., 31 рис., 2 додатки, 16 джерел.

ПУХЛИНА МОЗКУ, МРТ, ГЛИБОКЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ТРАНСФЕРНЕ НАВЧАННЯ, ЗАЛИШКОВЕ НАВЧАННЯ, ЗАДАЧА КЛАСИФІКАЦІЇ, ЗАДАЧА СЕГМЕНТАЦІЇ.

Об’єкт дослідження – знімки магнітно-резонансної томографії головного мозку людини.

Предмет дослідження – методи обробки зображень, мережі для класифікації знімків та сегментації пухлин головного мозку.

Мета роботи – розробка програмного продукту для класифікації знімків головного мозку та у разі виявлення новоутворень їх сегментація з використанням методів глибокого навчання.

Результати – для задачі виявлення пухлини навчено моделі ResNet-50, VGG-16, DenseNet та запропоновану модель. Для задачі сегментації – ResUnet, ResUnet++ та Attention U-Net.

Моделі з найкращими результатами метрик якості використано при реалізації програмного продукту на мові програмування Python, який дозволяє класифікувати зображення магнітно-резонансної томографії головного мозку та виділяти пухлину на знімках, якщо вона існує.

ABSTRACT

Thesis: 97 p., 9 tabl., 31 fig., 2 appendices, 16 sources.

BRAIN TUMOR, DEEP LEARNING, CONVOLUTIONAL NEURAL NETWORKS, TRANSFER LEARNING, RESIDUAL LEARNING, CLASSIFICATION TASK, SEGMENTATION TASK.

The object of the research is magnetic resonance imaging (MRI) scans of the human brain.

The subject of the research is image processing methods, networks for classification of brain MRI scans and tumor segmentation.

The purpose of the work is development of a software product for brain MRI image classification and tumor segmentation using deep learning methods.

Results are basic models ResNet-50, VGG-16, DenseNet, and a proposed model were used for classification, while ResUnet, ResUnet++ and Attention U-Net were used for segmentation.

The models with the best quality metrics were used in the implementation of the software product in Python, which allows for classification of brain MRI images and identification of tumors on the scans if present.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ЗНМ – згорткова нейронна мережа

МРТ – магнітно-резонансна томографія

TP (True Positive) – істинно-позитивні значення, які означають, що передбачили позитивне, і це правильно

TN (True Negative) – істинно-негативні значення, які означають, що передбачили негативне, і це правильно

FP (False Positive) – хибно-позитивні значення, які означають, що передбачили позитивне, і це неправильно

FN (False Negative) – хибно-негативні значення, які означають, що передбачили негативне, і це неправильно

ВСТУП

Мозок є одним з найскладніших органів у людському тілі. Він складається із 100 мільярдів клітин. Неконтрольований ріст та поділ клітин може призвести до утворення пухлин, які можуть бути класифіковані як доброякісні або злоякісні в залежності від їхньої здатності до поширення і руйнування оточуючих тканин.

За даними Всесвітньої організації охорони здоров'я в 2020 році було зареєстровано більше 330 тисяч нових випадків пухлин головного мозку у всьому світі. Згідно із даними Національного канцер-реєстру України за 2020-2021 роки в Україні захворюваність на злоякісні новоутворення головного мозку складає 1778 випадків, з них 940 у чоловіків та 838 у жінок[1].

Аналіз знімків головного мозку є важливим етапом у лікуванні. Традиційний метод, який включає ручний аналіз зображень лікарем, має обмежену ефективність через його потенційну неточність та може призвести до несвоєчасного виявлення злоякісних пухлин. Крім того, некоректне визначення розмірів та меж пухлини збільшує ймовірність неправильного планування лікування та зменшує шанс на швидке одужання пацієнта.

Сучасні технології та методи для класифікації медичних зображень найчастіше використовують машинне навчання та глибоке навчання. Використання цих методів дозволяє автоматизувати процес аналізу медичних зображень. Це дозволить отримати більш точні результати, ніж традиційний підхід, завдяки здатності до автоматичного виявлення складних зв'язків у великих обсягах даних.

Сучасні алгоритми комп'ютерної обробки медичних зображень дозволяють автоматично визначати розміри та межі пухлини на знімках МРТ головного мозку.

Метою роботи є розробка програмного продукту для класифікації знімків головного мозку та у разі виявлення новоутворень їх сегментація з використанням методів глибокого навчання.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. АНАЛІЗ ЗАДАЧІ

1.1 Методи діагностики пухлин головного мозку

Інформація про вид пухлини в головному мозку дуже важлива для визначення стратегії лікування. Первинними пухлинами називають пухлини, які починаються саме в мозку. Вторинні пухлини це ті, які утворюються в інших частинах тіла та розповсюджується до мозку. Існує збільшений ризик розвитку вторинних пухлин у людей, близькі родичі яких мали таке захворювання. Причини утворення первинних пухлин – невідомі. Приклад пухлини головного мозку зображено на рисунку 1.1.

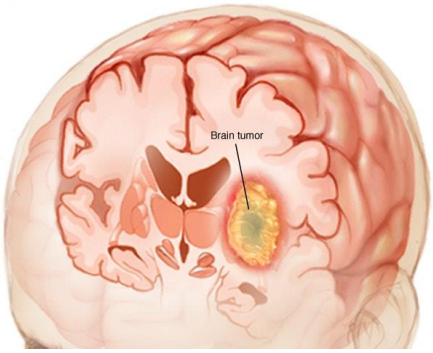
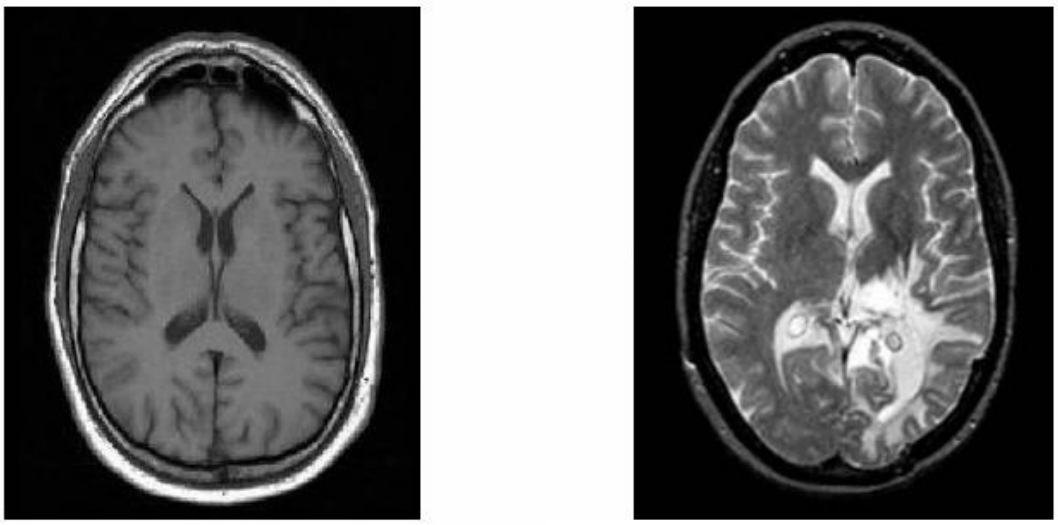


Рисунок 1.1 – Приклад головного мозку з пухлиною

Для діагностики пухлини головного мозку застосовують різні методи, такі як магнітно-резонансна томографія (МРТ), комп’ютерна томографія та біопсія.

МРТ є безпечним методом дослідження, оскільки не використовує іонізуюче випромінювання, на відміну від комп’ютерної томографії. МРТ використовує магнітні поля та радіохвилі, тому її можна призначати для пацієнтів у будь-якому віці.

МРТ містить великі обсяги даних, що вимагає високої обчислювальної потужності для їх аналізу. Машинне навчання має важливу роль у комп’ютерному аналізі зображень мозку, автоматизує процес класифікації знімків МРТ на нормальні та аномальні згідно з відсутністю або наявністю пухлини. Приклад знімків МРТ головного мозку зображено на рисунку 1.2.



(а) (б)

Рисунок 1.2 (а) – знімок МРТ головного мозку здорової людини, (б) – знімок МРТ головного мозку з пухлиною

Оскільки пухлини головного мозку швидко ростуть та можуть поширюватись на інші органи, даних про те, чи є пухлина на знімку МРТ недостатньо для призначення лікування. Визначення меж та розмірів пухлини на знімку МРТ дозволяє отримати детальну інформацію про її структуру. Це покращує якість діагностики, допомагає у виборі стратегії лікування та збільшує шанси на успішне лікування й повне одужання пацієнта.

Використання сучасних алгоритмів глибокого навчання дозволяє автоматизувати цей процес та збільшити точність локалізації пухлини.

1.2 Формалізація постановки задачі

Для задачі класифікації знімків МРТ головного мозку задано набір зображень та міток, які вказують на наявність або відсутність пухлини. Необхідно побудувати моделі класифікації знімків МРТ, які зможуть визначати чи є пухлина на знімку МРТ головного мозку. Також необхідно провести порівняльний аналіз моделей та обрати модель з найвищими показниками якості.

Для задачі сегментації пухлини головного мозку задано набір зображень та маска пікселів, що задають область пухлини. Необхідно побудувати сегментаційні моделі для локалізації пухлини та провести порівняльний аналіз моделей для вибору моделі з найвищими показниками якості.

1.3 Огляд існуючих підходів до аналізу МРТ знімків

Глибоке навчання використовується для задач обробки зображень, ідентифікації та класифікації МРТ знімків, сегментації пухлин. У статті Зайченка Ю. П., Здора К. А. та Гамідова Г. “Диагностика МРТ-изображений опухолей головного мозга с использованием гибридных сверточных нейронечетких сетей”[2] запропоновано різні методи обробки зображень, розроблено гібридна ЗНМ CNN-ANFIS із використанням VGG-16. У роботі проведено експериментальне дослідження створеної моделі для класифікації знімків МРТ, точність якої дорівнює 83%.

У роботі Чапалюка Б. В. та Зайченка Ю. П. “Використання рекурентних нейронних мереж для автоматичної діагностики раку легенів”[3] побудовано рекурентну нейронну мережу, яка складалась з двовимірної ЗНМ DenseNet та LSTM(Long short-term memory), точність розробленої моделі складає 72%. Також запропоновано підхід з додаванням механізму уваги, який допомагає мережі навчатися звертати увагу на дійсно злоякісні новоутворення. Точність даної моделі становить 81%.

У роботі Зайченка Ю. П. Надеран М. та Наполі А. “Використання згорткових нейронних мереж для діагностування раку молочної залози”[4], розроблено нову архітектуру гібридної згорткової мережі для розпізнавання раку молочної залози. У запропонованій моделі використано згортковий автоенкодер для пошуку інформативних ознак, що підвищує критерій якості моделі при меншому часі навчання. Точність розробленої моделі становить 93%.

У статті Nabila Abraham, Naimul Mefraz Khan “A Novel Focal Tversky loss function with improved Attention U-Net for lesion segmentation”[5] запропоновано узагальнену функцію втрат індексу Тверського для вирішення проблеми дисбалансу даних у сегментації медичних зображень. У порівнянні з функцією втрат Dice запропонована функція втрат забезпечує кращий компроміс між точністю та запам’ятовуванням під час навчання. Також в роботі наведено вдосконалену модель U-Net з механізмом уваги, що містить піраміду зображень для збереження контекстних особливостей. Експеримент навченої моделі показує кращі результати порівняно з класичною мережею U-Net на 26%.

У статті Harun Bingol, Bilal Alatas “Classification of Brain Tumor Images using Deep Learning Methods”[6] використано архітектури глибокого навчання Alexnet, Googlenet та Resnet-50 для виявлення пухлини на знімках МРТ. Найвищий показник точності, що становить 85.7%, досягнуто у моделі Resnet-50.

1.4 Висновки до розділу 1

У даному розділі розглянуто, що таке пухлина головного мозку та які існують методи діагностики головного мозку. Швидке та точне встановлення діагнозу є важливою складовою успішного лікування хворих. На сьогодні для аналізу зображень використовуються алгоритми глибокого навчання. Це допоможе лікарям швидко та якісно отримати результати МРТ, що сприятиме вибору стратегії лікування пацієнтів.

Проведено аналіз робіт із застосуванням згорткових нейронних мереж у сфері аналізу медичних зображень. Крім цього, формалізована постановка задачі класифікації знімків МРТ та сегментації пухлин за допомогою методів глибокого навчання.

2 ПІДХОДИ КЛАСИФІКАЦІЇ ТА ЛОКАЛІЗАЦІЇ ПУХЛИН З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

2.1 Комп’ютерний зір

Комп’ютерний зір (Computer Vision) – це напрям штучного інтелекту, який спеціалізується на обробці, класифікації та розпізнаванні інформації, отриманої із зображень та відео.

Комп’ютерний зір використовується в багатьох сферах:

- транспортні системи (розпізнавання дорожніх знаків, виявлення перешкод на дорозі, розрахунок безпечної швидкості під час руху, автоматичний доступ до паркінгу, відстеження руху автомобілів);
- агрономія (дослідження ґрунту, визначення стану рослин, виявлення захворювань рослин та шкідників);
- безпека (розпізнавання обличчя та ідентифікація осіб, контроль доступу до приміщень);
- медицина (діагностика медичних зображень).

У задачах комп’ютерного зору використовують глибинні нейронні мережі, а саме згорткові нейронні мережі.

2.2 Згорткові нейронні мережі

Згорткові нейронні мережі (Convolutional Neural Network) або скорочено ЗНМ – це клас глибинних штучних нейронних мереж прямого поширення, який застосовується до аналізу зображень. Однією з головних причин використання ЗНМ є їх функція розподілу ваг, яка сприяє зменшенню кількості параметрів мережі під час навчання. Це допомагає мережі посилити генералізацію та уникнути перенавчання.

Основними шарами ЗНМ є шар згортки (convolutional layer), шар пулінгу (pooling layer) та повнозв'язний шар (fully connected layer). Розгорнута схема роботи ЗНМ класифікатора зображено на рисунку 2.1.

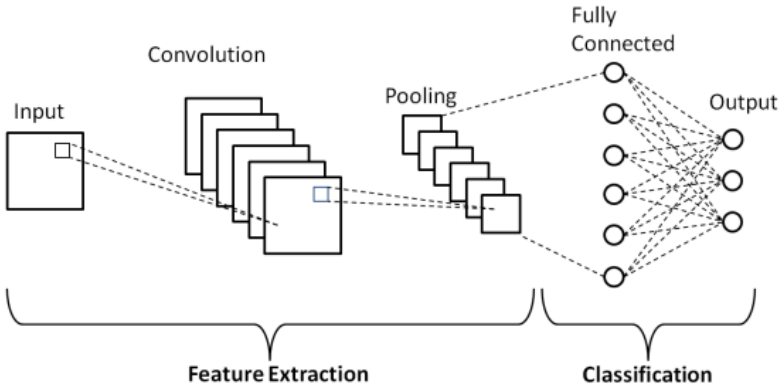


Рисунок 2.1 – Розгорнута схема роботи ЗНМ класифікатора

Згортка – процес застосування фільтра до зображення. Фільтр це матриця ваг фіксованого розміру (у більшості випадків 3×3), яка "ковзає" із певним кроком зсуву по оброблювальному шару. Чим більший крок зсуву тим більша ймовірність пропустити важливі ознаки.

На кожному етапі обчислюється скалярний добуток між фільтром згортки та частиною матриці такого ж розміру як і матриця фільтру. Результатом буде матриця значень (карта активації), яка складається із значень скалярного добутку по кожній частині зображення. Приклад застосування згортки показано на рисунку 2.2.

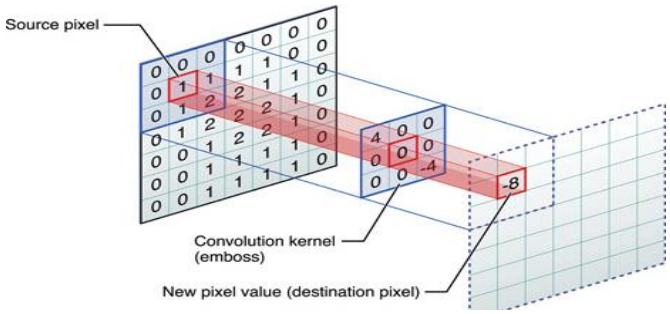


Рисунок 2.2 – Приклад застосування згортки

Функції активації дозволяють зробити мережу нелінійною. Найбільше застосовують функції гіперболічного тангенсу, sigmoid, ReLU.

Операція згортки є важливою складовою ЗНМ, оскільки дозволяє моделі автоматично визначити важливі ознаки вхідних даних.

Операція пулінгу – це процес зменшення розмірів зображення шляхом складання значень блоків пікселів. Початкове зображення ділиться на блоки розміром $w \times h$ та для кожного блоку обчислюється функція максимуму або середнього, результат якої додається в нову матрицю ознак.

Отже, результатом буде зменшене зображення з пікселями, які найкраще описують область зображення. Приклад застосування операції пулінгу зображено на рисунку 2.3.

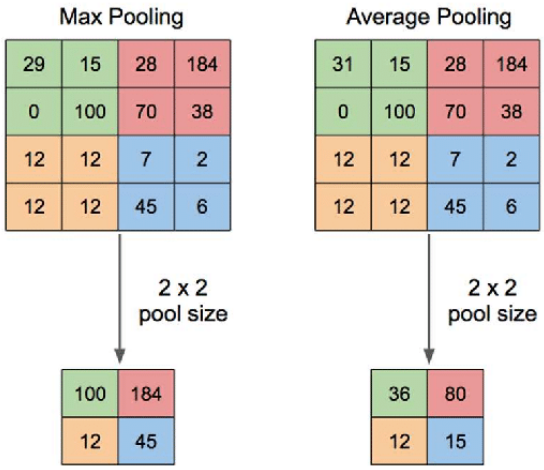


Рисунок 2.3 – Приклад застосування операції пулінгу

У ЗНМ відбувається чергування шарів згортки та пулінгу, кількість яких залежить від моделі.

Результатом згорткових та пулінгових операцій є отримання великої кількості окремих якісних характерних ознак. Після цього застосовується операція згладжування (Flatten) для кожної ознаки. Операція не змінює значення даних, а перетворює двовимірний вектор виходу ЗНМ у одновимірний. Потім за допомогою операції конкатенація (операція об’єднання декількох векторів в один більший вектор) утворюємо вектор, який можна передавати до повнозв’язного шару.

Повнозв’язний шар – це багатошаровий персептрон, за допомогою якого відбувається класифікація зображення. Графік застосування повноз’язного шару зображено на рисунку 2.4.

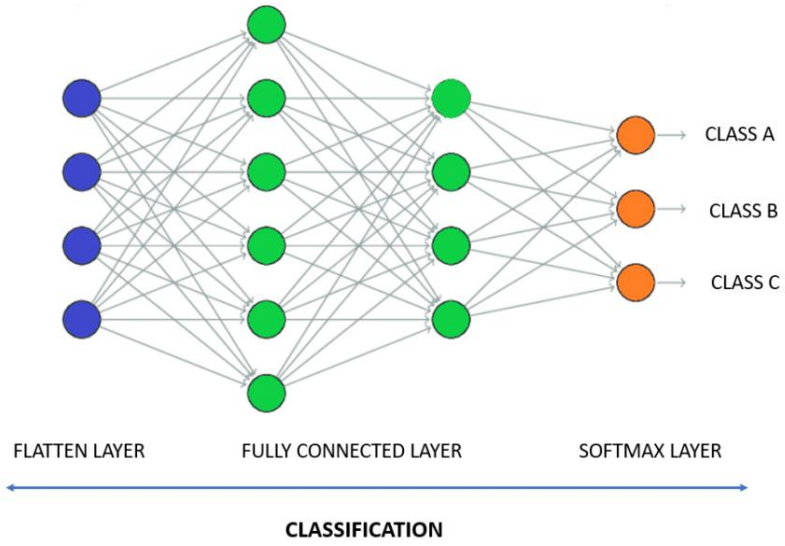


Рисунок 2.4 – Застосування повнозв’язного шару

Транспонована згортка (Transposed convolution) – це обернена операція до звичайної згортки, тобто замість зменшення розміру вхідних даних вона їх збільшує. Більшість згорткових мереж використовує транспоновану згортку для збільшення розміру вхідних даних, відновлення зображення до їх початкового розміру.

Процедура збільшення розміру необхідна для покращення якості зображення в задачах комп’ютерного зору, таких як, класифікація та сегментація пухлин.

Алгоритм застосування транспонованої згортки:

1) взяти карти ознак розміром 2×2, утворені після операції згортки;
2) в якості фільтра взяти матрицю ваг розміром 2×2;
3) кожен елемент вхідної карти ознак необхідно перемножити з кожним елементом ядра з одиничним кроком зсуву та нульовим доповненням;

4) якщо деякі елементи вихідної карти ознак перекриваються, їх необхідно додати;

5) результатом буде нова карта ознак із розміром 3×3.

На рисунку 2.5 зображено приклад операції транспонованої згортки.

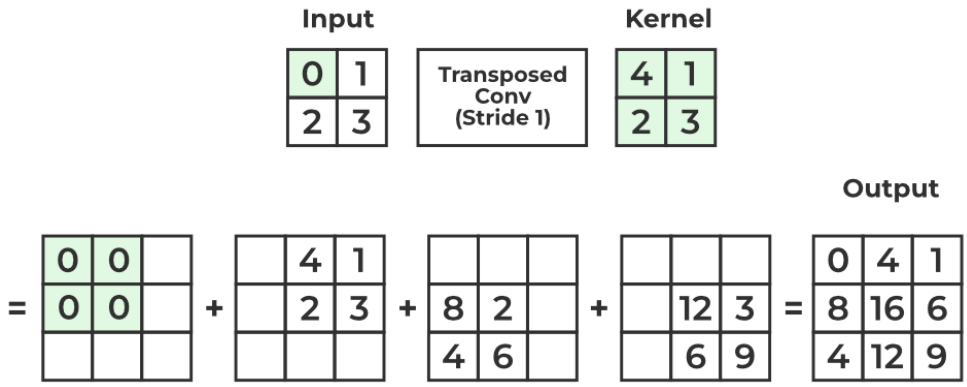


Рисунок 2.5 – Приклад операції транспонованої згортки

2.3 Критерій оцінки якості роботи моделей класифікації

Введемо позначення:

- $y \in \{0,1\}$ – істинна ймовірність приналежності пікселя до позитивного класу;
- $\hat{y} \in \{0,1\}$ – прогнозована ймовірність належності пікселя до позитивного класу.

Для визначення якості моделі застосовується функція втрат: бінарна хрест-ентропія (Binary Cross-Entropy), яка задається формулою (2.1).

$$\text{loss} = -(y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})). \quad (2.1)$$

Також для аналізу результатів застосовуються метрики accuracy, precision, recall та F-score.

Метрика accuracy (точність) визначає, яка частина зображень була класифікована правильно та обчислюється за формулою (2.2).

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.2)$$

Метрика precision (точність) визначає на скільки точно класифікатор ідентифікує знімки МРТ із пухлиною та обчислюється за формулою (2.3).

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.3}$$

Метрика recall (повнота) визначає на скільки добре класифікатор виявляє знімки МРТ із пухлиною та обчислюється за формулою (2.4).

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.4}$$

Метрика F-міра це гармонічне середнє між точністю та повнотою й обчислюється за формулою (2.5).

$$\text{F\_score} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.5}$$

2.4 Критерій оцінки якості роботи моделей сегментації пухлини

Використання класичних показників таких, як метрика accuracy, для сегментації пухлин не є точною, оскільки важливо не лише правильно класифікувати пікселі зображення, а й визначити розмір та форму пухлини.

Точність діагностики пухлини має велике значення у лікуванні пацієнта, тому для задач сегментації у медичній сфері використовують Tversky index[5] та його альтернативи.

Нехай $\alpha, \beta > 0$, $y, \hat{y} \in \{0,1\}^{M}$. Тоді, Tversky index визначається як:

$$T_{\alpha,\beta}(y, \hat{y}) = \frac{y\hat{y}}{y\hat{y} + \alpha(1-y)\hat{y} + \beta y(1-\hat{y})} \tag{2.6}$$

Або використовуючи терміни матриці невідповідності:

$$\text{T}_{\alpha,\beta} = \frac{\text{TP}}{\text{TP} + \alpha\text{FP} + \beta\text{FN}} \tag{2.7}$$

Smooth Tversky index є покращеною версією Tversky index, яка дозволяє зменшити вплив шуму, випадкових помилок та підвищити точність моделі.

Smooth Tversky index визначається як:

$$\mathrm{T}_{\alpha,\beta}(\mathrm{y},\hat{\mathrm{y}}) = \frac{\mathrm{y}\hat{\mathrm{y}} + \delta}{\mathrm{y}\hat{\mathrm{y}} + \alpha(1-\mathrm{y})\hat{\mathrm{y}} + \beta\mathrm{y}(1-\hat{\mathrm{y}}) + \delta} \quad (2.8)$$

де

δ – значення згладжування.

Під час навчання глибинних нейронних мереж потрібна диференційована функція втрат, яка оцінює подібність цілі та прогнозу. Функція Tversky loss обчислюється за формулою (2.9).

$$\mathcal{L}_{\mathrm{Tversky}} = 1 - \mathrm{T}_{\alpha,\beta}(\mathrm{y},\hat{\mathrm{y}}). \quad (2.9)$$

Вона є ефективною функцією втрат для сегментації пухлин з таких причин:

- функція є ефективною для обробки нерівномірних даних. МРТ знімок має нерівномірний розподіл класів, тобто один клас може бути набагато більший за інший.
- є можливість налаштовувати гіперпараметри.

Focal Tversky loss – це модифікована функція втрат, яка вводить параметр фокусування γ для зменшення ваги внеску добре класифікованих пікселів і збільшення внеску неправильно класифікованих пікселів.

$$\mathcal{L}_{\text{Focal Tversky}} = 1 - \mathrm{T}_{\alpha,\beta}(\mathrm{y},\hat{\mathrm{y}})^{\gamma} \quad (2.10)$$

де

γ −параметр фокусування.

При зменшенні кількості добре класифікованих пікселів Focal Tversky Loss може покращити здатність моделі обробляти незбалансовані дані та збільшити стійкість до помилок.

Альтернативними оцінками Tversky index є Dice index та Jaccard index.

Dice index – метрика, яка ґрунтується на Tversky index із параметрами $\alpha = 0.5, \beta = 0.5$. Метрика обчислюється за допомогою формули (2.11). У позначеннях матриці невідповідності метрика задається формулою (2.12).

$$Dice(y, \hat{y}) = T_{0.5,0.5}(y, \hat{y}) = \frac{y\hat{y}}{y\hat{y} + 0.5(1-y)\hat{y} + 0.5y(1-\hat{y})} \quad (2.11)$$

$$Dice = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \quad (2.12)$$

Jaccard index – метрика, яка ґрунтується на Tversky index із параметрами $\alpha = 1, \beta = 1$. Метрика обчислюється за допомогою формули (2.13). У позначеннях матриці невідповідності метрика задається формулою (2.14).

$$\mathrm{Jaccard}(y, \hat{y}) = T_{1,1}(y, \hat{y}) = \frac{y\hat{y}}{y\hat{y} + 1(1-y)\hat{y} + 1y(1-\hat{y})} \quad (2.13)$$

$$\mathrm{Jaccard} = \frac{TP}{TP + FP + FN} \quad (2.14)$$

2.5 Модель оптимізації

Оптимізатор Adam (Adaptive moment estimation)[7] використовує стохастичний градієнтний спуск (SGD), де крок навчання залежить від градієнту та моменту (попереднього градієнту). Це означає, що під час оптимізації, Adam використовує знання про мінімальне та максимальне значення градієнту для кожного параметра.

Він контролює швидкість навчання для оновлення ваги та зберігає альфа- швидкість незмінною під час запуску моделі. Для багатьох параметрів він обчислює конкретні темпи навчання з оцінок, які є першого та другого порядку моментів градієнта. Adam використовує дуже невеликі ресурси пам’яті. Ідея оптимізатора: більш слабке оновлення ваг для типових ознак згладженої версії середнього і середньоквадратичного градієнтів. Формули (2.15) використовуються для обчислення оптимізатора Adam.

$$\begin{aligned} \mathrm{m_t} &= \beta_1 \mathrm{m_{t-1}} + (1-\beta_1)\mathrm{g_t}, \\ \mathrm{v_t} &= \beta_2 \mathrm{v_{t-1}} + (1-\beta_2)\mathrm{g_t^2}, \\ \widehat{\mathrm{m_t}} &= \frac{\mathrm{m_t}}{1-\beta_1^{\mathrm{t}}}, \end{aligned} \quad (2.15)$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

де

$$0 \ll \beta_1,\ \beta_2 \ll 1$$

Для оновлення ваг використовуються формули (2.16).

$$\begin{aligned} w(t) &= w(t) + \Delta w(t), \\ \Delta w(t) &= -\frac{\alpha \widehat{m_t}}{\sqrt{v_t + \varepsilon}} \end{aligned} \tag{2.16}$$

Використання оновлених версій моментів дозволяє більш точно оцінити значення градієнтів на ранніх етапах навчання та знизити вплив зміщення оцінки моменту.

2.6 Функції активації

Функцію активації ReLU (Rectified Linear Unit) часто використовують у прихованих шарах ЗНМ. Вона швидко обчислюється, не спричиняє втрати градієнта вхідних даних. Це прискорює навчання моделі. Функція задається формулою (2.17).

$$f(x) = \max(0, x) \tag{2.17}$$

Графік функції активації ReLU зображено на рисунку 2.6.

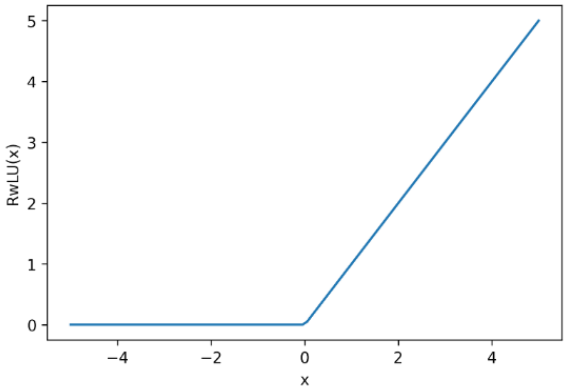


Рисунок 2.6 – Графік функції активації ReLU

Функція активації Sigmoid використовується на вихідному шарі нейронної мережі. Sigmoid функція є гладкою, диференційованою та неперервною, що дозволяє використовувати алгоритми градієнтного спуску для оптимізації ваг моделі. Функція задається формулою (2.18).

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

Графік функції активації Sigmoid зображено на рисунку 2.7.

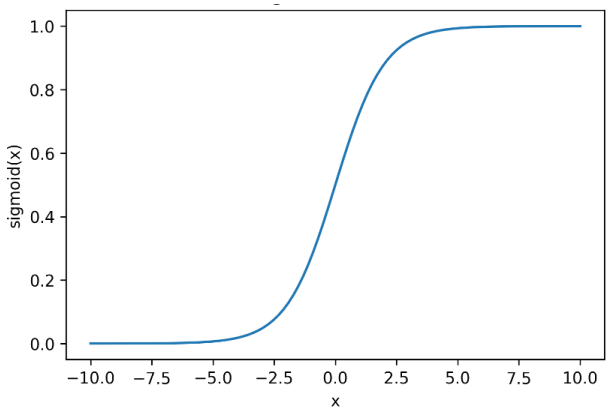


Рисунок 2.7 – Графік функції активації Sigmoid

2.7 Регуляризація ЗНМ

Однією з головних проблем ЗНМ є перенавчання, коли модель демонструє високу точність на тренувальних даних, але показує гірші результати на тестових даних. Модель вважається недостатньо навченою, коли вона не досягає успішності на тренувальних даних.

Існує декілька способів регуляризації ЗНМ. Основні з яких мають наступний вид.

1. Dropout. Метод виконує випадкове занулення деяких вихідних значень під час тренування моделі, що змушує модель розподіляти ваги між усіма функціями вхідних параметрів, а не зосереджуватися на конкретних функціях.

2. Batch Normalization. Шар додається між лінійним та активаційним шарами мережі. Він допомагає зменшити проблему зникаючого та вибухаючого градієнта, збільшити швидкість тренування. Batch Normalization регулює значення активаційного шару, змінюючи його дисперсію та середнє значення, що дозволяє збільшити стійкість мережі та уніфікувати вихідні значення шару.
3. Early stopping. Метод зупиняє тренування моделі, коли показники втрат не покращуватися на окремих епохах тренування.

2.8 Залишкове навчання

Для навчання моделей із великою кількістю шарів необхідно використовувати обхідні зв'язки (skip connection). Сигнал, який передається в шар, також додається до виходу шару, який розташовано трохи вище у послідовності шарів.

При навчанні нейронної мережі необхідно навчити її моделювати цільову функцію $h(x)$. Якщо додати вхід x до виходу мережі, тобто додати обхідний зв'язок, то мережа буде примусово моделювати $f(x) = h(x) - x$, а не $h(x)$. Таким чином утворюється залишковий шар (residual block). На виході отримаємо $h(x) = f(x) + x$. Схема роботи залишкового блоку зображена на рисунку 2.8.

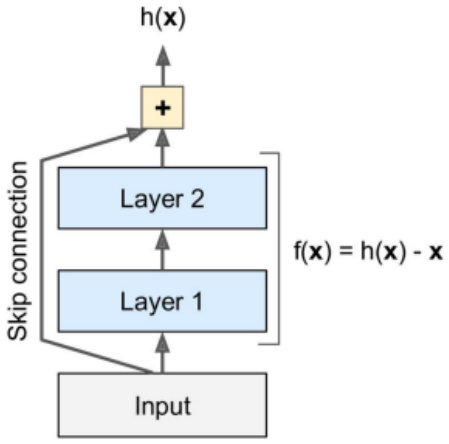


Рисунок 2.8 – Схема роботи залишкового блоку

Перевагою додавання обхідних зв’язків є те, шо це дозволяє більш ефективно передавати градієнти через мережу, а отже, ранні шари мережі зможуть отримувати достатню кількість інформації. Це підвищує їх здатність знаходити важливі ознаки, вирішує проблеми затухання та вибуху градієнта.

2.9 Трансферне навчання

Трансферне навчання – це метод машинного навчання, який передбачає використання попередньо навченої моделі та доналаштування (fine-tuning) її. Тобто використання моделі, яка навчилась на великому наборі даних ImageNet, що містить 1.4 мільйона зображень із 1000 класів. Потім виконується повторне донавчання конкретного завдання з новою вибіркою, яка містить менше даних.

Використовуючи даний підхід навчання покращує результат роботи у завданнях із малим набором даних.

Алгоритм застосування трансферного навчання:

1) завантажити попередньо навчену модель;
2) заморозити ваги базових шарів (або частину із них);
3) додати додаткові шари для класифікації пухлини;
4) створити нову модель на основі попередньо навченої моделі та нових додаткових шарів.

2.10 Моделі класифікації

Для задач у сфері медичної діагностики, найбільш поширені згорткові нейронні мережі VGG-16, ResNet-50 та DenseNet-201. Кожна з них має свої особливості та недоліки. У наступних розділах буде більш детально розглянута кожна мережа.

2.10.1 VGG-16

VGG-16 – глибока згорткова нейронна мережа, запропонована в 2014 році K. Simonyan та A. Zisserman у роботі «Very Deep Convolutional Networks for Large - Scale Image Recognition»[9]. Мережа складається з 16 шарів: 13 згорткових та 3 повнозв'язних шарів. Архітектура мережі VGG-16 показана на рисунку 2.9.

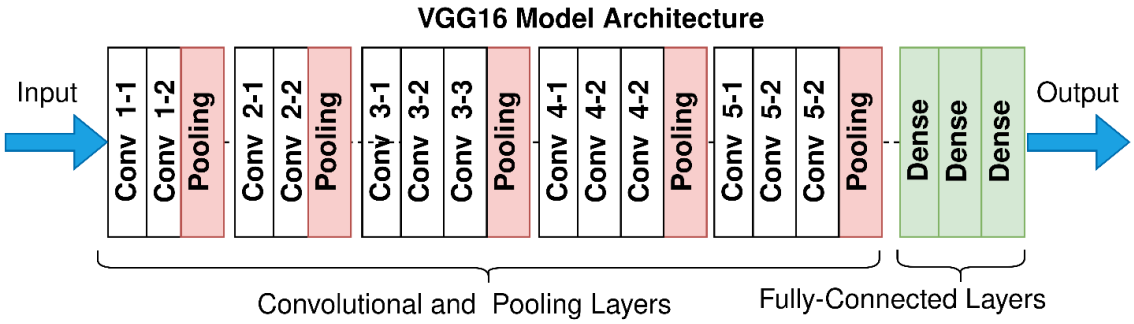


Рисунок 2.9 – Архітектура мережі VGG-16

У мережі використовують малі згорткові фільтри 3 × 3 із малою кількістю фільтрів на кожному шарі. Це дозволяє зменшити кількість параметрів та зберегти точність класифікації. Усі згорткові шари містять нелінійну функцію активації ReLU. Використання 3 повнозв'язних шарів потребує великої обчислювальної потужності, що зменшує швидкість навчання.

2.10.2 ResNet-50

ResNet-50 (Residual Network 50) — це згорткова нейронна мережа, представлена в 2015 році на конференції Computer Vision and Pattern Recognition компанією Microsoft Research Asia[8].

Мережа складається з 50 шарів, що дозволяє тренувати нейромережі значно глибше ніж, наприклад, VGG-16. Але зі збільшенням глибини сигнал, необхідний для зміни ваг, який виходить від кінця мережі, стає дуже слабким.

Для уникнення цієї проблеми в ResNet-50 використовують залишкові блоки між згортковими шарами. Кожен залишковий блок являє собою невелику нейронну мережу з обхідним зв'язком. Архітектура мережі ResNet-50 показана на рисунку 2.10.

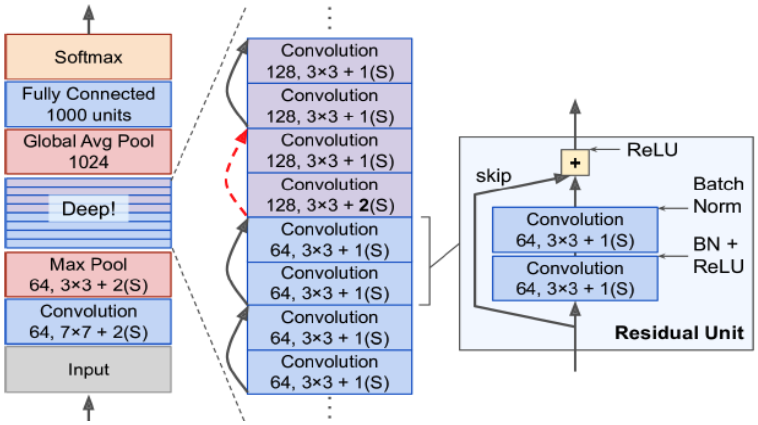


Рисунок 2.10 – Архітектура ResNet-50

2.10.3 DenseNet-201

DenseNet-201 – це згорткова нейронна мережа, яка була запропонована в 2017 році командою дослідників з університету Квебеку в Монреалі[10].

Структура DenseNet-201, яка містить 201 шар. Архітектура моделі DenseNet-201 зображена на рисунку 2.11.

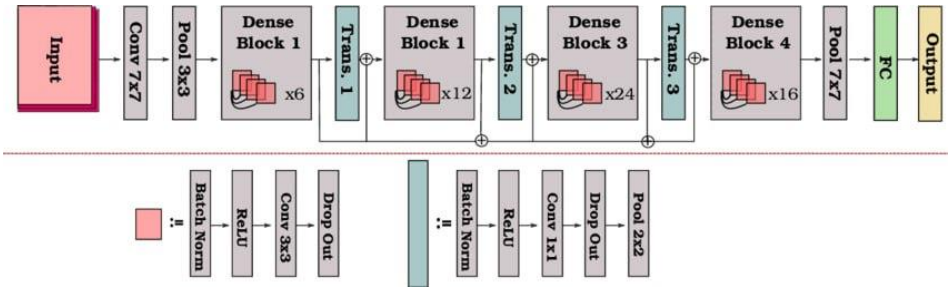


Рисунок 2.11 – Архітектура DenseNet-201

Особливість архітектури DenseNet-201 полягає в тому, що кожен шар зв'язаний з усіма попередніми шарами у зв'язному блоці (dense block). Оскільки кожен шар отримує всі попередні шари як вхідні, це дозволяє отримати більш різноманітні ознаки.

Переваги DenseNet-201:

1) вирішення проблеми затухання градієнта;
2) зменшення кількості параметрів та покращення їх використання, що допомагає уникнути перенавчання.

Ці переваги збільшують ефективність у роботі із зображеннями високої роздільної здатності.

2.10.4 Запропонована модель згорткової нейронної мережі

Запропонована модель, яка містить 4 згорткові шари, що допомагають виявити особливості зображення. Розмір вхідного зображення 254×254, що має кольорові канали RGB. Перший згортковий шар містить 16 фільтрів із розміром фільтра 3×3. Невелика кількість фільтрів у першому шарі необхідна для виявлення кутів ліній та країв. Кожен наступний шар має 32, 64 та 128 фільтра відповідно з розміром фільтра 3×3. До кожного шару додано шар пулінгу максимум із фільтром 2×2 для того, щоб отримати максимальний підсумок із зображення, зменшити розмір зображення та підвищити швидкість обчислення. Також кожний згортковий шар містить шар Batch Normalization, що стандартизує дані та допомагає зменшити шум. Використано функцію активації ReLU, що допомагає зробити модель не лінійною. Після згорткових шарів слідує розгорнутий шар, що перетворює 3D-тензори згорткових шарів в 1D-тензор, необхідний для використання повнозв’язних шарів.

Модель містить два повнозв’язних шари. Перший з них має 256 нейронів та застосовує функцію активації ReLU. Щоб уникнути перенавчання за допомогою шару Dropout виконується випадкове відключення 30% нейронів шару. Другий повнозв’язний шар містить один нейрон та застосовує функцію активації Sigmoid для бінарної класифікації вхідного зображення МРТ. Структура запропонованої моделі показана на рисунку 2.12.

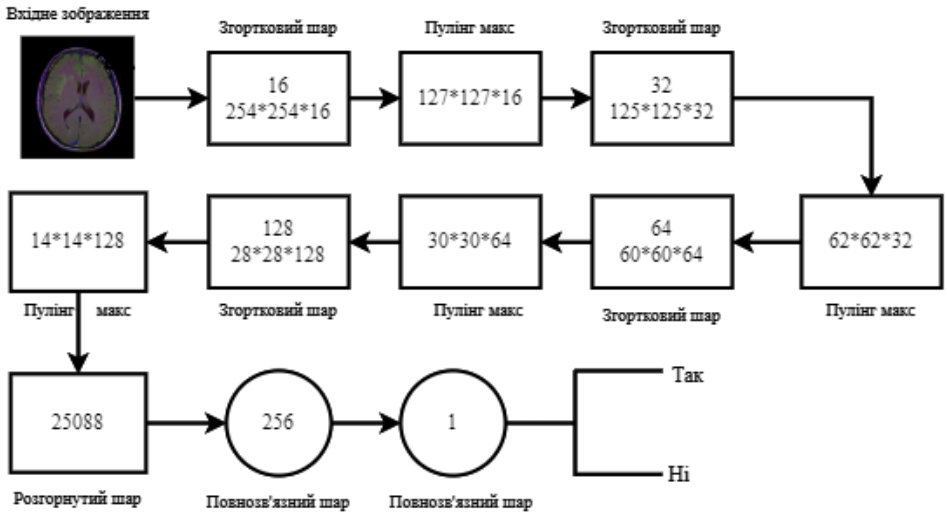


Рисунок 2.12 – Структура запропонованої моделі

2.11 Моделі сегментації

U-Net – це повна згорткова нейронна мережа, яка була запропонована в 2015 році для обробки медичних зображень. Звичайні згорткові нейронні мережі, які були розглянуті, зосереджуються на класифікації зображень.

Але для задач медицини необхідно не лише розпізнати наявність захворювання, а й локалізувати область аномалії. Саме тому було розроблено архітектуру U-Net. Вона складається з двох частин – енкодер (encoder) та декодер (decoder), які з’єднуються між собою. Енкодер це звужуючий шлях, тоді як декодер навпаки. Енкодер має структуру згорткової нейронної мережі, де зображення послідовно проходить через кілька шарів згортки та пулінгу. Це допомагає моделі виділити більш абстрактні та загальні риси зображення.

Декодер складається із транспонованих згорткових шарів (transpose convolution), кожен із яких збільшує зображення вдвічі, та об’єднувальних шарів (up-sampling). Кожен транспонований згортковий шар об’єднується з відповідним шаром енкодера за допомогою залишкового з’єднання.

Для цього спочатку відбувається копіювання (copy) відповідних вихідних даних з енкодера та звуження (crop) відповідних декодувальних даних. Використання залишкового з’єднання дозволяє передавати важливі ознаки від більш поверхневих шарів енкодера до глибоких шарів декодера. Це дозволяє збільшити точність сегментації. Bridge Connection використовується для з’єднання енкодера та декодера. Архітектура мережі U-Net зображена на рисунку 2.13.

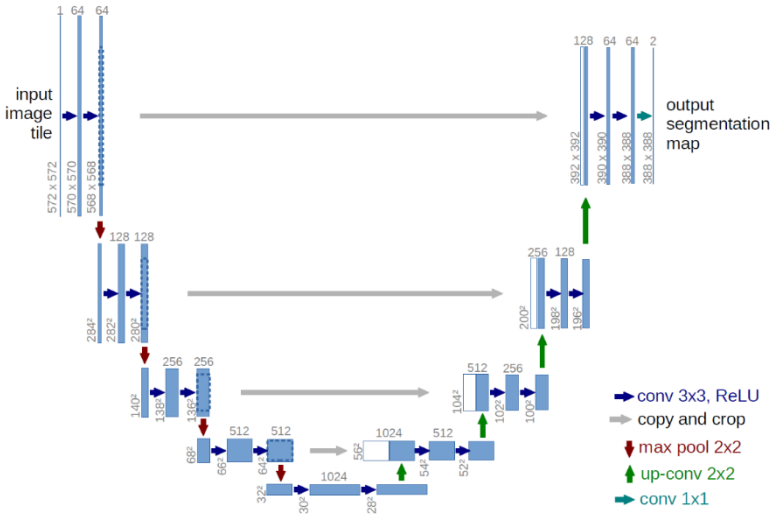


Рисунок 2.13 – Архітектура U-Net

U-Net може мати складнощі з сегментацією дрібних об'єктів на зображенні, так як модель має тенденцію до зведення малих об'єктів до шуму. Тому для сегментації пухлин головного мозку застосовано модифіковані архітектури ResUnet, ResUnet++, Attention U-Net.

2.11.1 ResUnet

ResUnet (Residual U-Net) є модифікацією архітектури U-Net, яка була запропонована в 2015 році. Мережа вважається однією з найбільш ефективних архітектур для сегментації медичних зображень, зокрема, для сегментації пухлин головного мозку.

ResUnet поєднує в собі дві архітектури: ResNet та U-Net. Архітектура ResUnet використовує залишкові блоки, що дозволяє зберігати та передавати більше інформації між різними шарами мережі, що зменшує ризик перенавчання та допомагає збільшити точність моделі.

Активація ReLU використовується на кожному рівні енкодера та декодера, тоді як вихідний сигнал декодера використовує функцію активації Sigmoid та проходить через згортковий шар 1×1. В енкодері, після кожного згорткового шару, використовується залишковий блок, що дозволяє зберігати корисну інформацію про зображення та запобігає затуханню або вибуху градієнта. У декодері також використовуються залишкові блоки, але зі зворотними згортковими шарами, щоб відтворити вихідне зображення з більш високою роздільною здатністю. Архітектура мережі ResUnet зображена на рисунку 2.14.

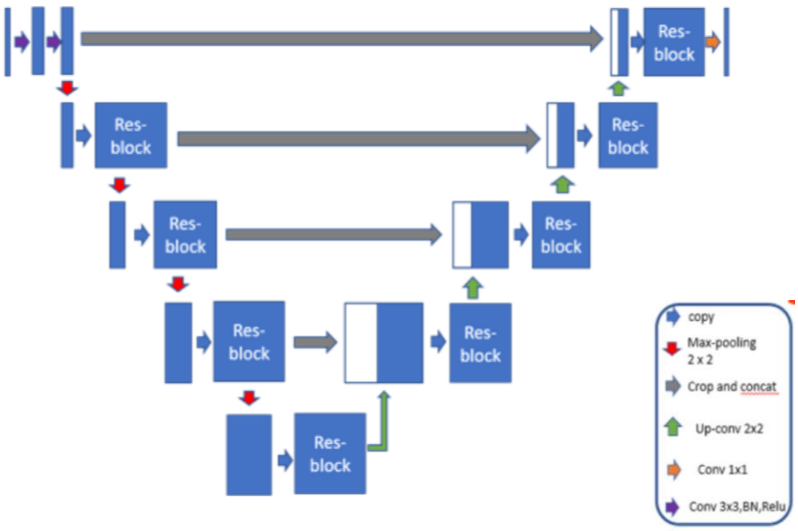


Рисунок 2.14– Архітектура ResUnet

2.11.2 ResUnet++

Архітектура ResUnet++ це модифікація мережі ResUnet, яка була запропонована в 2018 році. На рівні енкодера додано блоки стиснення та збудження (squeeze-and-excitation blocks).

На рівні декодера додано блоки уваги (attention blocks) та Atrous Spatial Pyramidal Pooling (ASPP) у блоці фінального шару і блоку, що поєднує енкодер та декодер[11]. Архітектура зображена на рисунку 2.15.

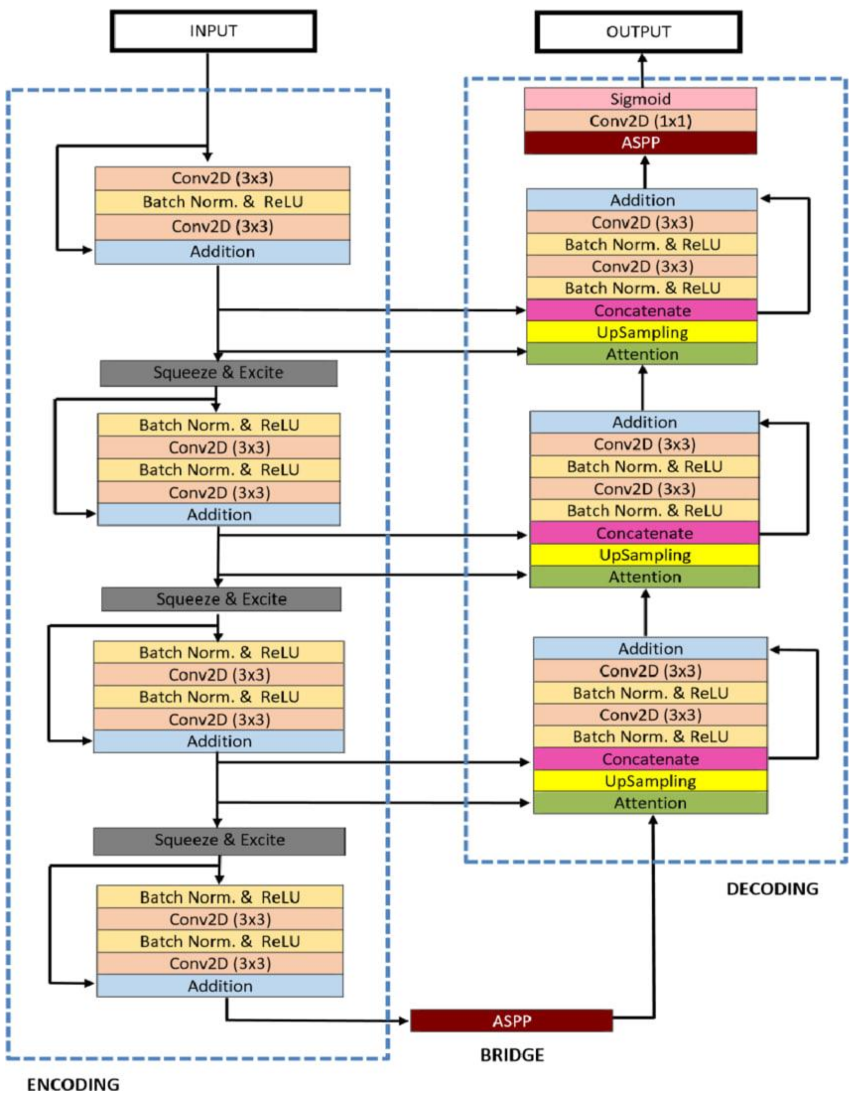


Рисунок 2.15 – Архітектура ResUnet++

Блок стиснення та збудження (Squeeze-and-excitation block)[12] – блок, який використовується у глибоких нейронних мережах для покращення їхньої

точності. Він допомагає зосередитися на найбільш важливих частин даних, з якими мережа працює.

Блок складається з двох етапів: squeeze та excitation. На етапі squeeze середнє значення по каналах обчислюється для кожного вхідного зображення. На етапі excitation ці значення проходять через повнозв'язний шар, який відповідає за виявлення важливих функцій зображення, що забезпечують високу точність в роботі з даними.

Механізм уваги (Attention mechanism) використовується в задачах семантичної сегментації, таких як попіксельне прогнозування[13]. Використовуючи механізм уваги, модель приділяє більше уваги областям зображення, які можуть містити пухлину. Це зменшує кількість надлишкових функцій та дозволяє отримати більш точну маску пухлини на зображенні.

Механізм уваги реалізований за допомогою воріт уваги(attention gate) на кожному залишковому блоці[14]. Схема механізму зображена на рисунку 2.16.

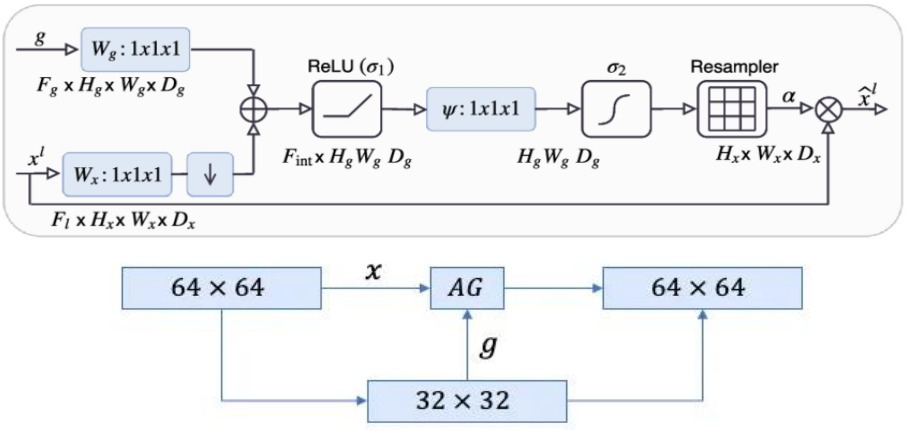


Рисунок 2.16 – Схема механізму уваги

Основна перевага механізмів уваги полягає в тому, що вони прості, можуть застосовуватися до будь-якого розміру вхідних даних, підвищують якість функцій, що покращує результати.

2.11.3 Attention U-Net

Attention U-Net – це архітектура, яка була представлена у 2018 році. Мережа поєднує U-Net та механізм уваги[14]. Даний механізм застосовано і в архітектурі ResUnet++, але в Attention U-Net механізм уваги використовується лише на етапі залишкового з'єднання. Це дозволяє мережі зосередитись на важливих деталях зображення та покращити точність сегментації ніж при використанні базової архітектури U-Net. Спираючись на цю різницю, Attention U-Net має меншу кількість параметрів та швидше навчається.

У багатьох медичних дослідженнях, де необхідно визначити межі аномалії на зображенні, Attention U-Net показала високу точність та ефективність. Архітектура мережі зображена на рисунку 2.17.

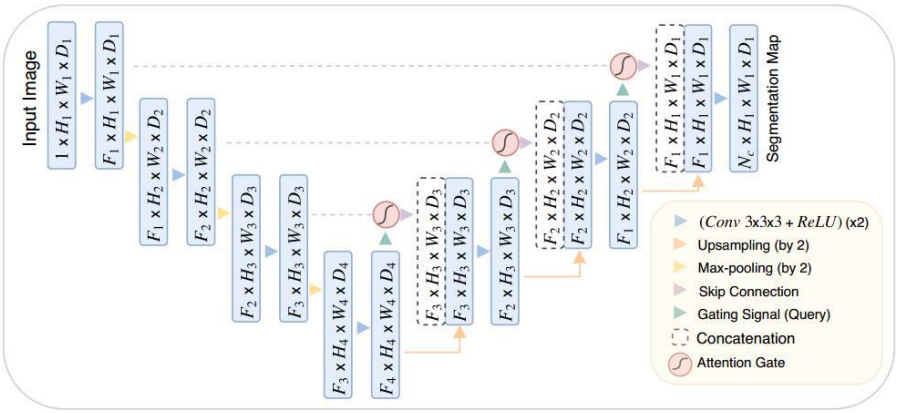


Рисунок 2.17 – Архітектура Attention U-Net

2.12 Висновки до розділу 2

У розділі 2 розглянуто застосування комп'ютерного зору у різних сферах із використанням згорткових нейронних мереж. Детально розглянуті основні шари ЗНМ, методи регуляризації для уникнення перенавчання, залишкове навчання для моделей із великою кількістю шарів. Також розглянуто трансферне навчання, яке використовується для побудови нових моделей штучних нейронних мереж.

Для класифікації знімків МРТ обрані класичні моделі ResNet-50, VGG-16 та DenseNet, які найчастіше використовуються для обробки медичних зображень. Також запропонована ЗНМ для бінарної класифікації знімків МРТ.

Для сегментації пухлин обрані моделі ResUnet, ResUnet++ та Attention U-Net. Крім цього, проведено опис функцій втрат та метрик для оцінювання якості роботи моделей класифікації та сегментації.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору платформи та мови програмування

Одним із важливих етапів написання дипломної роботи є вибір мови програмування. Python є однією з найбільш популярних мов програмування завдяки своїй простоті, гнучкості, швидкості та незалежності від платформи. Також Python має багато бібліотек для машинного навчання та глибокого навчання. Отже, для задачі з класифікацією та сегментацією пухлин головного мозку обрано мову програмування Python.

Основні бібліотеки для вирішення поставленого завдання:

- TensorFlow,
- Keras,
- Torch,
- Scikit-learn,
- Matplotlib,
- NumPy,
- Pandas,
- Cv2,
- Tkinter.

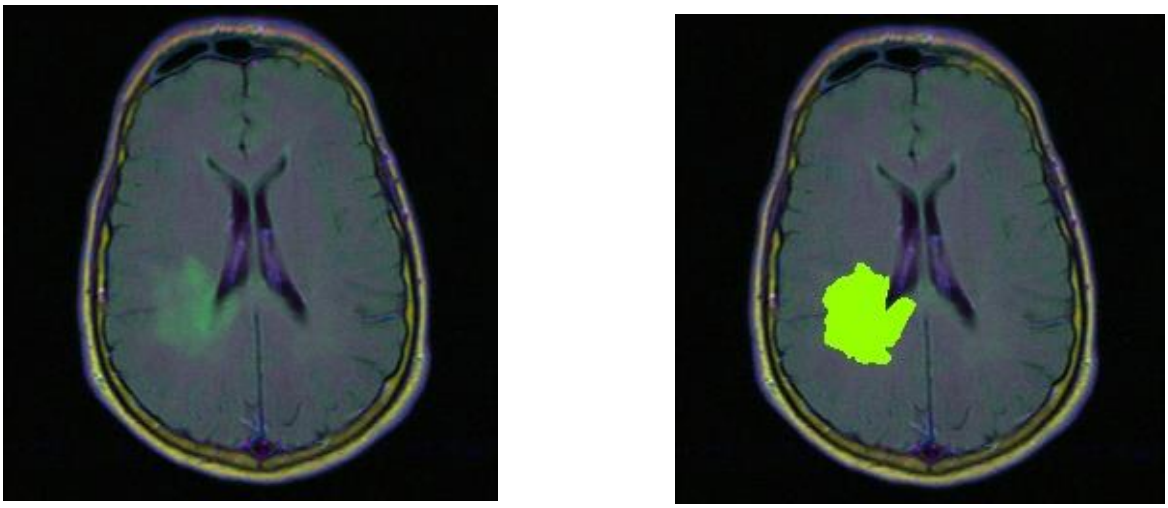
У якості платформи для вирішення завдання використовується Google Colab. Вона розміщена на хості служби Jupyter для ноутбуків, яка надає безкоштовний доступ до обчислювальних ресурсів. Colab використовує обчислювальну потужність Google замість локального комп’ютера. Тому для навчання моделей глибоких нейронних мереж використання Google Colab є одним з найкращих варіантів.

3.2 Опис вхідних даних

У дослідженні використано набір зображень МРТ головного мозку та зображень області, що відповідають пухлинам, які виділені окремим кольором. Таке зображення має назву маска. Використання масок дозволяє точно визначити місце розташування пухлин та є важливим елементом для навчання моделей з метою автоматичної класифікації та локалізації пухлин на знімках МРТ.

Вибірку було обрано на платформі Kaggle[16], яка складається із 3929 знімків МРТ, 65% з яких містить пухлину.

Зображення знімків МРТ представлене в форматі RGB(Red Green Blue), тобто кожний піксель має 3 значення, кожне з яких може набувати значення від 0 до 255. Розмір зображень – 256×256. Приклад МРТ знімка та МРТ знімка із маскою зображено на рисунку 3.1.



(a) (b)

Рисунок 3.1 (a) – знімок МРТ мозку,

(b) – знімок МРТ мозку з маскою

3.3 Алгоритм роботи програмного продукту

Користувач завантажує знімок МРТ головного мозку людини. Спочатку відбувається попередня обробка зображення. Наступний етап це класифікація. Дана програма класифікує знімки на 2 категорії: є пухлина або пухлина відсутня. Якщо пухлини не знайдено користувач отримує повідомлення про це та програма завершує роботу. Інакше відбувається процес сегментації даної пухлини, виділення меж і на екрані виводиться зображення знімка МРТ із виділеною пухлиною. Блок-схема програми зображена на рисунку 3.2.

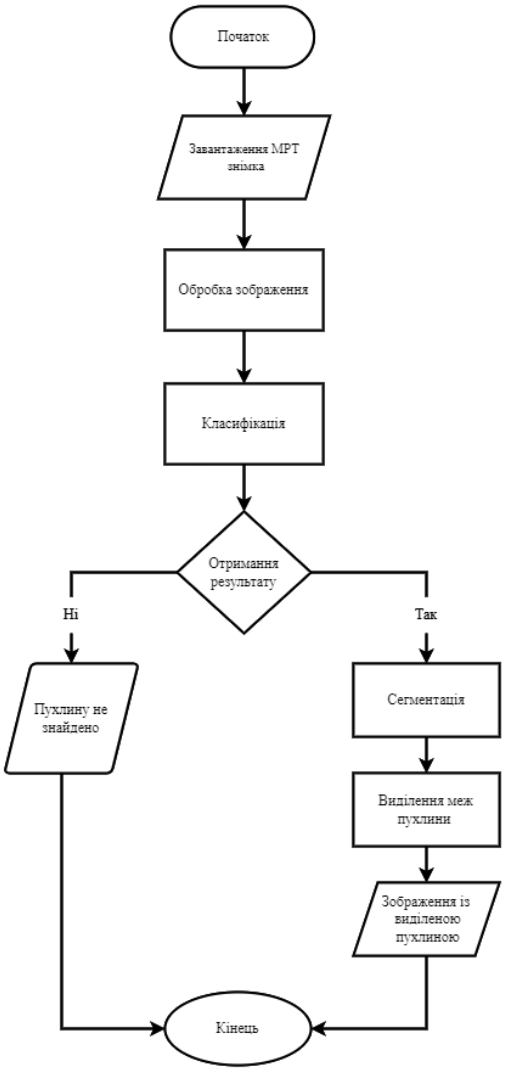


Рисунок 3.2 – Блок-схема програмного продукту

3.4 Навчання моделей класифікації

Для проведення експерименту датасет розбито на навчальну, валідаційну та тестову вибірки. Спочатку, датасет поділено на навчальну та тестову вибірки у співвідношенні 85% : 15% за допомогою функції “train_test_split” із бібліотеки Scikit-learn. Тестова вибірка додатково поділено на валідаційну та тестову вибірки у співвідношенні 50% : 50% за допомогою тієї ж функції. Результат розподілу показано на рисунку 3.3.

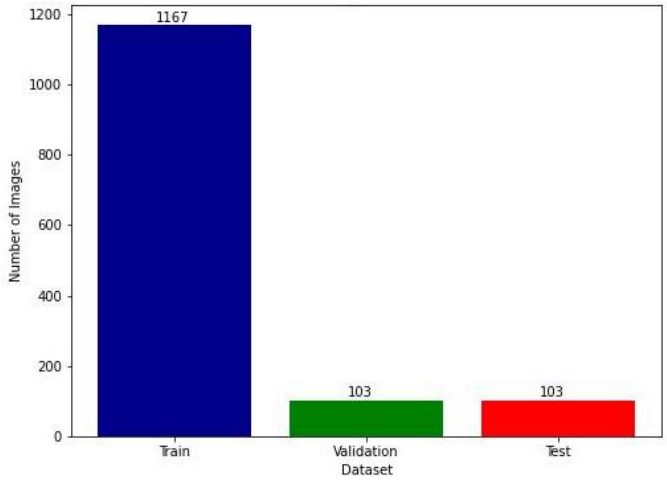


Рисунок 3.3 – Розподіл кількості елементів у кожній вибірці

Для генерації МРТ зображень та міток використано DataGenerator. Це інструмент у бібліотеці Keras для автоматичного завантаження та попередньої обробки даних для тренування моделей глибокого навчання. Даний генератор дозволяє завантажувати дані по частинам (batch), що покращує використання пам’яті.

Під час тренування застосована техніка Early Stopping. Це техніка дозволяє автоматично завершувати процес навчання, якщо метрика точності або функція втрат не покращується протягом певної кількості епох.

Навчальна вибірка містить лише 2860 знімків МРТ, тому застосовано трансферне навчання. Спочатку моделі ResNet-50, VGG-16 та DenseNet-50 навчалися на великому наборі даних зображень ImageNet, що містить понад 1.2 мільйона зображень з 1000 класів.

В якості оптимізатора використано Adam, функція втрат Binary Cross-Entropy та метрика accuracy.

3.5 Аналіз результатів моделей класифікації

На рисунку 3.4 показано графіки точності для моделей ResNet-50, VGG-16, DenseNet та для запропонованої моделі під час навчання.

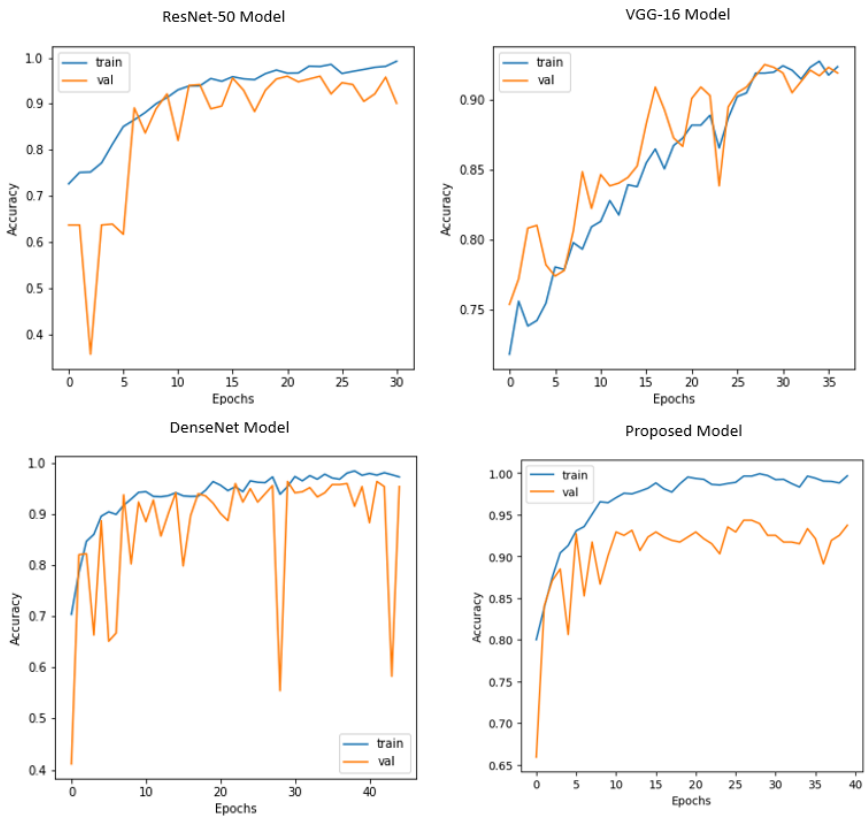


Рисунок 3.4 – Графіки точності моделей класифікації

Навчені моделі оцінено на тестуючих даних. У таблиці 3.1 TP та TN показують кількість правильно класифікованих знімків з пухлиною головного мозку та без пухлини.

FN – помилка першого роду, яка виникає, якщо модель не виявляє пухлини, коли вона насправді є на знімку. FN може мати серйозні наслідки, так як не виявлення пухлини може призвести до неправильного лікування пацієнта. Тому, важливо забезпечити якомога менше помилок такого типу.

FP – помилка другого роду, яка виникає, якщо модель хибно діагностує наявність пухлини.

Таблиця 3.1 – Результати навчання моделей класифікації

Модель	TP	TN	FP	FN
Запропонована ЗНМ	185	375	18	12
ResNet-50	181	391	2	16
VGG-16	189	379	14	8
DenseNet-50	192	389	4	5

У таблиці 3.2 показано значення метрик для кожної з моделей.

Таблиця 3.2 – Метрики для моделей класифікації

Модель	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)
Запропонована ЗНМ	94	91	93	92
ResNet-50	97	98	92	95
VGG-16	96	93	95	94
DenseNet-201	98	98	97	98

Модель DenseNet-201 має найбільші значення у кожній метриці якості.

Запропонована модель має менші показники метрик, оскільки архітектура моделі є простішою, містить меншу кількість параметрів та кількість згорткових шарів. Модель краще знаходить знімки МРТ із пухлиною, ніж модель ResNet-50. Модель ResNet-50 краще розпізнає знімки МРТ, які не містять пухлину, тоді як модель VGG-16 – навпаки.

Метрика recall (повнота) є важливою метрикою для задачі класифікації знімків МРТ, оскільки вона показує, чи правильно модель виявляє всі важливі ознаки хвороби на знімку. Якщо модель не здатна виявити ці ознаки, це може призвести до неправильної діагностики та лікування. Найбільше значення recall у моделі класифікації DenseNet-201.

3.6. Навчання моделей сегментації

Для проведення експерименту датасет поділено на навчальну, валідаційну та тестову вибірки. Результат розподілу показано на рисунку 3.5.

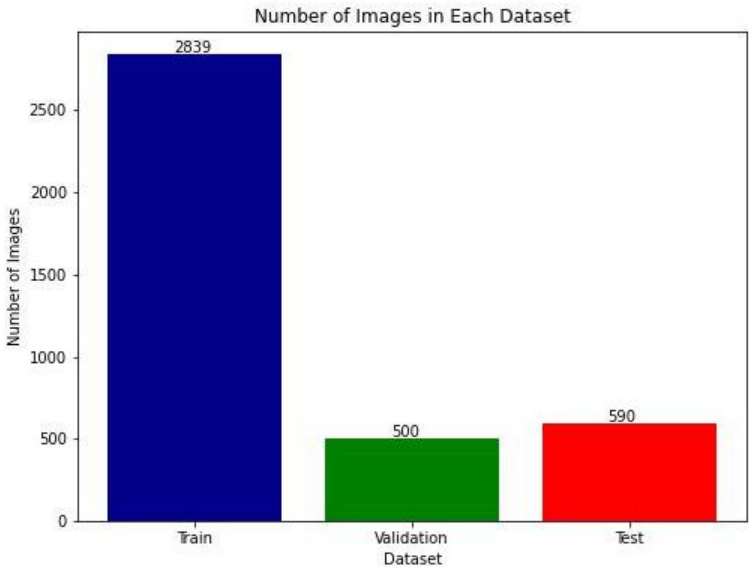


Рисунок 3.5 – Розподіл кількості елементів у кожній вибірці

Кількість епох навчання – 60. В якості оптимізатора використано Adam. Tversky index, Dice index та Jaccard index обрано в якості метрик точності. Функція втрат – Focal Smooth Tversky loss.

Під час навчання моделей застосовано техніку Early Stopping для уникнення перенавчання.

3.7 Аналіз результатів отриманих моделей сегментації

У результаті навчання отримано моделі ResUnet, ResUnet++ та Attention U-Net. На тестуючих даних розраховані метрики Tversky, Dice та Jaccard для заданих моделей сегментації, як показано в таблиці 3.3.

Таблиця 3.3 – Результат моделей сегментації

Модель	Tversky (%)	Dice (%)	Jaccard (%)
ResUnet	95	94	89
ResUnet++	95	93	87
Attention U-Net	94	93	87

Всі три навчені моделі демонструють високі показники точності. Однак, метрика Jaccard, що вимірює співставлення передбаченої множити пікселів зі справжньою множиною пікселів, які відповідають об’єкту на зображенні, для моделей ResNet++ та Attention U-Net трохи менша, чим для ResNet. Високе значення метрики Jaccard вказує на високу якість сегментації.

Результати застосування моделей сегментації зображено на рисунку 3.6.

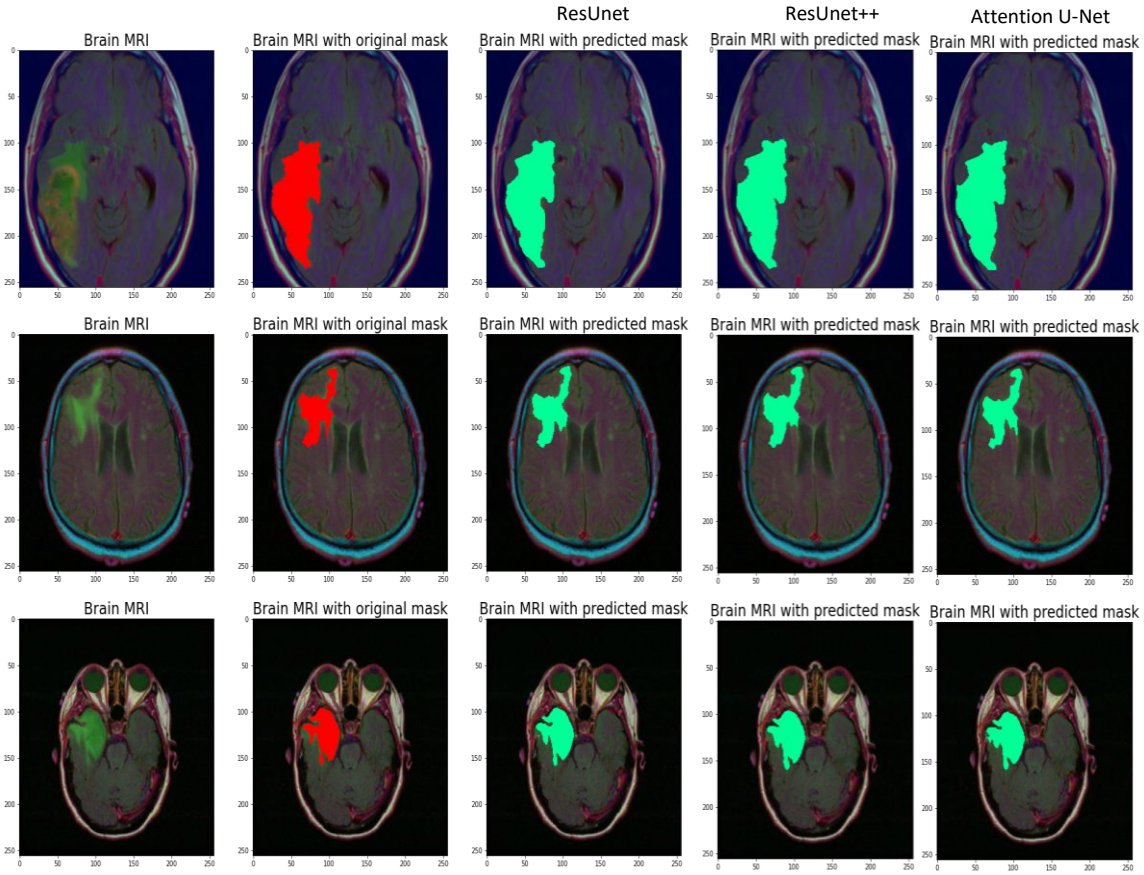


Рисунок 3.6 – Порівняння результатів сегментації

3.8 Результат роботи програми

Під час запуску програми користувач потрапляє на головну сторінку. Щоб завантажити зображення знімка МРТ, необхідно натиснути на кнопку “Load brain tumor”. Щоб дослідити знімок МРТ, потрібно натиснути на кнопку “Find brain tumor”. Результати роботи програми, якщо пухлину знайдено, зображено на рисунку 3.7.

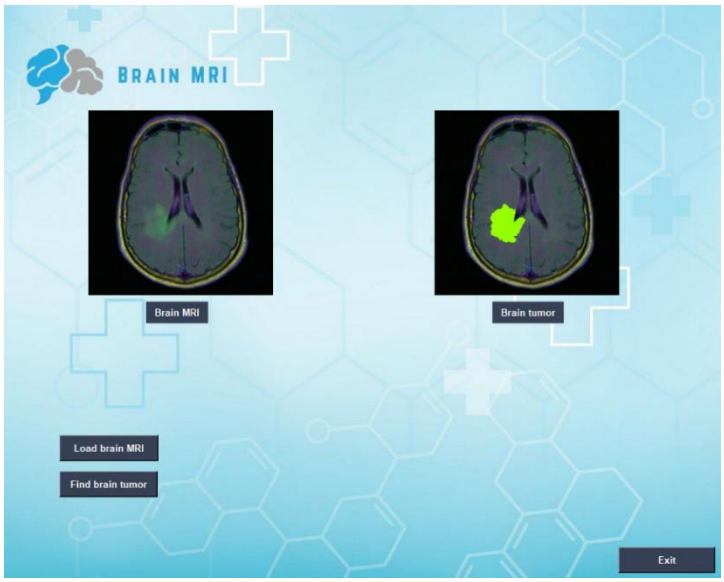


Рисунок 3.7 – Результат сегментації пухлини

Якщо на знімку МРТ пухлини не знайдено, на екрані з’явиться повідомлення про це. Результат роботи програми зображено на рисунку 3.8.

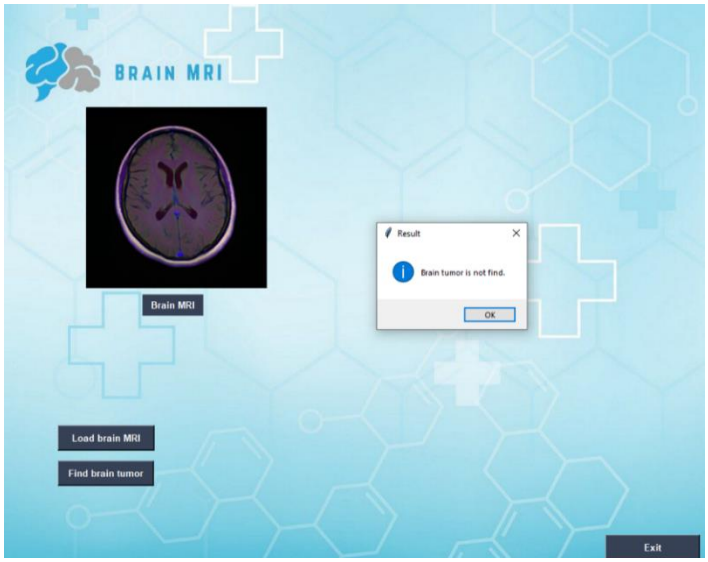


Рисунок 3.8 – Результат програми, якщо пухлину не знайдено

3.9 Висновки до розділу 3

У розділі 3 обґрунтовано вибір мови програмування Python, середовище для навчання моделей та створення програмного продукту. Наведено блок-схему програми, яку можна використовувати для аналізу знімків МРТ головного мозку людини. Також описано інтерфейс програми та приклади її використання .

При розробці програмного продукту навчено моделі ResNet-50, VGG-16 та DenseNet-201 із використанням трансферного навчання. Навчені моделі перевірено на тестовій вибірці. Найкращі результати точності, а саме 98%, показала модель DenseNet-201. Модель не тільки має високу точність класифікації, а й досягає мінімального значення помилки першого роду (FN), що є дуже важливим для правильного та своєчасного лікування. Результат точності запропонованої ЗНМ на тестовій вибірці складає 94%, що є достатньо гарним результатом.

Для локалізації пухлини на знімку МРТ навчено моделі ResUnet, ResUnet++ та Attention U-Net. Модель ResUnet має найвищі значення метрик при перевірці навченої моделі на тестовій вибірці.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі проектування

У даному розділі здійснюється оцінка основних характеристик, що стосуються розробки програмного продукту (ПП) для вирішення задачі класифікації знімків МРТ та, у випадку знаходження новоутворення, його сегментація.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість програмного продукту незалежно від організаційної структури компанії. Даний аналіз допомагає виявити резерви зниження витрат за рахунок ефективних варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його створення.

Фактичний аналіз це аналіз функцій програмного продукту, призначених для обробки, аналізу та прогнозування даних про наявність новоутворень на знімках МРТ та в разі виявлення – їх сегментація, тому що кожен компонент має задовольняти вимоги рішень для реалізації та проектування в цілому.

Основні технічні вимоги до програмного продукту:

- сумісність зі стандартними компонентами персональних комп'ютерів;
- зрозумілість та зручність інтерфейсу для користувача;
- швидке опрацювання даних та миттєвий доступ до інформації;
- можливість легко масштабувати та підтримувати програмний продукту;
- мінімізація витрат на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який вирішує задачу класифікації знімків МРТ на наявність новоутворення та в разі виявлення – його сегментація. Беручи за основу цю функцію, можна виділити наступні функції програмного продукту:

- F_1 – вибір мови програмування;
- F_2 – вибір фреймворку машинного навчання;
- F_3 – вибір середовища розробки.

Кожна з визначених функцій має декілька варіантів реалізації.

Функція F_1:

a) Python;

b) C++.

Функція F_2:

a) Tensorflow;

b) PyTorch.

Функція F_3:

a) Google Colab;

b) Visual Studio.

Варіанти реалізації основних функцій наведено у морфологічній карті, яка зображена на рисунку 4.1.

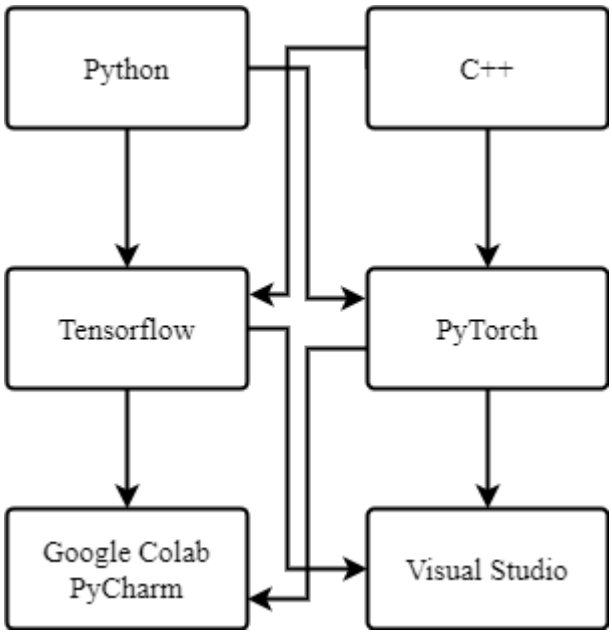


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів заданих функцій. На основі карти побудовано позитивно-негативну матрицю варіантів основних функцій, яку зображено у таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Простий для розуміння синтаксис та наявність бібліотек високого рівня.	Повільна обробка великого обсягу даних.
	B	Код швидко виконується.	Розробка програми займає багато часу.
F_2	A	Висока ефективність під час обробки даних.	Великий обсяг використання пам’яті.
	B	Простий для розуміння синтаксис.	Недостатня підтримка та розширюваність в порівнянні з Tensorflow.
F_3	A	Виконання коду в хмарних середовищах.	Відсутня можливість роботи без інтернету.
	B	Можливість інтеграції з іншими інструментами роботи.	Підтримується лише одна мова програмування одночасно.

На основі розглянутої матриці при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим задачам.

Функція F_1:

перевагу віддаємо варіанту А, мова програмування Python.

Функція F_2:

обидва варіанти можна використовувати при розробці.

Функція F_3:

Перевагу віддаємо варіанту А, виконання роботи в Google Colab.

Отже, будемо розглядати два варіанти реалізації програми:

Варіант I: $F_1a - F_2a - F_3a$;

Варіант II: $F_1a - F_2b - F_3a$.

Для оцінювання якості розглянутих функцій обрано систему параметрів, яка описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, які були розглянуті, визначаються основні параметри вибору, що будуть використані для розрахунку коефіцієнта технічного рівня. Для характеристики програмного продукту використовуються параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для збереження даних;
- X3 – час попередньої обробки даних;
- X4 – потенційний об’єм програмного коду.

Кращі, середні та гірші значення параметрів вибираються на основі вимог замовника та умов, що характеризують експлуатацію програмного продукту як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру		Значення параметрів		
				гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	10000	15000	20000	
Об’єм пам’яті для збереження даних	X2	Мб	600	300	128	
Час попередньої обробки даних	X3	Мс	9	5	3	
Потенційний об’єм програмного коду	X4	Кількість рядків коду	3000	2500	1500	

За даними таблиці 4.2 побудовано графіки параметрів: рисунок 4.2 – рисунок 4.4.

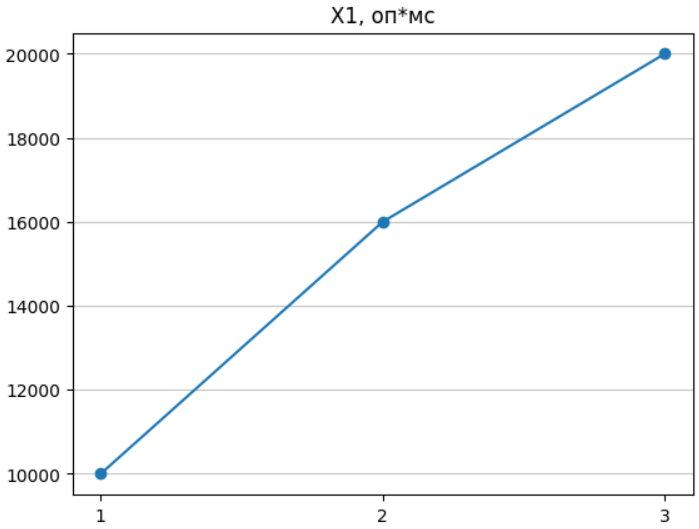


Рисунок 4.2 – X1, швидкодія мови програмування

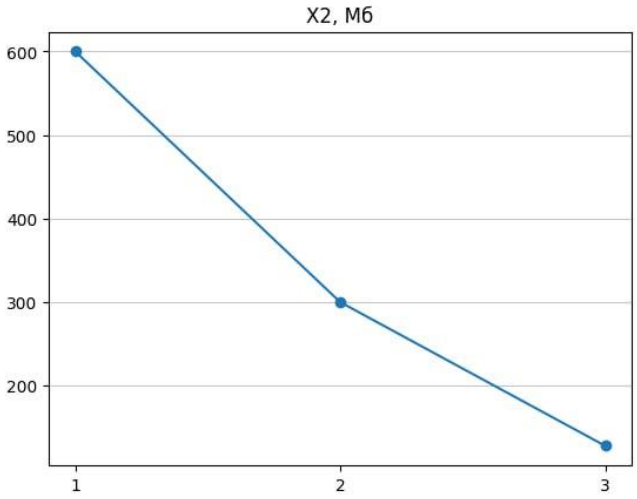


Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

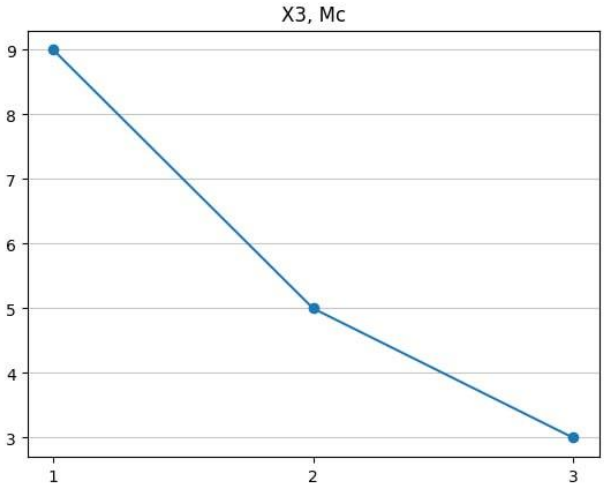


Рисунок 4.3 – X3, час попередньої обробки даних

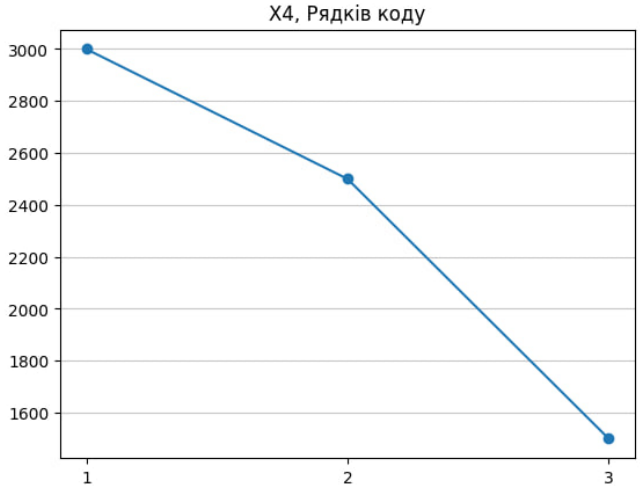


Рисунок 4.4 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої мети – розробка програмного продукту, яка дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування й обчислення прогнозованих значень.

Експертна комісія складається із 7 людей. Значимість кожного параметра визначається методом попарного порівняння. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірка придатності експертних оцінок для подальшого використання;
- обробка результатів та визначення коефіцієнту значимості.

За найбільший ранг приймається значення 4, за найменший – 1. Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
X1	Оп/мс	4	5	2	4	3	4	5	27	2.5	6.25
X2	Мб	2	1	3	2	2	1	2	13	-11.5	132.25
X3	Мс	5	4	5	5	4	5	3	31	6.5	42.25
X4	Кількість рядків коду	3	4	4	3	5	4	4	27	2.5	6.25
Разом		14	14	14	14	14	14	14	98		187

Для перевірки степені достовірності експертних оцінок визначимо наступні параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98 \quad (4.1)$$

де N- число експертів, n – кількість параметрів.

- середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24.5 \quad (4.2)$$

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (4.3)$$

- загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 \quad (4.4)$$

Коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = 0.76 \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодження перевищує нормативний, який дорівнює 0.67.

Використовуючи результати ранжування, проведемо попарне порівняння всіх параметрів. Результати занесено в таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	>	>	>	>	>	1.5
X1 і X3	<	>	<	<	<	<	>	<	0.5
X1 і X4	>	>	<	>	<	=	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	>	=	>	>	<	>	<	>	1.5

Числове значення, що визначає ступінь переваги і-го параметра над j-тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складено матрицю $A = \|a_{ij}\|$.

Для кожного параметра розрахунок вагомості $К_{bi}$:

$$К_{bi} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів, поки наступні значення не будуть значно відрізнятись від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за формулами (4.9) та (4.10):

$$К_{bi} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}} \quad (4.9)$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_i \quad (4.10)$$

На основі даних таблиці 4.5 маємо, що різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$К_{bi}$	b_i^1	K_{bi}^1	b_i^2	K_{bi}^2
X1	1.0	1.5	0.5	1.5	4.5	0.36	17.75	0.25	73.38	0.25
X2	0.5	1.0	1.5	0.5	3.5	0.28	15.75	0.22	65.63	0.23
X3	1.5	1.5	1.0	1.5	5.5	0.44	22.75	0.33	93.6	0.32
X4	0.5	1.5	1.5	1.0	3.5	0.28	13.75	0.2	57.63	0.2
Всього:					12.5	1	70	1	290.25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Коефіцієнт технічного рівня для кожного варіанту реалізації програмного продукту розраховується за формулою (4.11).

$$К_К(j) = \sum_{i=1}^{n} К_{bi,j} В_{i,j} \quad (4.11)$$

де

n – кількість параметрів;

$К_{bi}$ – коефіцієнт вагомості і-го параметра;

$В_i$ – оцінка і-го параметра в балах.

Розрахунок показників рівня якості варіантів реалізації основних функцій зображено в таблиці 4.6.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій

Основні функції	Варіант реалізації функції	Пара-метри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	A	X1	10000	6	0.22	1.15
F_2	A	X2	128	5	0.33	1.45
	B	X2	300	2	0.3	0.8
F_3	A	X4	1000	8	0.15	1.36

За даними таблиці 4.6 за формулою (4.12) визначаємо рівень якості:

$$\mathrm{K_K} = \mathrm{K_{ТУ}}[\mathrm{F_{1k}}] + \mathrm{K_{ТУ}}[\mathrm{F_{2k}}] + \mathrm{K_{ТУ}}[\mathrm{F_{2k}}] \quad (4.12)$$

Визначаємо рівень якості кожного з варіантів:

$$\mathrm{K_{K1}} = 1.15 + 1.45 + 1.36 = 3.96$$

$$\mathrm{K_{K2}} = 1.15 + 0.8 + 1.36 = 3.31$$

Коефіцієнт технічного рівня більший для першого варіанту.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості. Всі варіанти містять два окремих завдання:

- завдання 1. Розробка проєкту програмного продукту;
- завдання 2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи В. За складністю алгоритми, які використовуються в завданні 1 належать

до групи 1, а в завданні 2 – до групи 3. Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного завдання.

Загальна трудомісткість обчислюється за формулою (4.13).

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де

T_P – трудомісткість розробки ПП;

$K_П$ – коефіцієнт на складність вхідної інформації;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи з норм часу для завдань розрахункового характеру степеню новизни А та групи складності варіанту I, трудомісткість дорівнює: $T_P = 100$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П = 1.6$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, враховуємо це за допомогою коефіцієнта $K_{СТ} = 0.8$.

Загальна трудомісткість програмування першого завдання:

$$T_1 = 100 \cdot 1.6 \cdot 0.8 = 128 \text{ людино-днів.}$$

Для другого завдання при $T_P = 30$ людино-днів, $K_П = 0.9$, $K_{СТ} = 0.8$.

$$T_2 = 30 \cdot 0.9 \cdot 0.8 = 21.6 \text{ людино-днів.}$$

Трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (128 + 19.44 + 4.8 + 19.44) \cdot 8 = 1373.44 \text{ людино-годин.}$$

$$T_{II} = (21.6 + 19.44 + 6.91 + 19.44) \cdot 8 = 1390.32 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

У розробці беруть участь два програмісти з окладом 30000 грн. та один дата аналітик з окладом 25000 грн. Визначимо середню зарплату за годину за формулою:

$$C_ч = \frac{M}{T_m \cdot t} \text{грн.} \quad (4.13)$$

де

M – місячний оклад працівників;

T_m – кількість робочих днів на тиждень;

t – кількість робочих годин у день.

$$C_ч = \frac{30000 + 30000 + 25000}{3 \cdot 21 \cdot 8} = 169 \text{ грн.}$$

Заробітна плата розраховується за формулою:

$$C_{зп} = C_ч \cdot T_i \cdot K_Д \quad (4.14)$$

де

$C_ч$ – величина погодинної оплати праці;

T_i – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Отже, для завдання I та завдання II заробітна плата:

$$\text{I. } C_{зп} = 169 \cdot 1373.44 \cdot 1.2 = 278533.63 \text{ грн.}$$

$$\text{II. } C_{зп} = 169 \cdot 1390.32 \cdot 1.2 = 281956.896 \text{ грн.}$$

Відраховування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{ВІД} = C_{зп} \cdot 0.22 = 278533.63 \cdot 0.22 = 61277.40 \text{ грн.}$$

$$\text{II. } C_{ВІД} = C_{зп} \cdot 0.22 = 281956.896 \cdot 0.22 = 62030.52 \text{ грн.}$$

Необхідно обчислити витрати на оплату однієї машино-години.

Одна ЕОМ обслуговує одного програміста з окладом 30000 грн. з коефіцієнтом зайнятості 0.2, то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0.2 = 72000 \text{ грн.} \quad (4.15)$$

З урахуванням додаткової заробітної плати:

$$С_{зп} = С_Г(1 + К_З) = 72000 \cdot (1 + 0.2) = 86400 \text{ грн.} \quad (4.16)$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{зп} \cdot 0.22 = 86400 \cdot 0.22 = 19008 \text{ грн.} \quad (4.17)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 20000 = 5750 \text{ грн.} \quad (4.18)$$

де

$К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу в користувача;

$К_А$ –річна норма амортизації;

$Ц_{ПР}$ –договірна ціна приладу.

Витрати на ремонт та профілактику:

$$С_Р = К_{ТМ} \cdot К_р \cdot Ц_{ПР} = 1.15 \cdot 0.05 \cdot 18000 = 1035 \text{ грн.} \quad (4.19)$$

де

$К_р$ –відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховується за формулою 4.20.

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_з \cdot К_В, \quad (4.20)$$

де

$Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

$t_з$ – кількість робочих годин у день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

$$Т_{ЕФ} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.9 = 1491.2 \text{ годин.}$$

Витрати на оплату електроенергії розраховуються за формулою (4.21).

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491.2 \cdot 0.3 \cdot 0.9 \cdot 4.87 = 1960 \text{ грн.} \quad (4.21)$$

де

$N_С$ – середньо-споживана потужність приладу;

$К_З$ – коефіцієнт зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуються за формулою (4.22).

$$С_Н = Ц_{ЕН} \cdot 0.67 = 12060 \text{ грн.} \quad (4.22)$$

Річні експлуатаційні витрати розраховуються за формулою (4.23).

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (4.23)$$

$$С_{ЕКС} = 86400 + 19008 + 5750 + 1035 +$$

$$+1960 + 12060 = 126213 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ задається формулою (4.24).

$$С_{М-Г} = \frac{С_{ЕКС}}{Т_{ЕФ}} = \frac{126213}{1677.6} = 75.2 \text{ грн/год.} \quad (4.24)$$

Оскільки у даному випадку всі роботи, які пов’язані з розробкою програмного продукту проводяться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанту реалізації:

$$С_М = С_{М-Г} \cdot Т \quad (4.25)$$

$$\text{I. } С_М = 75.2 \cdot 1373.44 = 103282.69 \text{ грн.}$$

$$\text{II. } С_М = 75.2 \cdot 1390.32 = 104552.06 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67 \quad (4.26)$$

$$\text{I. } С_Н = 278533.63 \cdot 0.67 = 186617.53 \text{ грн.}$$

$$\text{II. } С_Н = 281956.896 \cdot 0.67 = 188911.12 \text{ грн.}$$

Отже, вартість розробки ПП розраховується за формулою:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н \quad (4.27)$$

Для кожного варіанту:

$$\text{I. } С_{ПП} = 278533.63 + 61277.40 + 103282.69 + 186617.53 = 629711.25 \text{ грн.}$$

$$\text{II. } С_{ПП} = 281956.896 + 62030.52 + 104552.04 + 188911.12 = 637450.58 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Коефіцієнт техніко-економічного рівня визначається за формулою (4.28).

$$К_{ТЕРj} = К_{Кj}/С_{Фj} \quad (4.28)$$

$$К_{ТЕР1} = \frac{3.96}{629711.25} = 0.63 \cdot 10^{-5}$$

$$К_{ТЕР2} = \frac{3.31}{637450.58} = 0.51 \cdot 10^{-5}$$

Отже, найбільш ефективним є варіант I реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0.63 \cdot 10^{-5}$.

4.8 Висновки до розділу 4

У розділі 4 проведено функціонально-вартісний аналіз програмного продукту. Визначено та оцінено основні функції програмного продукту та параметри, що характеризують його, проведено експертне оцінювання параметрів та аналіз варіантів реалізації функцій.

Також застосовано економічний аналіз різних варіантів розробки, у якому враховано трудомісткість, витрати на заробітну плату та інші витрати. З двох альтернатив, що залишилися після першого відбору виконання програмного продукту, було визначено, що перший варіант з показником техніко-економічного рівня якості $К_{ТЕР1} = 0.63 \cdot 10^{-5}$ є оптимальним варіантом.

Параметри обраного варіанту реалізації програмного продукту:

- мова програмування – Python;
- використання бібліотеки Tensorflow для навчання нейронних мереж;

- використання стандартного середовища розробки Google Colab для створення нейронних мереж та Pycharm для реалізації інтерфейсу програми.

Заданий варіант виконання експертної системи надає можливість досягти оптимального балансу швидкодії, зручності програмного продукту і необхідних ресурсів на його створення.

ВИСНОВКИ

Розроблено програмний продукт на мові програмування Python для автоматичного розпізнавання пухлин на знімках МРТ головного мозку та їх подальшої сегментації. Це дозволить лікарям швидко та якісно отримати результати МРТ, що допоможе у виборі стратегії лікування пацієнтів. Використання глибоких нейронних мереж дозволяє ефективно визначати наявність пухлини на знімках МРТ головного мозку.

При навчанні моделей необхідно досягти мінімального значення помилки першого роду – уникнути ситуації, коли пухлина є на знімку МРТ, але модель її не виявляє. Це дозволить забезпечити більш точне виявлення пухлин, допоможе уникнути помилки у діагностуванні та лікуванні пацієнтів.

Для класифікації знімків навчено моделі ResNet-50, VGG-16, DenseNet-201 та запропоновану модель, яка містить 4 згорткових шари. Найбільш точні результати було отримано з використанням моделі DenseNet-201, яка має точність класифікації 95%. Для сегментації пухлин головного мозку навчено моделі ResUnet, ResUnet++ та Attention U-Net, і кожна з цих моделей показала високу точність сегментації. Однак, модель ResUnet виявилася кращою в локалізації малих областей пухлини порівняно з іншими моделями. Точність сегментації з використанням моделі ResUnet складає 95%.

Отримані результати дозволяють зробити висновок про доцільність використання розглянутих моделей для аналізу знімків МРТ головного мозку. Для покращення точності можна збільшити розмір вибірки МРТ-знімків, що дозволить більш детально дослідити структуру пухлини та вдосконалити алгоритми класифікації та сегментації. Також можна розширити архітектуру моделей глибокого навчання та використати 3D-моделі пухлин.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Рак в Україні, 2019-2020. Захворюваність, смертність, показники діяльності онкологічної служби. Федоренко З. П., Гулак Л. О., Михайлович Ю. Й. та інші. Бюлетень Національного канцер-реєстру України №22. – 2021. URL: http://www.ncru.inf.ua/publications/BULL_23/index.htm (дата звернення: 17.04.2023).
2. Зайченко Ю. П., Здор К. А., Гамидов Г. Диагностика МРТ-изображений опухолей головного мозга с использованием гибридных сверточных нейронечетких сетей // Системні дослідження та інформаційні технології, 2020. т.№ 1.-С.68-77. (дата звернення: 17.04.2023).
3. Чапалюк Б. В., Зайченко Ю. П. Використання рекурентних нейронних мереж для автоматичної діагностики раку легенів // Системні дослідження та інформаційні технології, 2019. т.№ 3.-С.33-40. (дата звернення: 17.04.2023).
4. Надеран М., Зайчченко Ю. П., Наполі А. Використання згорткових нейронних мереж для діагностування раку молочної залози // Системні дослідження та інформаційні технології, 2019. т.№ 4.-С.85-93. (дата звернення: 17.04.2023).
5. Abraham N., Khan N. M. A Novel Focal Tversky Loss Function with Improved Attention U-Net for Lesion Segmentation URL: https://arxiv.org/abs/1810.07842 (дата звернення: 24.04.2023).
6. Harun Bingol, Bilal Alatas. Classification of Brain Tumor Images using Deep Learning Methods. 2019. URL: https://dergipark.org.tr/en/download/article-file/1556139 (дата звернення: 24.04.2023).
7. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. 2015. URL: https://arxiv.org/abs/1412.6980 (дата звернення: 28.04.2023).

8. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. 2016. URL: https://arxiv.org/abs/1512.03385 (дата звернення: 01.05.2023).
9. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition. URL: https://arxiv.org/abs/1409.1556 (дата звернення: 01.05.2023).
10. Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. URL: https://ieeexplore.ieee.org/document/8099726 (дата звернення: 01.05.2023).
11. Jha D., Smedsrud P. H., Riegler M. A., Johansen D., de Lange T., Halvorsen P., Johansen H. D. ResUNet++: Advanced Architecture for Medical Image Segmentation. 2019. URL: https://arxiv.org/abs/1911.07067 (дата звернення: 01.05.2023).
12. Hu J., Shen L., Sun G. Squeeze-and-excitation networks. In Proceedings of IEEE conference on computer vision and pattern recognition (CVPR) (pp. 7132-7141).
13. Li, H., Xiong, P., An, J., & Wang, L. (2018). Pyramid attention network for semantic segmentation URL: https://arxiv.org/abs/1805.10180 (дата звернення: 01.05.2023)
14. Oktay, O., Schlemper J., Le Folgoc, Heinrich M., Misawa K., Mori K., Glocker B., Rueckert D. Attention U-Net: Learning where to look for the pancreas. URL: https://arxiv.org/abs/1804.03999.
15. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional networks for biomedical image segmentation. URL: https://arxiv.org/abs/1505.04597.
16. Dataset "Brain MRI segmentation". URL: https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import cv2
import random
import glob
from skimage import io
import tensorflow as tf
from tensorflow.python.keras import Sequential
from tensorflow import keras
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.applications.densenet import DenseNet201
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from keras import layers, optimizers
from keras.layers import *
from keras.models import Model
from keras.initializers import glorot_uniform
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint, L
earningRateScheduler
import keras.backend as K
import torch
import torch.nn.functional as F
import torch.nn as nn
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.model_selection import train_test_split
from IPython.display import display
from PIL import Image

data = pd.read_csv('/content/drive/My Drive/datasets/kaggle_3m/data.csv')
data_map = []
for sub_dir_path in glob.glob("/content/drive/My Drive/datasets/kaggle_3m/" + "*
"):
    try:
        dir_name = sub_dir_path.split('/')[-1]
        for file_name in os.listdir(sub_dir_path):
            image_path = sub_dir_path + '/' + file_name
            data_map.extend([dir_name, image_path])
```

```
    except Exception as e:
        print(e)

patient_id_and_path = pd.DataFrame({"patient_id": data_map[::2],
                                    "path": data_map[1::2]})
patient_id_and_path.head()

df_imgs = patient_id_and_path[~patient_id_and_path['path'].str.contains("mask")]
df_masks = patient_id_and_path[patient_id_and_path['path'].str.contains("mask")]
BASE_LEN = len("/content/drive/My Drive/datasets/kaggle_3m/TCGA_CS_4941_19960909
/TCGA_CS_4941_19960909_")
END_IMG_LEN = 4
END_MASK_LEN = 9

imgs = sorted(df_imgs["path"].values, key=lambda x: int(x[BASE_LEN:-
END_IMG_LEN]))
masks = sorted(df_masks["path"].values, key=lambda x: int(x[BASE_LEN:-
END_MASK_LEN]))

brain_df = pd.DataFrame({"patient_id": df_imgs.patient_id.values,
                         "image_path": imgs,
                         "mask_path": masks})

def positive_negative_diagnosis(mask_path):
    value = np.max(cv2.imread(mask_path))
    if value > 0:
        return 1
    else:
        return 0


brain_df["mask"] = brain_df["mask_path"].apply(lambda m: positive_negative_diagn
osis(m))
brain_df.head()

count = 0
i = 0
count_elements = 6
fig, axs = plt.subplots(count_elements,3, figsize=(20,30))
for mask in brain_df['mask']:
    if (mask):
        img = io.imread(brain_df.image_path[i])
        axs[count][0].title.set_text("Brain MRI")
        axs[count][0].title.set_fontsize(18)
        axs[count][0].imshow(img)

        mask = io.imread(brain_df.mask_path[i])
```

```
        axs[count][1].title.set_text("Mask")
        axs[count][1].title.set_fontsize(18)
        axs[count][1].imshow(mask, cmap='gray')

        img[mask==255] = (0,255,150)
        axs[count][2].title.set_text("Brain MRI with mask")
        axs[count][2].title.set_fontsize(18)
        axs[count][2].imshow(img)
        count +=1
    i += 1
    if (count==count_elements):
        break
fig.tight_layout()
fig.savefig('/content/drive/My Drive/brain_mri_with_origin_mask.png')

brain_df_train = brain_df.drop(columns=['patient_id'])
brain_df_train.shape
brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))

train, test = train_test_split(brain_df_train, test_size=0.15)
datagen = ImageDataGenerator(rescale=1. / 255., validation_split=0.15)

train_generator = datagen.flow_from_dataframe(
    dataframe=train,
    directory='./',
    x_col='image_path',
    y_col='mask',
    subset="training",
    batch_size=16,
    shuffle=True,
    class_mode="binary",
    target_size=(254, 254))

valid_generator = datagen.flow_from_dataframe(
    dataframe=train,
    directory='./',
    x_col='image_path',
    y_col='mask',
    subset="validation",
    batch_size=16,
    shuffle=True,
    class_mode="binary",
    target_size=(254, 254))

test_datagen = ImageDataGenerator(rescale=1. / 255.)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=test,
```

```
    directory='./',
    x_col='image_path',
    y_col='mask',
    batch_size=16,
    shuffle=False,
    class_mode='binary',
    target_size=(254, 254))

def visualization_history_clf(name_model, history):
  plt.figure(figsize=(12, 5))
  plt.subplot(1, 2, 1)
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title("Loss graph")
  plt.ylabel("Loss")
  plt.xlabel("Epochs")
  plt.legend(['train', 'val'])

  plt.subplot(1, 2, 2)
  plt.plot(history.history['accuracy'])
  plt.plot(history.history['val_accuracy'])
  plt.title("Accuracy coefficient")
  plt.ylabel("Accuracy")
  plt.xlabel("Epochs")
  plt.legend(['train', 'val'])

  plt.savefig(f'/content/drive/My Drive/result_classification_{name_model}.png')

def DenseNet():
    base_model = DenseNet201(weights='imagenet', include_top=False, input_tensor
=Input(shape=(254, 254, 3)))
    for layer in base_model.layers:
        layer.trainable = False
    x = base_model.output
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.3)(x)
    x = Dense(256, activation='relu')(x)
    predictions = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accura
cy"])
    return model

denseNet_model = DenseNet()
earlystopping = EarlyStopping(monitor='val_loss',
                              mode='min',
```

```
                                verbose=1,
                                patience=15
                                )
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/classifier-
DendeNet-weights.hdf5",
                                 verbose=1,
                                 save_best_only=True
                                 )
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                mode='min',
                                verbose=1,
                                patience=10,
                                min_delta=0.0001,
                                factor=0.2
                                )
callbacks = [checkpointer, earlystopping, reduce_lr]
history = denseNet_model.fit(train_generator,
                               steps_per_epoch=train_generator.n // train_generato
r.batch_size,
                               epochs=50,
                               validation_data=valid_generator,
                               validation_steps=valid_generator.n // valid_generat
or.batch_size,
                               callbacks=[checkpointer, earlystopping])

model_json = denseNet_model.to_json()
with open("/content/drive/My Drive/classifier-DenseNet.json", "w") as json_file:
    json_file.write(model_json)

visualization_history_clf('denseNet201', history)

_, acc = denseNet_model.evaluate(test_generator)
print("Test accuracy : {:.2f} %".format(acc*100))

def resNet50():
  base_model = ResNet50(weights = 'imagenet', include_top = False, input_tensor
= Input(shape=(254, 254, 3)))
  for layer in base_model.layers:
    layer.trainable = False
  x = base_model.output
  x = AveragePooling2D(pool_size=(3,3))(x)
  x = Flatten()(x)
  x = Dense(256, activation='relu')(x)
  x = Dropout(0.3)(x)
  x = Dense(256, activation='relu')(x)
  predictions = Dense(1, activation='sigmoid')(x)
  model = Model(inputs=base_model.input, outputs=predictions)
```

```
  model.compile(loss = 'binary_crossentropy', optimizer='adam', metrics= ["accur
acy"])
  return model

resNet50_model = resNet50()

checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/classifier-
resnet-weights.hdf5",
                               verbose=1,
                               save_best_only=True)
earlystopping = EarlyStopping(monitor='val_loss',
                              mode='min',
                              verbose=1,
                              patience=15)
history = resNet50_model.fit(train_generator,

                             epochs=50,
                             validation_data=valid_generator)
model_json = resNet50_model.to_json()
with open("/content/drive/My Drive/classifier-resnet50-
model.json","w") as json_file:
    json_file.write(model_json)
visualization_history_clf('resNet50', history)

_, acc = resNet50_model.evaluate(test_generator)
print("Test accuracy : {:.2f} %".format(acc*100))

def vgg16():
  base_model = VGG16(weights = 'imagenet', include_top = False, input_tensor = I
nput(shape=(254, 254, 3)))
  for layer in base_model.layers:
    layer.trainable = False
  x = base_model.output
  x = AveragePooling2D(pool_size=(3,3))(x)
  x = Flatten()(x)
  x = Dense(256, activation='relu')(x)
  x = Dropout(0.3)(x)
  x = Dense(256, activation='relu')(x)
  predictions = Dense(1, activation='sigmoid')(x)
  model = Model(inputs=base_model.input, outputs=predictions)
  model.compile(loss = 'binary_crossentropy', optimizer='adam', metrics= ["accur
acy"])
  return model
vgg16_model = vgg16()
earlystopping = EarlyStopping(monitor='val_loss',
                              min_delta=0.0001,
                              mode='min',
```

```
                                verbose=1,
                                patience=20,
                                restore_best_weights=True
                                )

checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/classifier-
vgg16.h5",
                                 monitor='val_accuracy',
                                 verbose=1,
                                 save_best_only=True
                                 )

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                mode='min',
                                verbose=1,
                                patience=10,
                                min_delta=0.0001,
                                factor=0.2
                                )

callbacks = [checkpointer, earlystopping, reduce_lr]

history = vgg16_model.fit(train_generator,
                            steps_per_epoch=train_generator.n // train_generator.b
atch_size,
                            epochs=50,
                            validation_data=valid_generator,
                            validation_steps=valid_generator.n // valid_generator.
batch_size,
                            callbacks=callbacks)
model_json = vgg16_model.to_json()
with open("classifier-vgg16-model.json","w") as json_file:
    json_file.write(model_json)

visualization_history_clf('vgg16', history)


_, acc = vgg16_model.evaluate(test_generator)
print("Test accuracy : {:.2f} %".format(acc*100))
earlystopping = EarlyStopping(monitor='val_loss',
                                min_delta=0.0001,
                                mode='min',
                                verbose=1,
                                patience=20,
                                restore_best_weights=True
                                )
```

```
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/custom_model.h5
",
                               monitor='val_accuracy',
                               verbose=1,
                               save_best_only=True
                               )
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              mode='min',
                              verbose=1,
                              patience=10,
                              min_delta=0.0001,
                              factor=0.2
                              )
callbacks = [checkpointer, earlystopping, reduce_lr]
model = Sequential()

model.add(Conv2D(filters=16, kernel_size=(3, 3), input_shape=(254, 254, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=32, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(rate=0.3))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']
)
```

```
history = model.fit(train_generator,
                              steps_per_epoch=train_generator.n // train_generato
r.batch_size,
                              epochs=40,
                              validation_data=valid_generator,
                              validation_steps=valid_generator.n // valid_generat
or.batch_size
                              )
visualization_history_clf('own ', history)
test_loss, test_acc = model.evaluate(test_generator, steps=test_generator.n // t
est_generator.batch_size)
print('Test accuracy:', test_acc)

brain_df_mask = brain_df[brain_df['mask'] == 1]
brain_df_mask.shape


X_train, X_val = train_test_split(brain_df_mask, test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)
print("Train size is {}, valid size is {} & test size is {}".format(len(X_train)
, len(X_val), len(X_test)))
train_ids = list(X_train.image_path)
train_mask = list(X_train.mask_path)
val_ids = list(X_val.image_path)
val_mask= list(X_val.mask_path)


class DataGenerator(tf.keras.utils.Sequence):
  def __init__(self, ids , mask, image_dir = './', batch_size = 16, img_h = 256,
 img_w = 256, shuffle = True):

    self.ids = ids
    self.mask = mask
    self.image_dir = image_dir
    self.batch_size = batch_size
    self.img_h = img_h
    self.img_w = img_w
    self.shuffle = shuffle
    self.on_epoch_end()

  def __len__(self):
    return int(np.floor(len(self.ids)) / self.batch_size)

  def __getitem__(self, index):
    indexes = self.indexes[index* self.batch_size : (index+1) * self.batch_size]
    list_ids = [self.ids[i] for i in indexes]
    list_mask = [self.mask[i] for i in indexes]
```

```
    X, y = self.__data_generation(list_ids, list_mask)
    return X, y

  def on_epoch_end(self):
    self.indexes = np.arange(len(self.ids))
    if self.shuffle:
      np.random.shuffle(self.indexes)

  def __data_generation(self, list_ids, list_mask):
    X = np.empty((self.batch_size, self.img_h, self.img_w, 3))
    y = np.empty((self.batch_size, self.img_h, self.img_w, 1))
    for i in range(len(list_ids)):
      img_path = str(list_ids[i])
      mask_path = str(list_mask[i])
      img = io.imread(img_path)
      mask = io.imread(mask_path)
      img = cv2.resize(img,(self.img_h,self.img_w))
      img = np.array(img, dtype = np.float64)
      mask = cv2.resize(mask,(self.img_h,self.img_w))
      mask = np.array(mask, dtype = np.float64)
      img -= img.mean()
      img /= img.std()
      mask -= mask.mean()
      mask /= mask.std()
      X[i,] = img
      y[i,] = np.expand_dims(mask, axis = 2)
    y = (y > 0).astype(int)

    return X, y
train_data = DataGenerator(train_ids, train_mask)
val_data = DataGenerator(val_ids, val_mask)

smooth = 0.00001
def tversky(y_true, y_pred):
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1-y_pred_pos))
    false_pos = K.sum((1-y_true_pos)*y_pred_pos)
    alpha = 0.7
    return (true_pos + smooth)/(true_pos + alpha*false_neg + (1-
alpha)*false_pos + smooth)

def focal_tversky(y_true,y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
```

```
    pt_1 = tversky(y_true, y_pred)
    gamma = 0.75
    return K.pow((1-pt_1), gamma)

def tversky_loss(y_true, y_pred):
    return 1 - tversky(y_true,y_pred)

def dice(y_true, y_pred):
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1-y_pred_pos))
    false_pos = K.sum((1-y_true_pos)*y_pred_pos)
    alpha = 0.5
    return (true_pos + smooth)/(true_pos + alpha*false_neg + (1-
alpha)*false_pos + smooth)

def jaccard(y_true, y_pred):
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1-y_pred_pos))
    false_pos = K.sum((1-y_true_pos)*y_pred_pos)
    alpha = 1
    return (true_pos + smooth)/(true_pos + alpha*false_neg + alpha*false_pos + s
mooth)


adam = tf.keras.optimizers.Adam()
def visualization_history_seg(name_model, history, name_metric1, name_metric2):
  plt.figure(figsize=(12,5))
  plt.subplot(1,2,1)
  plt.plot(history.history[name_metric1]);
  plt.plot(history.history[f'val_{name_metric1}'])
  plt.title(f'{name_metric1} graph')
  plt.ylabel(name_metric1)
  plt.xlabel("Epochs")
  plt.legend(['train', 'val'])

  plt.subplot(1,2,2)
  plt.plot(history.history[name_metric2])
  plt.plot(history.history[f'val_{name_metric2}'])
  plt.title(f'{name_metric2} graph');
  plt.ylabel("Accuracy")
  plt.xlabel("Epochs")
  plt.legend(['train', 'val'])
  plt.savefig(f'/content/drive/My Drive/seg_{name_metric1}-{name_metric1}.png')
```

```
def batchnorm_relu(inputs=(256, 256, 3)):
     x = BatchNormalization()(inputs)
     x = Activation("relu")(x)
     return x

def residual_block(inputs, num_filters, strides=1):
     x = batchnorm_relu(inputs)
     x = Conv2D(num_filters, 3, padding="same", strides=strides)(x)
     x = batchnorm_relu(x)
     x = Conv2D(num_filters, 3, padding="same", strides=1)(x)
     s = Conv2D(num_filters, 1, padding="same", strides=strides)(inputs)
     x = x + s
     return x

def decoder_block(inputs, skip_features, num_filters):
      x = UpSampling2D((2, 2))(inputs)
      x = Concatenate()([x, skip_features])
      x = residual_block(x, num_filters, strides=1)
      return x

def build_resunet(input_shape):
      inputs = Input(input_shape)
      """ Endoder 1 """
      x = Conv2D(64, 3, padding="same", strides=1)(inputs)
      x = batchnorm_relu(x)
      x = Conv2D(64, 3, padding="same", strides=1)(x)
      s = Conv2D(64, 1, padding="same")(inputs)
      s1 = x + s
      """ Encoder 2, 3 """
      s2 = residual_block(s1, 128, strides=2)
      s3 = residual_block(s2, 256, strides=2)
      """ Bridge """
      b = residual_block(s3, 512, strides=2)
      """ Decoder 1, 2, 3 """
      x = decoder_block(b, s3, 256)
      x = decoder_block(x, s2, 128)
      x = decoder_block(x, s1, 64)
      """ Classifier """
      outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(x)
      model = Model(inputs, outputs, name="RESUNET")
      return model
resUnet_model = build_resunet()
resUnet_model.compile(optimizer = adam,
                  loss = focal_tversky,
                  metrics = [tversky, dice, jaccard]
                 )
earlystopping = EarlyStopping(monitor='val_loss',
```

```
                                mode='min',
                                verbose=1,
                                patience=20
                               )
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/resUNetModel-
weightsM.hdf5",
                                verbose=1,
                                save_best_only=True
                               )
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                mode='min',
                                verbose=1,
                                patience=10,
                                min_delta=0.0001,
                                factor=0.2
                               )
history = resUnet_model.fit(train_data,
                  epochs = 60,
                  validation_data = val_data,
                  callbacks = [checkpointer, earlystopping, reduce_lr]
                 )
model_resUNet_json = resUnet_model.to_json()
with open("/content/drive/My Drive/resUNetModelM.json", "w") as json_file:
    json_file.write(model_resUNet_json)

visualization_history_seg('resunet', history, 'loss', 'tversky')
visualization_history_seg('resunet', history, 'dice', 'jaccard')

test_ids = list(X_test.image_path)
test_mask = list(X_test.mask_path)
test_data = DataGenerator(test_ids, test_mask)
resUnet_model.evaluate(test_data)

loss, tversky, dice, jaccard  = resUnet_model.evaluate(test_data)
print("tversky index is {:.2f}%".format(tversky*100))
print("dice index is {:.2f}%".format(dice*100))
print("jaccard index is {:.2f}%".format(jaccard*100))


def squeeze_excite_block(inputs, ratio=8):
    init = inputs
    channel_axis = -1
    filters = init.shape[channel_axis]
    se_shape = (1, 1, filters)
    se = GlobalAveragePooling2D()(init)
    se = Reshape(se_shape)(se)
```

```
    se = Dense(filters // ratio, activation='relu', kernel_initializer='he_norma
l', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid', kernel_initializer='he_normal', us
e_bias=False)(se)
    x = Multiply()([init, se])
    return x

def stem_block(x, n_filter, strides):
    x_init = x
    x = Conv2D(n_filter, (3, 3), padding="same", strides=strides)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(n_filter, (3, 3), padding="same")(x)
    s  = Conv2D(n_filter, (1, 1), padding="same", strides=strides)(x_init)
    s = BatchNormalization()(s)
    x = Add()([x, s])
    x = squeeze_excite_block(x)
    return x

def resnet_block(x, n_filter, strides=1):
    x_init = x
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(n_filter, (3, 3), padding="same", strides=strides)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(n_filter, (3, 3), padding="same", strides=1)(x)
    s  = Conv2D(n_filter, (1, 1), padding="same", strides=strides)(x_init)
    s = BatchNormalization()(s)
    x = Add()([x, s])
    x = squeeze_excite_block(x)
    return x

def aspp_block(x, num_filters, rate_scale=1):
    x1 = Conv2D(num_filters, (3, 3), dilation_rate=(6 * rate_scale, 6 * rate_sca
le), padding="same")(x)
    x1 = BatchNormalization()(x1)

    x2 = Conv2D(num_filters, (3, 3), dilation_rate=(12 * rate_scale, 12 * rate_s
cale), padding="same")(x)
    x2 = BatchNormalization()(x2)

    x3 = Conv2D(num_filters, (3, 3), dilation_rate=(18 * rate_scale, 18 * rate_s
cale), padding="same")(x)
    x3 = BatchNormalization()(x3)

    x4 = Conv2D(num_filters, (3, 3), padding="same")(x)
```

```
    x4 = BatchNormalization()(x4)

    y = Add()([x1, x2, x3, x4])
    y = Conv2D(num_filters, (1, 1), padding="same")(y)
    return y

def attetion_block(g, x):
    """
        g: Output of Parallel Encoder block
        x: Output of Previous Decoder block
    """
    filters = x.shape[-1]
    g_conv = BatchNormalization()(g)
    g_conv = Activation("relu")(g_conv)
    g_conv = Conv2D(filters, (3, 3), padding="same")(g_conv)
    g_pool = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(g_conv)
    x_conv = BatchNormalization()(x)
    x_conv = Activation("relu")(x_conv)
    x_conv = Conv2D(filters, (3, 3), padding="same")(x_conv)
    gc_sum = Add()([g_pool, x_conv])
    gc_conv = BatchNormalization()(gc_sum)
    gc_conv = Activation("relu")(gc_conv)
    gc_conv = Conv2D(filters, (3, 3), padding="same")(gc_conv)
    gc_mul = Multiply()([gc_conv, x])
    return gc_mul
class ResUnetPlusPlus:
    def __init__(self, input_size=256):
        self.input_size = input_size

    def build_model(self):
        n_filters = [16, 32, 64, 128, 256]
        inputs = Input((self.input_size, self.input_size, 3))
        c0 = inputs
        c1 = stem_block(c0, n_filters[0], strides=1)
        # Encoder
        c2 = resnet_block(c1, n_filters[1], strides=2)
        c3 = resnet_block(c2, n_filters[2], strides=2)
        c4 = resnet_block(c3, n_filters[3], strides=2)
        # Bridge
        b1 = aspp_block(c4, n_filters[4])
        # Decoder
        d1 = attetion_block(c3, b1)
        d1 = UpSampling2D((2, 2))(d1)
        d1 = Concatenate()([d1, c3])
        d1 = resnet_block(d1, n_filters[3])
        d2 = attetion_block(c2, d1)
        d2 = UpSampling2D((2, 2))(d2)
```

```
        d2 = Concatenate()([d2, c2])
        d2 = resnet_block(d2, n_filters[2])
        d3 = attetion_block(c1, d2)
        d3 = UpSampling2D((2, 2))(d3)
        d3 = Concatenate()([d3, c1])
        d3 = resnet_block(d3, n_filters[1])
        outputs = aspp_block(d3, n_filters[0])
        outputs = Conv2D(1, (1, 1), padding="same")(outputs)
        outputs = Activation("sigmoid")(outputs)
        model = Model(inputs, outputs)
        return model

resUnetPlusPlus = ResUnetPlusPlus()
resUnetPlusPlus_model = resUnetPlusPlus.build_model()


resUnetPlusPlus_model.compile(optimizer = adam,
                  loss = focal_tversky,
                  metrics = [tversky, dice, jaccard]
                 )
earlystopping = EarlyStopping(monitor='val_loss',
                              mode='min',
                              verbose=1,
                              patience=20
                             )
checkpointer = ModelCheckpoint(filepath="/content/drive/My Drive/ResUNetPlusPlus
.hdf5",
                               verbose=1,
                               save_best_only=True
                              )
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              mode='min',
                              verbose=1,
                              patience=10,
                              min_delta=0.0001,
                              factor=0.2
                             )
history = resUnetPlusPlus_model.fit(train_data,
                  epochs = 60,
                  validation_data = val_data,
                  callbacks = [checkpointer, earlystopping, reduce_lr]
                 )
resUnetPlusPlus_json = resUnetPlusPlus_model.to_json()
with open("/content/drive/My Drive/ResUNetPlusPlusNew.json", "w") as json_file:
    json_file.write(resUnetPlusPlus_json)

visualization_history_seg('resunet++', history, 'loss', 'tversky')
```

```
visualization_history_seg('resunet++', history, 'dice', 'jaccard')

test_ids = list(X_test.image_path)
test_mask = list(X_test.mask_path)
test_data = DataGenerator(test_ids, test_mask)


loss, tversky, dice, jaccard  = resUnet_model.evaluate(train_data)
print("tversky index is {:.2f}%".format(tversky*100))
print("dice index is {:.2f}%".format(dice*100))
print("jaccard index is {:.2f}%".format(jaccard*100))


class attention_unet():
  def __init__(self,img_rows=256,img_cols=256):
    self.img_rows=img_rows
    self.img_cols=img_cols
    self.img_shape=(self.img_rows,self.img_cols,3)
    self.df=64
    self.uf=64

  def build_unet(self):
    def conv2d(layer_input,filters,dropout_rate=0,bn=False):
      d=layers.Conv2D(filters,kernel_size=(3,3),strides=(1,1),
                      padding='same')(layer_input)
      if bn:
        d=layers.BatchNormalization()(d)
      d=layers.Activation('relu')(d)

      d=layers.Conv2D(filters,kernel_size=(3,3),strides=(1,1),
                      padding='same')(d)
      if bn:
        d=layers.BatchNormalization()(d)
      d=layers.Activation('relu')(d)
      if dropout_rate:
        d=layers.Dropout(dropout_rate)(d)
      return d

    def deconv2d(layer_input,filters,bn=False):
      u=layers.UpSampling2D((2,2))(layer_input)
      u=layers.Conv2D(filters,kernel_size=(3,3),strides=(1,1),
                      padding='same')(u)
      if bn:
        u=layers.BatchNormalization()(u)
      u=layers.Activation('relu')(u)
      return u

    def attention_block(F_g,F_l,F_int,bn=False):
```

```
    g=layers.Conv2D(F_int,kernel_size=(1,1),strides=(1,1),
                    padding='valid')(F_g)
    if bn:
      g=layers.BatchNormalization()(g)
    x=layers.Conv2D(F_int,kernel_size=(1,1),strides=(1,1),
                    padding='valid')(F_l)
    if bn:
      x=layers.BatchNormalization()(x)
    psi=layers.Add()([g,x])
    psi=layers.Activation('relu')(psi)

    psi=layers.Conv2D(1,kernel_size=(1,1),strides=(1,1),
                      padding='valid')(psi)

    if bn:
      psi=layers.BatchNormalization()(psi)
    psi=layers.Activation('sigmoid')(psi)

    return layers.Multiply()([F_l,psi])

inputs=layers.Input(shape=self.img_shape)

conv1=conv2d(inputs,self.df)
pool1=layers.MaxPooling2D((2,2))(conv1)

conv2=conv2d(pool1,self.df*2,bn=True)
pool2=layers.MaxPooling2D((2,2))(conv2)

conv3=conv2d(pool2,self.df*4,bn=True)
pool3=layers.MaxPooling2D((2,2))(conv3)

conv4=conv2d(pool3,self.df*8,dropout_rate=0.5,bn=True)
pool4=layers.MaxPooling2D((2,2))(conv4)

conv5=conv2d(pool4,self.df*16,dropout_rate=0.5,bn=True)

up6=deconv2d(conv5,self.uf*8,bn=True)
conv6=attention_block(up6,conv4,self.uf*8,bn=True)
up6=layers.Concatenate()([up6,conv6])
conv6=conv2d(up6,self.uf*8)

up7=deconv2d(conv6,self.uf*4,bn=True)
conv7=attention_block(up7,conv3,self.uf*4,bn=True)
up7=layers.Concatenate()([up7,conv7])
conv7=conv2d(up7,self.uf*4)

up8=deconv2d(conv7,self.uf*2,bn=True)
```

```
    conv8=attention_block(up8,conv2,self.uf*2,bn=True)
    up8=layers.Concatenate()([up8,conv8])
    conv8=conv2d(up8,self.uf*2)

    up9=deconv2d(conv8,self.uf,bn=True)
    conv9=attention_block(up9,conv1,self.uf,bn=True)
    up9=layers.Concatenate()([up9,conv9])
    conv9=conv2d(up9,self.uf)

    outputs=layers.Conv2D(1,kernel_size=(1,1),strides=(1,1),
                          activation='sigmoid')(conv9)

    model=Model(inputs=inputs,outputs=outputs)
    return model
def BatchActivate(x):
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    return x

def convolution_block(x, filters, size, strides=(1,1), padding='same',
                      activation=True):
    x = Conv2D(filters, size, strides=strides, padding=padding)(x)
    if activation == True:
        x = BatchActivate(x)
    return x

def residual_block(blockInput, num_filters=16, batch_activate = False):
    x = BatchActivate(blockInput)
    x = convolution_block(x, num_filters, (3,3))
    x = convolution_block(x, num_filters, (3,3), activation=False)
    x = Add()([x, blockInput])
    if batch_activate:
        x = BatchActivate(x)
    return x


model=attention_unet()
attention_unet_model=model.build_unet()
filepath_weight = "/content/drive/My Drive/Attention_UNet_model.hdf5"
attention_unet_model.compile(optimizer = adam, loss = focal_tversky,
                             metrics = [tversky, dice, jaccard])

earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
                              patience=20)

checkpointer = ModelCheckpoint(filepath=filepath_weight, verbose=1,
                               save_best_only=True)
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', mode='min', verbose=1,
                              patience=10, min_delta=0.0001, factor=0.2)

history = attention_unet_model.fit(train_data, epochs = 60,
                      validation_data = val_data,
                     callbacks = [checkpointer, earlystopping, reduce_lr])
attention_unet_json = attention_unet_model.to_json()
with open("attention_unet.json", "w") as json_file:
    json_file.write(attention_unet_json)

test_ids = list(X_test.image_path)
test_mask = list(X_test.mask_path)
test_data = DataGenerator(test_ids, test_mask)
loss, tversky, dice, jaccard  = attention_unet_json.evaluate(test_data)
print("tversky index is {:.2f}%".format(tversky*100))
print("dice index is {:.2f}%".format(dice*100))
print("jaccard index is {:.2f}%".format(jaccard*100))
Main_activity.py

from tkinter import *
from tkinter import filedialog
from PIL import Image, ImageTk
import research_MRI
from message_boxs import open_message_errorbox, open_message_infobox
from settings import DATA_DIRECTORY_PATH, IMAGE_SIZE


class MainActivity:
    def __init__(self):
        self.root = Tk()
        self.mri_image_path = None
        self.label_tumor_brain_mri = None
        self.label_brain_mri = None
        self.display_img_tumor = None
        self.display_mri_img = None
        self.frame_image_tumor = None
        self.frame_image_mri = None

    def load_mri_image(self):
        image_path  =  filedialog.askopenfilename(initialdir=DATA_DIRECTORY_PATH,
filetypes=[("Image file", ".tif")])
        if image_path is not None:
            try:
                image = Image.open(image_path)
                tkImage = ImageTk.PhotoImage(image)
                self.frame_image_mri  =  Frame(self.root,  height=IMAGE_SIZE[0],
width=IMAGE_SIZE[1])
                self.frame_image_mri.pack_propagate(0)
                self.frame_image_mri.place(x=120, y=150)
                label = Label(self.frame_image_mri, image=tkImage)
                label.pack(fill=BOTH, expand=1)
                self.display_mri_img = label
                self.display_mri_img.image = tkImage
                self.display_mri_img.pack()
```

```
                self.mri_image_path = image_path
            except:
                open_message_errorbox("No image selected.")

    def load_mri_image_with_tumor(self, result_image_path):
        image = Image.open(result_image_path)
        tkImage = ImageTk.PhotoImage(image)
        self.frame_image_tumor    =    Frame(self.root,    height=IMAGE_SIZE[0],
width=IMAGE_SIZE[0])
        self.frame_image_tumor.pack_propagate(0)
        self.frame_image_tumor.place(x=600, y=150)
        label = Label(self.frame_image_tumor, image=tkImage)
        label.pack(fill=BOTH, expand=1)
        self.display_img_tumor = label
        self.display_img_tumor.image = tkImage
        self.display_img_tumor.pack()

    def show_label_mri_image(self):
        self.label_brain_mri     =     Label(self.root,     text="Brain     MRI",
background="#364156", padx=10, pady=5,
                                    foreground="white",
                                    font=('arial', 10, 'bold'))
        self.label_brain_mri.place(x=200, y=415)

    def show_label_tumor_mri_image(self):
        self.label_tumor_brain_mri   =   Label(self.root,   text="Brain   tumor",
background="#364156", padx=10, pady=5,
                                        foreground="white",
                                        font=('arial', 10, 'bold'))
        self.label_tumor_brain_mri.place(x=680, y=415)

    def show_find_tumor_button(self):
        find_tumor_btn    =    Button(self.root,    text="Find    brain    tumor",
command=self.show_brain_mri_with_tumor, padx=10,
                                pady=5)
        find_tumor_btn.configure(background="#364156",        foreground="white",
font=('arial', 10, 'bold'))
        find_tumor_btn.place(x=80, y=650)

    def show_load_mri_btn(self):
        load_mri_image    =    Button(self.root,    text="Load    brain    MRI",
command=self.research_mri_image, padx=15, pady=5)
        load_mri_image.configure(background="#364156",        foreground="white",
font=('arial', 10, 'bold'))
        load_mri_image.place(x=80, y=600)

    def show_exit_btn(self):
        exit_btn  =  Button(self.root,  text="Exit",  command=self.root.destroy,
padx=50, pady=5)
        exit_btn.configure(background="#364156",              foreground="white",
font=('arial', 10, 'bold'))
        exit_btn.place(x=855, y=755)

    def clean_images_info(self):
        if self.label_tumor_brain_mri is not None:
            self.label_tumor_brain_mri.destroy()
            self.label_tumor_brain_mri = None
        if self.display_img_tumor is not None:
            self.display_img_tumor.destroy()
```

```
            self.display_img_tumor = None
            self.frame_image_tumor.destroy()
            self.frame_image_tumor = None
        if self.display_mri_img is not None:
            self.display_mri_img.destroy()
            self.display_mri_img = None
            self.frame_image_mri.destroy()
            self.frame_image_mri = None
            self.label_brain_mri.destroy()
            self.label_brain_mri = None

    def research_mri_image(self):
        self.clean_images_info()
        self.load_mri_image()
        if self.display_mri_img is not None:
            self.show_label_mri_image()

    def show_brain_mri_with_tumor(self):
        if self.display_mri_img is not None:
            result = research_MRI.research_mri(self.mri_image_path)
            if result:
                open_message_infobox("Brain tumor is find.")
                self.load_mri_image_with_tumor(result)
                self.show_label_tumor_mri_image()
            else:
                open_message_infobox("Brain tumor is not find.")
        else:
            open_message_errorbox("Image of brain MRI is not find.")

    def launching(self):
        img = ImageTk.PhotoImage(Image.open("assert/background.jpg"))
        canvas1 = Canvas(self.root, width=400, height=400)
        canvas1.pack(fill="both", expand=True)
        canvas1.create_image(0, 0, image=img, anchor="nw")
        self.root.geometry('1000x800')
        self.show_load_mri_btn()
        self.show_find_tumor_button()
        self.show_exit_btn()
        self.root.mainloop()
```

research_MRI.py

```
import cv2
from skimage import io
import tensorflow as tf
import numpy as np
import pandas as pd

from classification_model import ClassificationModel
from segmentation_model import SegmentationModel
from settings import DATA_DIRECTORY_PATH


def find_prediction(image_test, model_clf, model_seg):
    mask, image_id, has_mask = [], [], []
    for i in image_test:
        img = io.imread(i)
        img = img * 1. / 255.
        img = cv2.resize(img, (256, 256))
```

```
        img = np.array(img, dtype=np.float64)
        img = np.reshape(img, (1, 256, 256, 3))
        is_defect = model_clf.predict(img)
        if np.argmax(is_defect) == 0:
            image_id.append(i)
            has_mask.append(0)
            mask.append('No tumor')
            continue
        X = np.empty((1, 256, 256, 3))
        img = io.imread(i)
        img = cv2.resize(img, (256, 256))
        img = np.array(img, dtype=np.float64)
        img -= img.mean()
        img /= img.std()
        X[0,] = img
        predict = model_seg.predict(X)
        if predict.round().astype(int).sum() == 0:
            image_id.append(i)
            has_mask.append(0)
            mask.append('No tumor')
        else:
            image_id.append(i)
            has_mask.append(1)
            mask.append(predict)
    return pd.DataFrame({'image_path': image_id,
                         'predicted_mask': mask,
                         'has_mask': has_mask})


def tumor_selection(df_prediction):
    pred = np.array(df_prediction.predicted_mask[0]).squeeze().round()
    img = io.imread(df_prediction.image_path[0])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img[pred == 1] = (0, 255, 150)
    return img


def research_mri(path):
    res_image_path = path.split('.')[0]
    res_image_path += '_with_tumor.jpg'
    clf_model = ClassificationModel()
    clf_model.compile_model()
    seg_model = SegmentationModel()
    seg_model.compile_model()
    df_prediction      =      find_prediction([path],      clf_model.get_model(),
seg_model.get_model())
    if df_prediction.has_mask[0] == 1:
        image = tumor_selection(df_prediction)
        cv2.imwrite(res_image_path, image)
        return res_image_path
    else:
        return False
setting.py

DATA_DIRECTORY_PATH = "C:\\Brain_MRI_images\\"
IMAGE_SIZE = (256, 256)


message_boxs.py
```

```
from tkinter import messagebox


def open_message_infobox(message):
    messagebox.showinfo(title="Result", message=message)


def open_message_errorbox(message):
    messagebox.showerror(title="Error", message=message)
```

paths_to_models.py

```
CLF_DENSENET_MODEL = 'assert/models/clf_models/DenseNet.json'
CLF_DENSENET_WEIGHTS = 'assert/models/clf_models/DenseNet.hdf5'

SEG_RESNET_PLUS_PLUS_MODEL = 'assert/models/seg_models/ResUNetPlusPlusNew.json'
SEG_RESNET_PLUS_PLUS_WEIGHTS = 'assert/models/seg_models/ResUNetPlusPlus.hdf5'
```

segmentation_model.py

```
from keras.models import model_from_json
import tensorflow as tf

from loss import focal_tversky
from metrics import tversky, dice, jaccard
from          paths_to_models          import          SEG_RESNET_PLUS_PLUS_MODEL,
SEG_RESNET_PLUS_PLUS_WEIGHTS


def create_model_from_json(json_path):
    json_file = open(json_path, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    model = model_from_json(loaded_model_json)
    return model


class SegmentationModel:
    def __init__(self):
        self.seg_model = create_model_from_json(SEG_RESNET_PLUS_PLUS_MODEL)
        self.seg_model.load_weights(SEG_RESNET_PLUS_PLUS_WEIGHTS)
        self.adam = tf.keras.optimizers.Adam()

    def compile_model(self):
        self.seg_model.compile(optimizer=self.adam,           loss=focal_tversky,
metrics=[tversky, dice, jaccard])

    def get_model(self):
        return self.seg_model
```

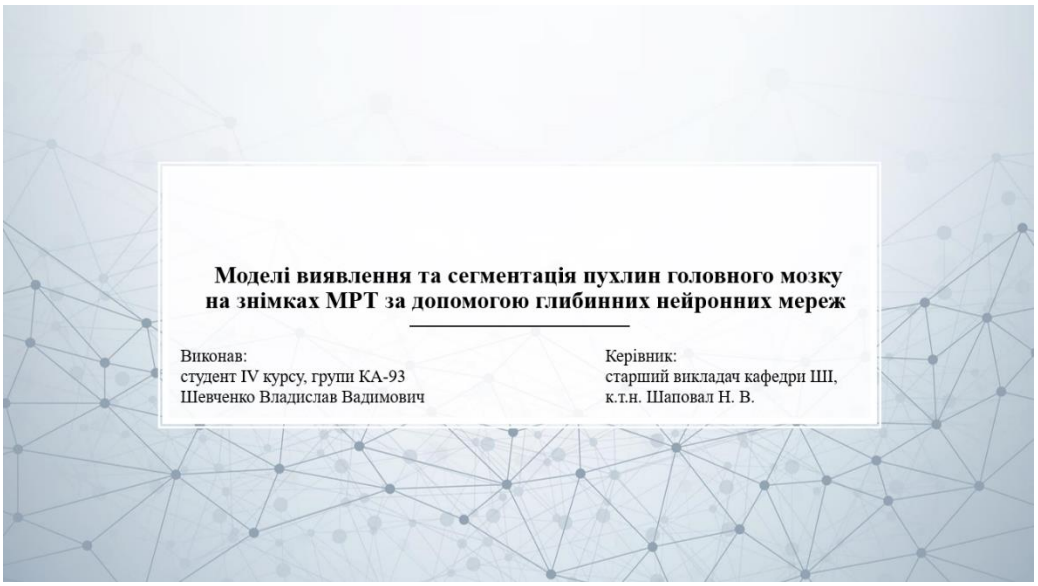
classification_model.py

```
from keras.models import model_from_json

from paths_to_models import CLF_DENSENET_MODEL, CLF_DENSENET_WEIGHTS


def create_model_from_json(json_path):
    json_file = open(json_path, 'r')
```

```
    loaded_model_json = json_file.read()
    json_file.close()
    model = model_from_json(loaded_model_json)
    return model


class ClassificationModel:
    def __init__(self):
        self.clf_model = create_model_from_json(CLF_DENSENET_MODEL)
        self.clf_model.load_weights(CLF_DENSENET_WEIGHTS)

    def compile_model(self):
        self.clf_model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=["accuracy"])

    def get_model(self):
        return self.clf_model


loss.py

import keras.backend as K
import tensorflow as tf


def tversky(y_true, y_pred):
    smooth = 1
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1 - y_pred_pos))
    false_pos = K.sum((1 - y_true_pos) * y_pred_pos)
    alpha = 0.7
    return (true_pos + smooth) / (true_pos + alpha * false_neg + (1 - alpha) *
false_pos + smooth)


def focal_tversky(y_true, y_pred):
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)

    pt_1 = tversky(y_true, y_pred)
    gamma = 0.75
    return K.pow((1 - pt_1), gamma)


metrics.py

import keras.backend as K


def tversky(y_true, y_pred):
    smooth = 1
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1 - y_pred_pos))
    false_pos = K.sum((1 - y_true_pos) * y_pred_pos)
    alpha = 0.7
```

```
    return (true_pos + smooth) / (true_pos + alpha * false_neg + (1 - alpha) *
false_pos + smooth)


def dice(y_true, y_pred):
    smooth = 1
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1 - y_pred_pos))
    false_pos = K.sum((1 - y_true_pos) * y_pred_pos)
    alpha = 0.5
    return (true_pos + smooth) / (true_pos + alpha * false_neg + (1 - alpha) *
false_pos + smooth)


def jaccard(y_true, y_pred):
    smooth = 1
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1 - y_pred_pos))
    false_pos = K.sum((1 - y_true_pos) * y_pred_pos)
    alpha = 1
    return (true_pos + smooth) / (true_pos + alpha * false_neg + alpha * false_pos
+ smooth)
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ



Об'єкт дослідження – знімки магнітно-резонансної томографії головного мозку людини.

Предмет дослідження – методи обробки зображень, мережі для класифікації знімків та сегментації пухлин головного мозку.

Мета роботи – розробка програмного продукту для класифікації знімків головного мозку та у разі виявлення новоутворень їх сегментація з використанням методів глибокого навчання.

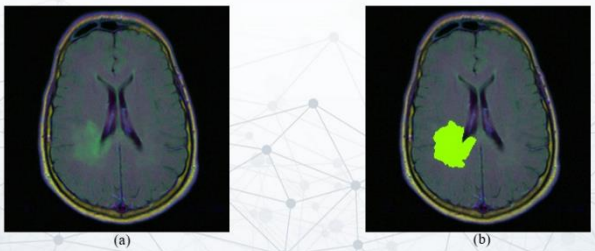
Актуальність роботи

Аналіз знімків головного мозку є важливим етапом у лікуванні. Традиційний метод, який включає ручний аналіз зображень лікарем, має обмежену ефективність через його потенційну неточність та може призвести до несвоєчасного виявлення злоякісних пухлин.

Сучасні алгоритми комп'ютерної обробки медичних зображень дозволяють автоматично визначати розміри та межі пухлини на знімках МРТ головного мозку. Визначення меж та розмірів пухлини на знімку МРТ дозволяє отримати детальну інформацію про її структуру. Це покращує якість діагностики, допомагає у виборі стратегії лікування та збільшує шанси на успішне лікування й повне одужання пацієнта.

Вхідні дані

У дослідженні використано набір зображень МРТ головного мозку та областей, що відповідають пухлинам, які виділені окремим кольором. Вибірку було обрано на платформі Kaggle, яка складається із 3929 знімків МРТ, 65% яких містять пухлину. Зображення знімків МРТ представлене в форматі RGB (Red Green Blue), тобто кожний піксель має 3 значення, кожне з яких може набувати значення від 0 до 255. Розмір зображень – 256×256.



(a) (b)

Рисунок 1 (a) – знімок МРТ мозку, (b) – знімок МРТ мозку з маскою

Моделі класифікації

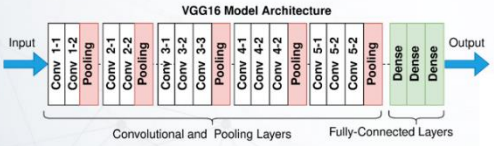


Рисунок 3 – Архітектура мережі VGG-16

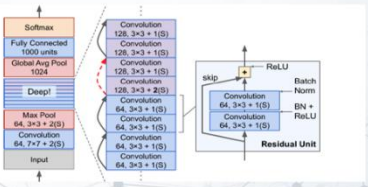


Рисунок 4 – Архітектура мережі ResNet-50

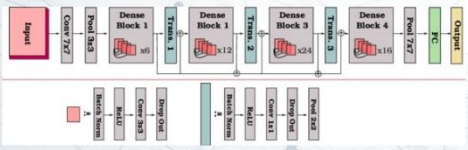


Рисунок 5 – Архітектура мережі DenseNet-201

Запропонована модель класифікації

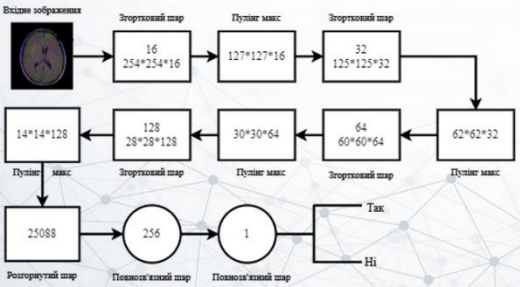


Рисунок 6 – Структура запропонованої моделі

Рисунок 7 – Графік функції активації ReLU

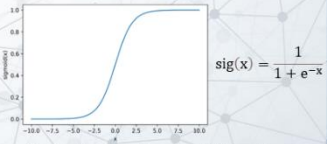


Рисунок 8 – Графік функції активації Sigmoid

Моделі сегментації

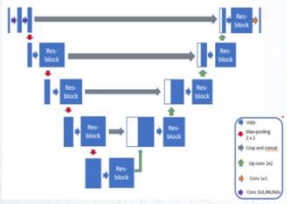


Рисунок 9 – Архітектура мережі ResUnet

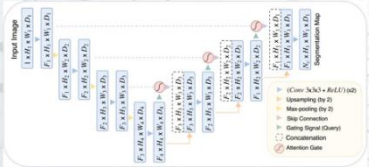


Рисунок 10 – Архітектура мережі Attention U-Net

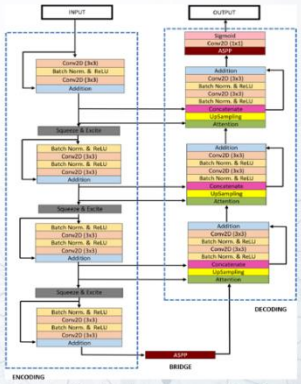


Рисунок 11 – Архітектура мережі ResUnet++

Метрики якості

Для моделей класифікації

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{F\_score} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Для моделей сегментації

$$T_{0.7,0.3} = \frac{TP}{TP + 0.7FP + 0.3FN}$$

$$\text{Dice} = \frac{2TP}{2TP + FP + FN}$$

$$\text{Jaccard} = \frac{TP}{TP + FP + FN}$$

Аналіз результатів навчання моделей класифікації

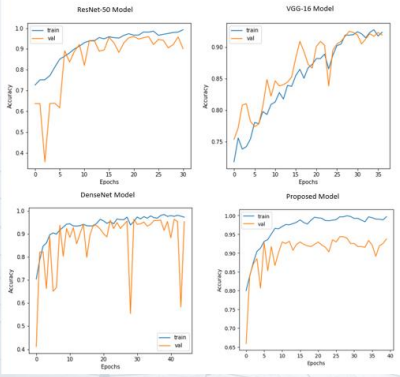


Рисунок 12 – Графіки точності моделей класифікації

Таблиця 1 – Результати моделей класифікації

Модель	TP	TN	FP	FN
Запропонована ЗНМ	185	375	18	12
ResNet-50	181	391	2	16
VGG-16	189	379	14	8
DenseNet-50	192	389	4	5

Таблиця 2 – Результати метрик

Модель	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)
Запропонована ЗНМ	94	91	93	92
ResNet-50	97	98	92	95
VGG-16	96	93	95	94
DenseNet-201	98	98	97	98

Аналіз результатів моделей сегментації

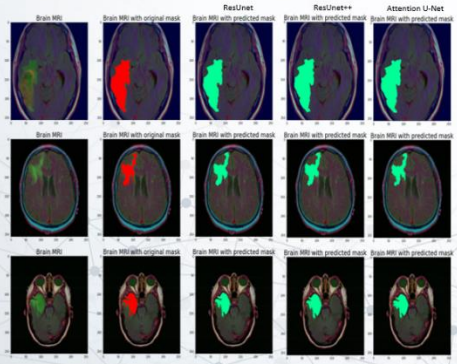


Рисунок 13 – Порівняння результатів сегментації

Таблиця 3 – Результати метрик

Модель	Tversky (%)	Dice (%)	Jaccard (%)
ResUnet	95	94	89
ResUnet++	95	93	87
Attention U-Net	94	93	87

Програмний продукт

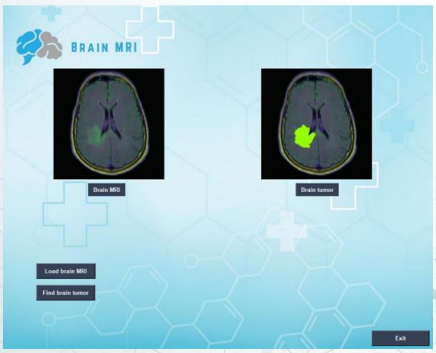


Рисунок 7 – Результат сегментації пухлини

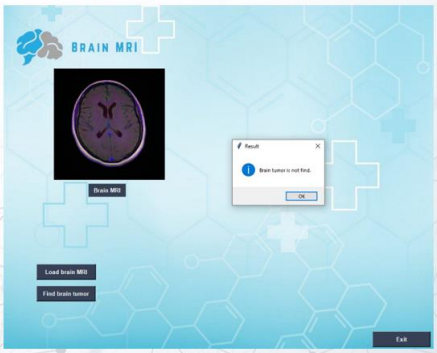


Рисунок 8 – Результат роботи програми, якщо пухлину не знайдено

Висновок

Розроблено програмний продукт на мові програмування Python для автоматичного розпізнавання пухлин на знімках МРТ головного мозку та їхньої подальшої сегментації. Це дозволить лікарям швидко та якісно отримати результати МРТ, що допоможе у виборі стратегії лікування пацієнтів. Використання глибоких нейронних мереж дозволяє ефективно визначати наявність пухлини на знімках МРТ головного мозку.

Найбільш точні результати класифікації було отримано з використанням моделі DenseNet-201, яка має точність класифікації 95%. Для сегментації пухлин головного мозку модель ResUnet має найбільшу точність, яка складає 95%.

Отримані результати дозволяють зробити висновок про доцільність використання розглянутих моделей для аналізу знімків МРТ головного мозку.

Дякую за увагу!