МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

РОЗПІЗНАВАННЯ ТА КЛАСИФІКАЦІЯ ХВОРОБИ АЛЬЦГЕЙМЕРА НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

Спеціальність 122 «Комп'ютерні науки»

122 – КРМ – 601.21810224

Виконав студент 6-го курсу, групи 601
_____ О. М. Сова
«19» лютого 2024 р.

Керівник: канд. техн. наук, доцент
_____Є. В. Сіденко
«19» лютого 2024 р.

Миколаїв – 2024

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20___ р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Сові Олександру Михайловичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розпізнавання та класифікація хвороби Альцгеймера на основі методів машинного навчання».
Керівник роботи Сіденко Євген Вікторович, доцент кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «19» жовтня 2023 р. № 201
2. Строк подання студентом роботи 19 лютого 2024 р.
3. Вхідні (початкові) дані до роботи: датасет зображень знімків МРТ головного мозку.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

– дослідження теоретичних засад створення рекомендаційних систем та здійснення аналізу існуючих систем класифікації хвороби Альцгеймера;

– обґрунтування вибору інструментальних засобів розробки забезпечення яке використовує комп'ютерний зір для класифікації хвороби Альцгеймера;

– розробка та здійснення програмної реалізації програмного застосунку для класифікації хвороби Альцгеймера;

5. Перелік графічного матеріалу: презентація, рисунки, таблиці.

6. Завдання до спеціальної частини: Аналіз виробничих факторів на робочому місці та техніка безпеки.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	д-р біол. наук, Л. І. Григор'єва	
Методична частина	канд. техн. наук, доцент Є. В. Сіденко	

Керівник роботи _____канд. техн. наук, доц. Сіденко Є. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання _____Сова О. М._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__31__» _____жовтня____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи магістра

Тема: Розпізнавання та класифікація хвороби Альцгеймера на основі методів машинного навчання

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми КРМ. Подання заяви на затвердження теми КРМ	01.09.2023	10.10.2023	Виконано
2	Отримання завдання на виконання КРМ	11.10.2023	01.11.2023	Виконано
3	Складання календарного плану на період виконання КРМ	02.11.2023	10.11.2023	Виконано
4	Огляд літератури за темою дослідження	11.11.2023	26.11.2023	Виконано
5	Проходження передатестаційної практики, збір та аналіз матеріалів до КРМ	27.11.2023	23.12.2023	Виконано
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	25.12.2023	12.01.2024	Виконано
7	Опис фахової частини КРМ, зокрема дослідження публікацій щодо хвороби Альцгеймера, огляд існуючих архітектур штучних нейронних мереж для вирішення поставленої задачі, реалізація обраних технологій з аналізом отриманих результатів	13.01.2024	25.01.2024	Виконано
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2024	02.02.2024	Виконано
9	Перший попередній захист КРМ на засіданні комісії кафедри	29.01.2024	29.01.2024	Виконано
10	Корегування роботи за результатами попереднього захисту	30.01.2024	05.02.2024	Виконано
11	Доробка та остаточне оформлення КРМ	06.02.2024	11.02.2024	Виконано
12	Другий попередній захист КРМ на засіданні комісії кафедри	12.02.2024	12.02.2024	Виконано
13	Подання КРМ, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.02.2024	20.02.2024	Виконано
14	Захист КРМ перед екзаменаційною комісією (ЕК)	26.02.2024	27.02.2024	

Розробив студент _____Сова О.М._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи _____канд. техн. наук., доцент Сіденко Є. В._ _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«09» листопада 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
студента групи 601 ЧНУ ім. Петра Могили

Сови Олександра Михайловича

на тему: **“РОЗПІЗНАВАННЯ ТА КЛАСИФІКАЦІЯ ХВОРОБИ АЛЬЦГЕЙМЕРА НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ”**

Об'єктом дослідження є процес розпізнавання та класифікація стадій захворювання хвороби Альцгеймера.

Предмет дослідження – методи машинного навчання та архітектури моделей нейромереж для автоматизованої класифікації.

Мета роботи – розробка та підвищення ефективності моделей автоматично виявляти та діагностувати ознаки хвороби Альцгеймера.

Кваліфікаційна робота складається з загальної частини, та індивідуальних томів.

У вступі визначається мета роботи та етапи потрібні для її досягнення. У першому розділі проаналізовано та оглянуто методи вирішення задач класифікації хвороби Альцгеймера. У другому розділі змодельовано та досліджено архітектури для вирішення задачі класифікації. У третьому розділі було імплементовано обрані алгоритми і розроблено програмний застосунок. У висновках проводиться аналіз проведеної роботи та отриманих результатів.

У методичній частині наведений приклад власної практичної роботи.

У спеціальній частині з охорони праці описано аналіз виробничих факторів на робочому місці та техніка безпеки.

Пояснювальна записка індивідуального тому роботи складається зі вступу, *3* розділів, висновків, методичної частини та спеціальної частини з охорони праці.

Загальний обсяг роботи – 82 сторінки. Кваліфікаційна робота містить один додаток, 69 рисунків, 5 формул, 4 таблиці та посилання на 63 літературних джерел.

Ключові слова: хвороба Альцгеймера, машинне навчання, згорткова нейронна мережа, модель, класифікація.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Sova Oleksandr Mykhailovych

“RECOGNITION AND CLASSIFICATION OF ALZHEIMER'S DISEASE BASED ON MACHINE LEARNING METHODS”

Object of research - recognition and classification of Alzheimer's disease.

Subject of research – machine learning methods and architectures for automated classification of Alzheimer's disease stages on brain MRI images.

The purpose of the thesis – development and optimization of models capable of automatically detecting pathological changes in the structure and functions of the brain, which are characteristic of the development of Alzheimer's disease.

Thesis consist of general part, and individual volumes.

The introduction defines the purpose of the work and the steps required to achieve it. In the first section, the methods of solving problems of classification of Alzheimer's diseases are analyzed and reviewed. In the second section, architectures for solving the recognition problem are modeled and investigated. In the third section, the selected algorithms were implemented and the software was developed. In the conclusions, an analysis of the work carried out and the results obtained is carried out.

In the methodological part an example of one's own practical work is given.

The special part on labor protection describes the analysis of production factors at the workplace and safety techniques.

Explanatory note of this individual volume of thesis consists of introduction, *3* sections, conclusions, methodological and special part on labor protection.

The total volume of work is 82 pages. The master's thesis contains one appendix, 69 figures, 5 formulas, 4 tables and references to 63 literary sources.

Keywords: Alzheimer Disease, Machine Learning, Convolutional Neural Network, model, classification.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ЗНМ - згорткова нейронна мережа.

МРТ - магнітно-резонансна томографія.

CNN - convolutional neural network, згорткова нейронна мережа.

CV - computer vision, комп’ютерний зір.

GPU - graphical processing unit, графічний процесор.

KNN – k-nearest neighbor, метод k-найближчих сусідів

ReLU - rectified linear unit, випрямляч.

ResNet - residual network, залишкова мережа.

SVM – support vector machine, метод опорних векторів.

VGG – Visual Geometry Group.

ВСТУП

У сучасному інформаційному суспільстві, в якому машинне навчання і штучний інтелект набувають все більшого значення, великою проблемою стає виявлення та ефективне лікування нейродегенеративних захворювань. Серед них особливою тяжкістю накладається хвороба Альцгеймера, яка впливає на когнітивні функції і психічний стан людини, погіршуючи якість її життя та соціальні взаємини і порушуючи звичайний спосіб життя.

З огляду на складність та ступінь поширеності хвороби Альцгеймера, розробка ефективних методів ранньої діагностики та класифікації стає вельми актуальною задачею в медичній галузі. В останні роки машинне навчання виявляється перспективним інструментом для розв'язання подібних завдань завдяки здатності аналізувати величезні об'єми даних та виявляти складні залежності між ними.

Ця кваліфікаційна робота присвячена дослідженню можливостей застосування методів машинного навчання для розпізнавання та класифікації хвороби Альцгеймера на основі аналізу різноманітних клінічних, біологічних та образних даних.

Об'єктом дослідження є процес розпізнавання та класифікація стадій захворювання хвороби Альцгеймера.

Предмет дослідження – методи машинного навчання та архітектури моделей нейромереж для автоматизованої класифікації.

Мета роботи – розробка та підвищення ефективності моделей автоматично виявляти та діагностувати ознаки хвороби Альцгеймера.

Дослідження такого роду має потенціал допомогти у вдосконаленні ранньої діагностики хвороби, забезпечуючи лікарям та дослідникам засоби для швидкого та точного виявлення патологічних змін, що є важливим етапом у розробці ефективних стратегій лікування та управління хворобою Альцгеймера.

Для досягнення поставленої мети та створення програмного додатка потрібно виконати наступні етапи:

– постановка задачі;
– вивчення теоретичної бази;
– аналіз вхідних даних;
– вибір технологій;
– імплементація обраних алгоритмів та методів класифікації;
– порівняльний аналіз результатів і вибір моделі та набору методів обробки для досягнення найвищого показника точності;
– підведення підсумків.

1 АНАЛІЗ ТА ОГЛЯД МЕТОДІВ ВИРІШЕННЯ ЗАДАЧ КЛАСИФІКАЦІЇ ХВОРОБИ АЛЬЦГЕЙМЕРА

1.1 Хвороба Альцгеймера та задача класифікації

Хвороба Альцгеймера є поширеною причиною деменції, яка є прогресуючим розладом мозку, який може спричинити пошкодження клітин головного мозку, і пацієнт, уражений цією хворобою, може погіршити свою поведінку, мислення та навички соціальної діяльності [1].

Відповідно до попереднього дослідження було підраховано, що в 2050 році 1 з 85 людей може постраждати від цієї хвороби [2]. Важливо діагностувати та проводити профілактику на ранніх стадіях пацієнтів з хворобою Альцгеймера. Існує кілька інструментів для виявлення та усунення цієї деменції за допомогою МРТ, ПЕТ (позитронно-емісійної томографії) та КТ (комп’ютерної томографії), але діагностика пацієнтів з хворобою Альцгеймера за допомогою МРТ зазнає найбільшого впливу та популярний метод нейровізуалізації.

Основні ознаки хвороби Альцгеймера включають:

– втрата пам'яті: це може виявитися у втраті короткочасної пам'яті та забудові навіть найбільш звичних речей або подій;

– порушення мислення та розуміння: пацієнти можуть виявляти труднощі з розв'язанням простих завдань або здійсненням звичних розрахунків;

– порушення мови: вони можуть мати труднощі з пошуком слів або формулюванням речень;

– зміни у поведінці та особистості: це може включати агресивність, роздратованість, апатію або відчуття втрати інтересу до попередніх захоплень;

– порушення спроможності до самообслуговування: пацієнти можуть потребувати допомоги в повсякденних діях, таких як одягання, годування чи особиста гігієна.

Діагноз хвороби Альцгеймера встановлюється на підставі клінічних симптомів, результатів обстежень та виключення інших можливих причин деменції. Хоча лікування хвороби Альцгеймера поки не існує, існують підходи до

діагностики, управління симптомами, сповільнення прогресу та прогнозування прогресу захворювання. Серед цих підходів методи машинного навчання мають значний потенціал покращити якість діагностики, ідентифікації ризикових груп, персоналізації лікування та прогнозування прогресу захворювання.

Рисунок 1.1 – Здоровий мозок (ліворуч) і пізня стадія хвороби Альцгеймера (праворуч)

1.2 Класичні алгоритми машинного навчання для задач класифікації

Сфера штучного інтелекту почала свій розвиток ще з кінця 1950-х років і в наш час налічує велику кількість різноманітних алгоритмів і методів для розв'язання завдань комп'ютерного зору. Часто розробники використовували прості концепції мозкових процесів людини, таких як передача нейроімпульсів (яка лягла в основу перцептрона), і поєднували їх із статистичними підходами. Нижче подано деякі із таких методів, що застосовуються для вирішення завдань класифікації.

1.2.1 Логістична регресія

Метод логістичної регресії є одним із класичних алгоритмів машинного навчання, який може бути застосований до задачі класифікації хвороби Альцгеймера на основі знімків МРТ головного мозку. Основна ідея логістичної регресії полягає в тому, що вона моделює ймовірність належності прикладу до певного класу шляхом застосування логістичної функції до лінійної комбінації вхідних ознак.

В контексті класифікації хвороби Альцгеймера, перш за все, потрібно підготувати дані, включаючи вектор ознак, який буде включати в себе відомості, отримані з зображень МРТ, такі як форма, текстура, інтенсивність та інші характеристики. Потім ці дані можна буде розділити на тренувальну та тестову вибірки для оцінки ефективності моделі.

Далі, застосовуючи метод логістичної регресії, модель буде навчатися за допомогою тренувальних даних з метою мінімізації функції втрат і максимізації правильної класифікації. Після завершення тренування модель буде готовою до класифікації нових зображень МРТ.

Логістична регресія представляє собою метод статистичної регресії, який використовується в ситуаціях, коли залежна змінна має бінарний характер, тобто може приймати лише два значення (0 або 1). У випадках, коли введено порогове значення, цей метод може знайти застосування в задачах класифікації. Функція логістичної регресії має такий вигляд:

$$\Lambda(x) = \frac{1}{1 + e^{-x}} \qquad (1.1)$$

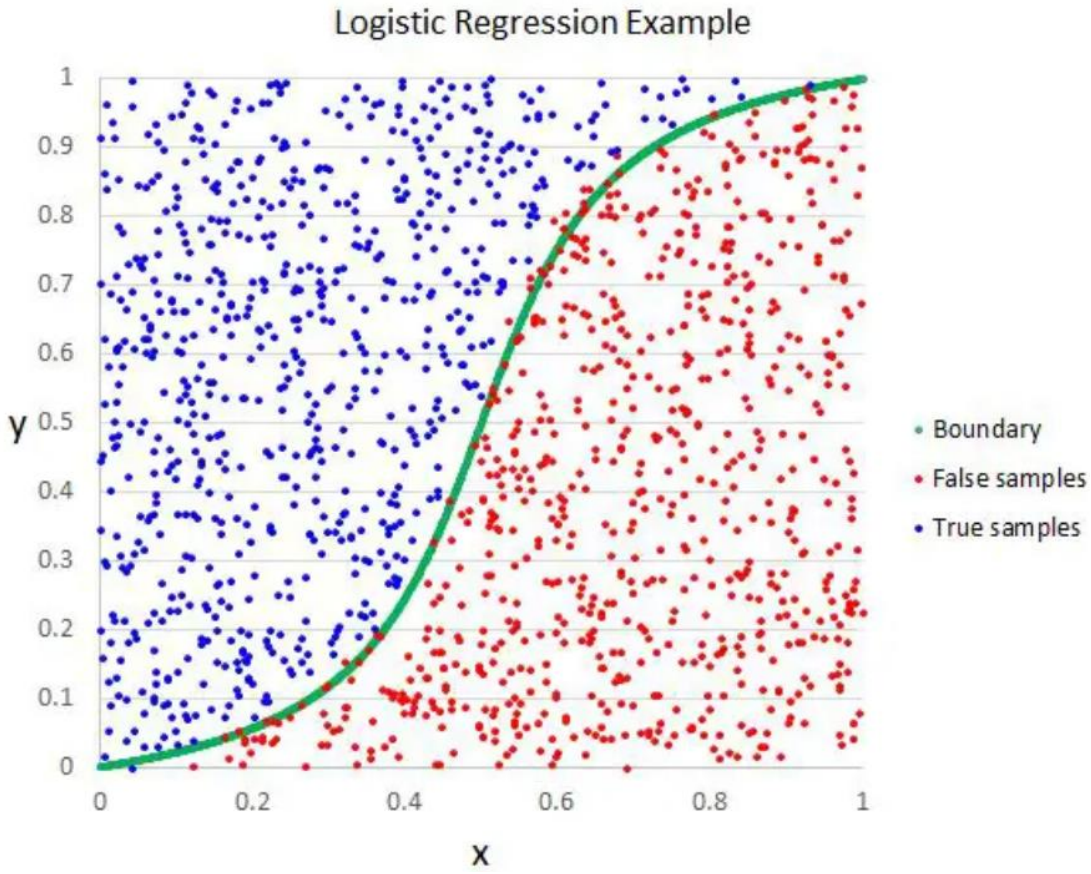

Рисунок 1.2 – Приклад роботи логістичної регресії для класифікації

Переваги даного алгоритму: логістична регресія розроблена саме для задачі класифікації і є найбільш корисною для розуміння впливу декількох незалежних змінних на одну вихідну змінну.

Недоліки: працює лише тоді, коли прогнозована змінна є двійковою, припускає, що всі предиктори незалежні один від одного та припускає, що дані не містять пропущених значень [3].

1.2.2 Метод k-найближчих сусідів (KNN)

Метод k-найближчих сусідів (k-NN) - це простий алгоритм машинного навчання, що використовується для класифікації та регресії. В контексті класифікації хвороби Альцгеймера, k-NN може бути використаний для прогнозування класу (наприклад, стадії захворювання) на основі ознак, отриманих з зображень МРТ головного мозку.

Основна ідея методу k-NN полягає у визначенні класу нового зразка шляхом найближчого у просторі ознак сусідства. Для цього визначається кількість найближчих сусідів (k), і для нового зразка обчислюється його відстань до всіх інших зразків у навчальному наборі. Потім вибирається k найближчих сусідів, і клас нового зразка призначається як клас, який найчастіше зустрічається серед цих сусідів [4].

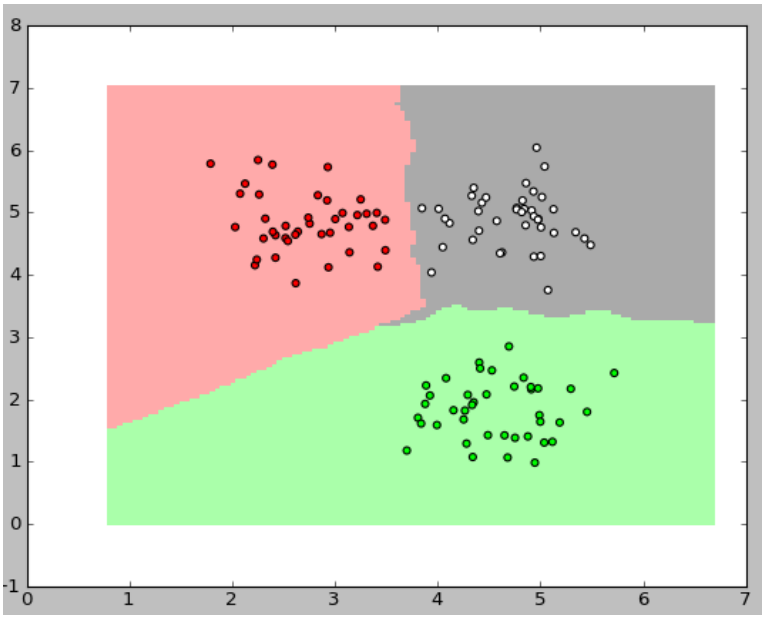

Рисунок 1.3 – Приклад класифікації методом k-NN (3 класи, k = 3)

Хоча метод k-NN є простим у реалізації та зрозумілим, він може бути обмеженим у використанні для великих обсягів даних через свою високу обчислювальну складність. Крім того, важливо правильно вибрати значення k, оскільки невірний вибір може призвести до перенавчання або недонавчання моделі.

1.2.3 Метод опорних векторів (SVM)

Метод опорних векторів (SVM) є одним з найпоширеніших методів машинного навчання, використовуваних для класифікації даних. Даний метод є особливо ефективним у випадках, коли маємо невелику кількість признаків та велику кількість прикладів.

У контексті класифікації хвороби Альцгеймера на знімках МРТ головного мозку, метод SVM може бути використаний для розподілу пацієнтів на кілька класів на основі ознак, отриманих з зображень. Наприклад, можна використовувати певні характеристики зображень, отримані під час обробки та аналізу, які вказують на наявність чи відсутність певних ознак або патологій, пов'язаних з хворобою Альцгеймера.

Метод SVM спробує знайти гіперплощину в просторі ознак, яка найкращим чином розділить дані на різні класи. Ця гіперплощина визначається таким чином, щоб максимізувати відстань між найближчими точками кожного класу, відомими як опорні вектори. Потім, для нових прикладів, метод SVM може визначити, до якого класу вони належать, шляхом перевірки їх розташування відносно цієї гіперплощини [5].

Одним з важливих переваг методу SVM є його здатність до роботи з даними високої розмірності та здатність до управління перенавчанням, що робить його відмінним вибором для задач класифікації на основі зображень, таких як класифікація хвороби Альцгеймера. Однак, ефективність методу SVM може залежати від правильного вибору параметрів та якості ознак, що використовуються для класифікації.

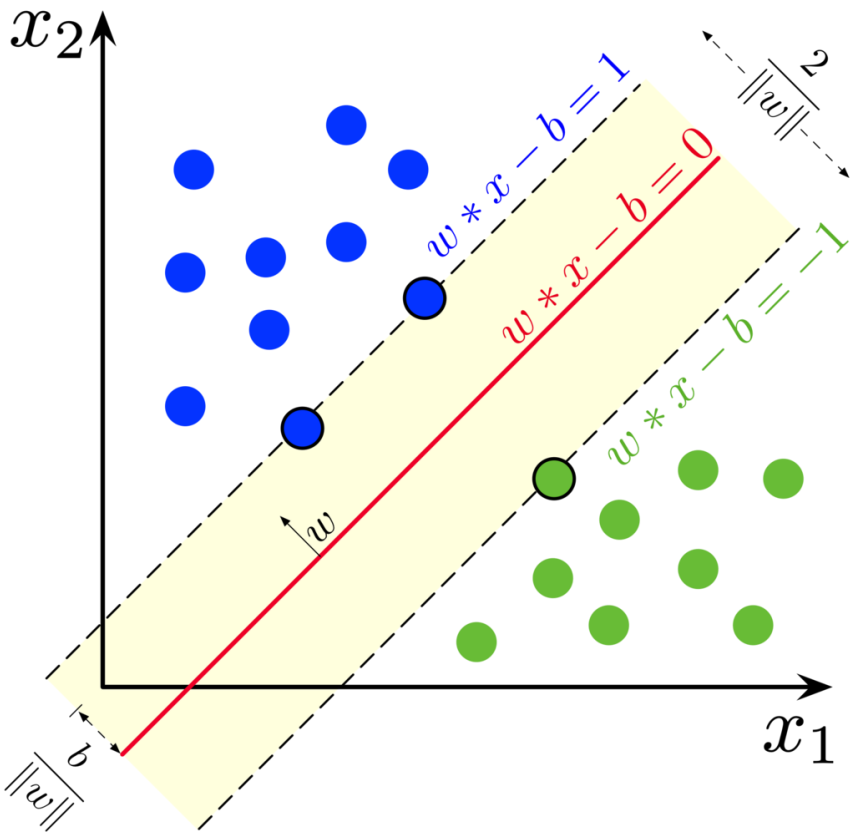

Рисунок 1.4 – Максимально розділова гіперплощина та межі для SVM, натренованої зразками з двох класів. Зразки на межах називаються опорними векторами

1.3 «Глибинне» навчання для задач класифікації

Глибинне навчання (Deep Learning) представляє собою галузь машинного навчання в сфері штучного інтелекту, яка базується на ряді методів, спрямованих на створення високорівневих абстракцій у візуальних даних. Дослідження в цьому напрямку спрямовані на розробку кращих представлень та моделей для автоматичного вивчення цих представлень з великим обсягом даних, які не мають професійної розмітки.

Алгоритми глибинного навчання базуються на певних принципах, в яких спостережувані дані виникають внаслідок взаємодії певних чинників, сформованих на глибинному рівні мережі. Глибинне навчання припускає, що ці рівні чинників відповідають різним рівням абстракції.

Методологія глибинного навчання використовує цю ідею факторів: вона передбачає формування та навчання більш абстрактних форм на вищих рівнях з використанням понять з нижчих рівнів. Ці архітектури часто конструюються з використанням жадібних покрокових алгоритмів [6].

Успіхи в області глибинного навчання привели до значного прогресу в оптимізації завдань оптичного розпізнавання образів. В даний час архітектури глибинного навчання стали стандартними в застосуванні до завдань комп'ютерного зору, таких як класифікація.

Комп'ютерний зір (Computer Vision, CV) є передовою технологією, яка дозволяє машинам виявляти, відстежувати та класифікувати різноманітні об'єкти на зображеннях. Ця галузь входить до технології і теорії розробки штучних систем, що обробляють візуальну інформацію [7].

Важливо відзначити, що немає єдиного стандарту для вирішення завдань комп'ютерного зору. Замість цього існує різноманітність методів для вирішення різних завдань, де ефективність методів часто залежить від конкретної задачі.

Однією з ключових областей застосування класифікації є біомедицина, де аналіз зображень дозволяє вимірювати розміри органів, вивчати структуру мозку та вплив медичних засобів. Також, класифікація може бути використана для виявлення пухлин на зображеннях томографії.

Класифікація у сфері комп'ютерного зору — це процес призначення об'єктів чи зображень до певних категорій чи класів на основі їхніх характеристик.

Процес класифікації визначається розподілом об'єктів на категорії відповідно до їхніх характеристик. Його мета - спростити або змінити представлення зображення для полегшення його аналізу. Результатом класифікації є набір категорій, які відображають різні об'єкти на зображенні, наприклад, за їхньою формою чи структурою. Класифіковані об'єкти можуть відрізнятися за різними факторами.

У випадку розглядуваної задачі, прикладом класифікації є визначення стадії хвороби Альцгеймера за допомогою зображення зрізу головного мозку. Для цього були використані як універсальні алгоритми класифікації, так і підходи з класичних

методів машинного навчання. З огляду на відсутність єдиного рішення для класифікації, часто використовують комбінації методологій та додаткові знання з предметної області для досягнення ефективних результатів.

1.3.1 Згорткові нейромережі

Згорткові нейронні мережі (ЗНМ або англ. Convolutional Neural Networks, CNN) представляють собою категорію глибоких штучних нейронних мереж, які з успіхом використовуються для обробки та аналізу візуальних зображень. Вони є варіацією багатошарових перцептронів, розробленою з урахуванням необхідності мінімізації обсягу попередньої обробки [8].

У порівнянні з іншими алгоритмами класифікації зображень, ЗНМ використовують обмежену кількість попередньої обробки. Під час тренування ці системи автоматично визначають ознаки та закономірності у зображеннях без значного впливу людського фактору, що відрізняє їх від класичних систем машинного навчання, де розробка ознак зазвичай залежить від людини-розробника.

Перед появою ЗНМ для ідентифікації об'єктів на зображеннях використовувалися трудомісткі та ручні методи вилучення ознак. Однак ЗНМ тепер надають більш масштабований підхід до завдань класифікації та розпізнавання об'єктів, використовуючи принципи лінійної алгебри, зокрема, множення матриць, для виявлення шаблонів у зображенні. Важливо відзначити, що вони можуть виявитися вимогливими до обчислень та потребувати використання графічних процесорів (GPU) для ефективного навчання моделей. ЗНМ також відрізняються високою продуктивністю у вирішенні завдань з обробки зображень, мовлення та звукового сигналу [9].

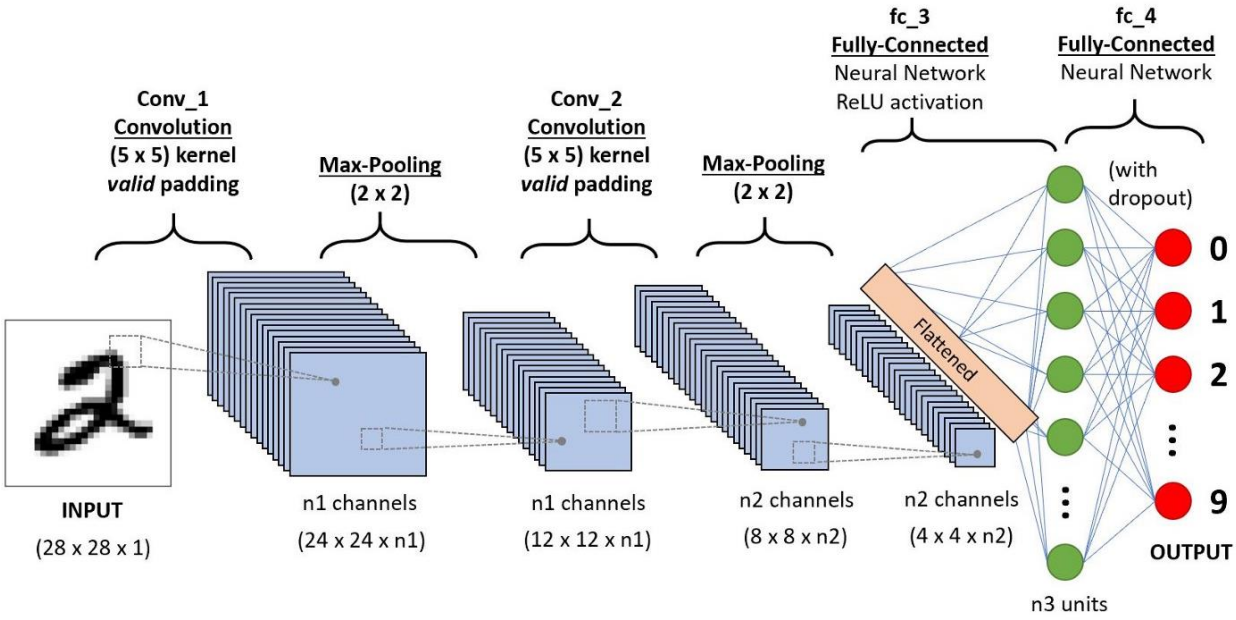

Рисунок 1.5 – Приклад структури ЗНМ для класифікації рукописних цифр

Для пояснення роботи даного типу моделей слід детально пояснити основні будівельні блоки ЗНМ.

1.3.1.1 Згортковий шар

Згортковий шар є фундаментальним блоком згорткових нейронних мереж (ЗНМ), де відбувається основна частина обчислень. Для цього потрібні різні компоненти, такі як вхідні дані, фільтр та карта ознак [10]. Наприклад, якщо вхідним є кольорове зображення, складене з матриці пікселів у тривимірному просторі, що представляє висоту, ширину та глибину (RGB), то вхідні дані матимуть три виміри. Детектор ознак, відомий як фільтр чи ядро, переміщується по зображенню, перевіряючи наявність ознак. Цей процес називається згорткою.

Фільтр, або детектор ознак, представляє собою двовимірний (2D) масив ваг, що представляє частину зображення. Хоча розміри фільтрів можуть відрізнятися, зазвичай вони мають розмір матриці 3x3, визначаючи розмір рецептивного поля. Фільтр застосовується до області зображення, обчислюючи скалярний добуток між вхідними пікселями та фільтром. Після цього фільтр зміщується на один крок, повторюючи процес до охоплення усього зображення. Результат, який отримано з

послідовності скалярних добутків, відомий як карта ознак, карта активацій або згорнута функція.

Важливо відзначити, що ваги у детекторі ознак залишаються незмінними під час руху по зображенню, це відомо як спільне використання параметрів. Певні параметри, такі як значення ваги, коригуються під час навчання за допомогою процесу зворотного поширення та градієнтного спуску. Троє гіперпараметрів, які визначають розмір виводу, повинні бути встановлені перед початком навчання нейронної мережі.

1. **Кількість фільтрів** впливає на глибину виводу, при цьому кожен фільтр створює власну карту об'єктів. Наприклад, якщо застосувати три різні фільтри, отримаємо три різні карти об'єктів, що формує глибину на третьому рівні.
2. **Крок** визначає відстань, на яку ядро згортки зсувається по вхідній матриці. Хоча значення кроку рідко буває більше двох, більший крок призводить до меншого виводу.
3. **Нульове заповнення** часто використовується, коли фільтри не відповідають розмірам вхідного зображення. Воно обнуляє всі елементи, які виходять за межі вхідної матриці, створюючи вивід того ж або більшого розміру. Три типи заповнення включають дійсне (або відсутнє) заповнення, ідентичне заповнення, яке забезпечує однаковий розмір виводу, та повне заповнення, яке збільшує розмір виводу, додаючи нулі до меж вхідних даних.

Після кожної операції згортки ЗНМ використовує функцію активації ReLU для нелінійного перетворення, що додає нелінійність до моделі. Якщо після першого шару згортки додається другий, структура ЗНМ може стати ієрархічною, дозволяючи пізнім рівням бачити пікселі в сприйнятливих полях попередніх шарів. Наприклад, при аналізі зображення велосипеда, окремі частини велосипеда можуть служити шаблонами на нижньому рівні, а їх комбінація створює шаблон вищого рівня, утворюючи ієрархію ознак у ЗНМ (див. рис. 1.6).

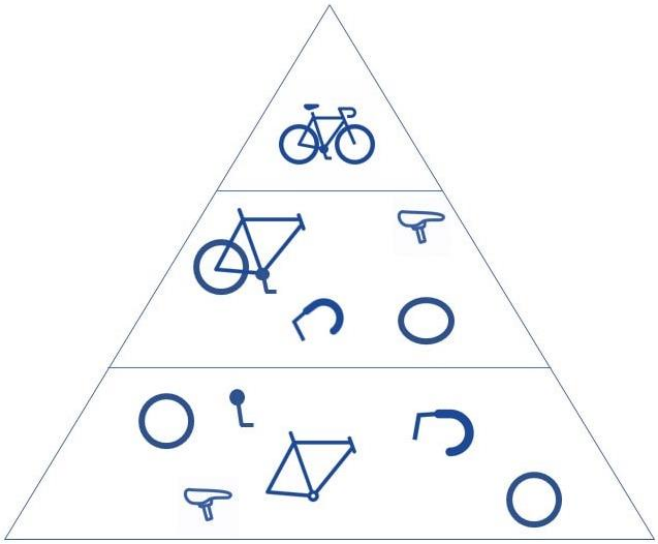

Рисунок 1.6 – Діаграма ієрархії ознак у згорткових нейронних мережах

У кінцевому підсумку, згортковий шар перетворює вхідне зображення в числові значення, що дозволяє нейронній мережі аналізувати та витягувати відповідні шаблони. Розглянемо приклад операції згортки, яка показана на рис. 1.7. У цьому контексті "Input" вказує на масив вхідних даних, таких як зображення (наприклад, бінарне зображення розміром 5x5 пікселів). Сам процес згортки здійснюється за допомогою фільтра (позначеного як "Filter/Kernel" на зображенні). Розміри та значення фільтра зазвичай визначаються розробником згорткової нейронної мережі.

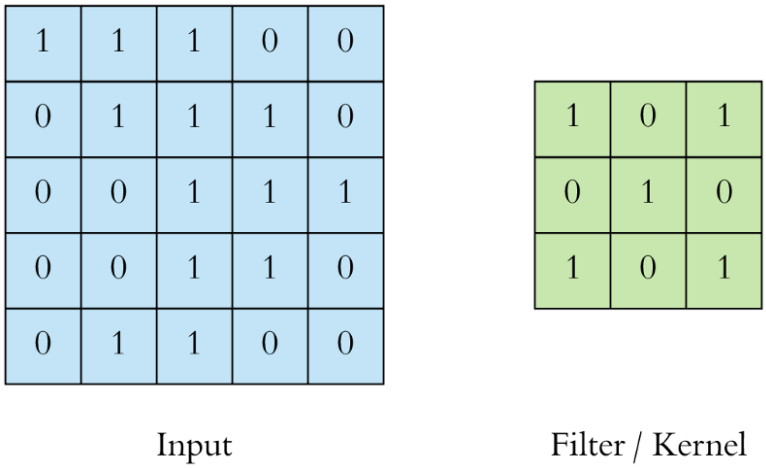

Рисунок 1.7 – Приклад вхідних даних у процесі згортки

Під час здійснення процесу згортки фільтр "накладається" на вихідне зображення. Цей процес ілюструється на рис. 1.8. Під час накладання відповідні значення з обох масивів множаться, і отримані результати сумуються. Для комірки із значенням 4 на зображенні відповідна область на вхідному масиві отримує назву його рецептивного поля, оскільки значення у цій області впливають на результат у відповідній комірці.

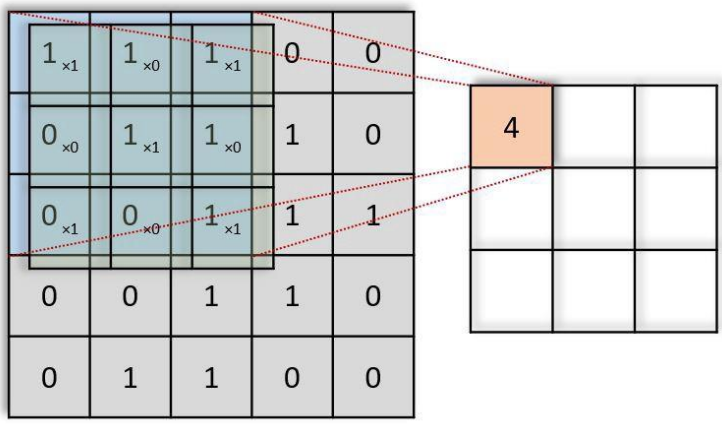

Рисунок 1.8 – Перша ітерація процесу згортки

Після цього, фільтр розпочинає переміщення по зображенню із визначеним кроком (у цьому випадку крок становить 1 піксель) та формує вихідний шар даних. Розмірність цього вихідного шару менша за розмірність вхідних даних. Наприклад, для зображення із розміром 5x5, фільтра 3x3 та кроку 1 піксель при згортці отримується вихідний масив розміром 3x3. На рис. 1.9 червоним кольором відзначено результат - вихідний масив, який є проміжним шаром у загальній структурі згорткової нейронної мережі.

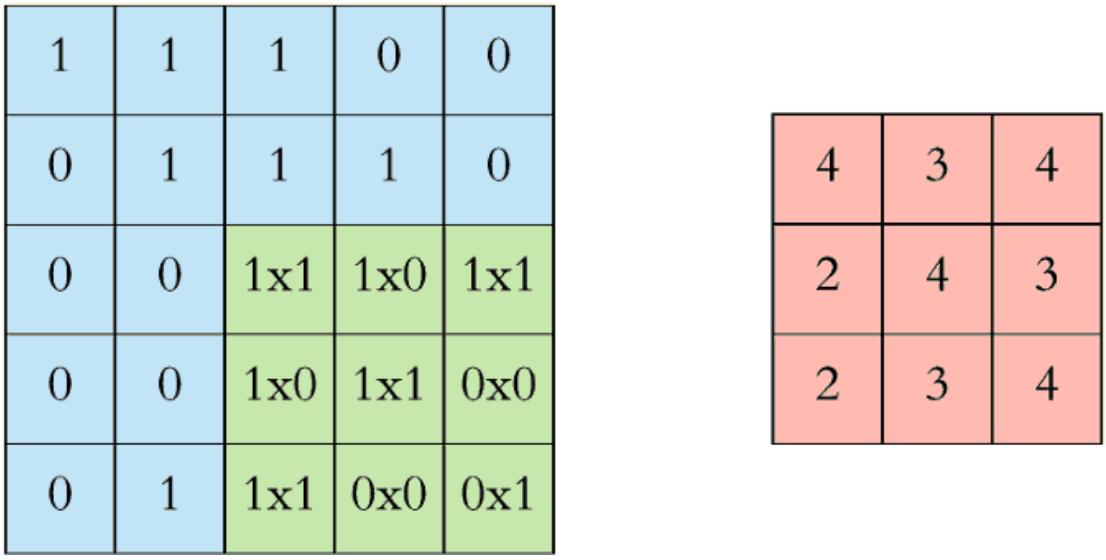

Рисунок 1.9 – Остання ітерація процесу згортки

1.3.1.2 Агрегувальний шар

Після проведення операції згортки, отримана карта ознак проходить через процес агрегування (пулінгу). Під час агрегування дані стискаються та зменшуються в розмірі, виділяючи ключові ознаки з точки зору моделі. Агрегування також є ефективним методом уникнення перенавчання моделі [11]. Під час цього процесу визначається розмір та крок фільтру, схожого на фільтр згортки. Однак під час агрегування фільтр виділяє лише області рецептивних полів для значень результуючого масиву. Існують різні методики агрегування карти ознак. Найрозповсюдженішим є макс-пулінг**.** Приклад його роботи зображений на рис. 1.10.

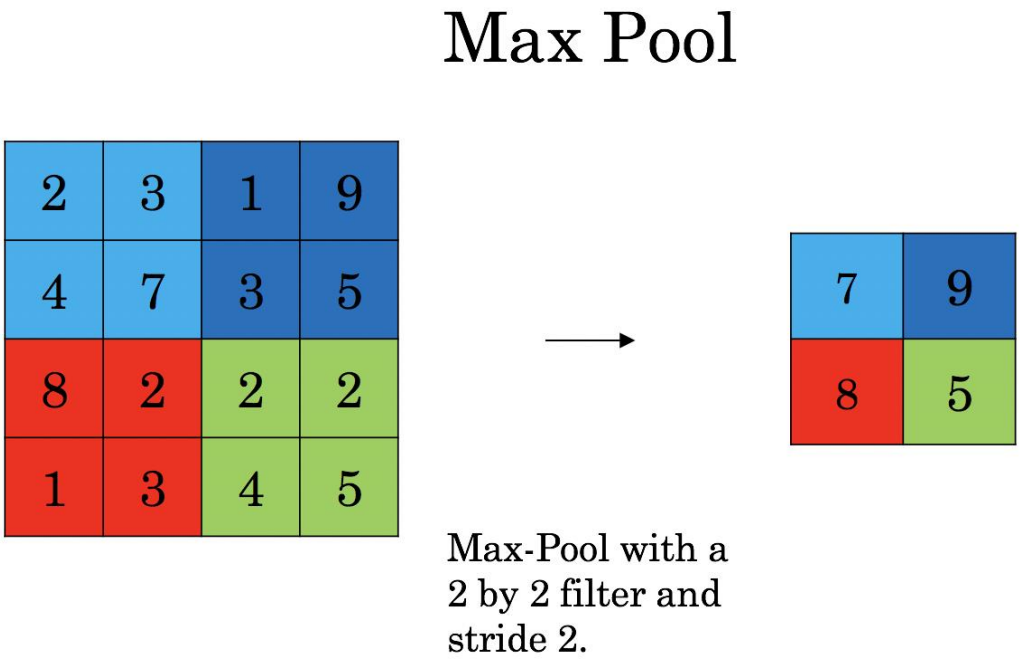

Рисунок 1.10 – Операція макс-пулінгу

У даному випадку був обраний фільтр розміром 2x2 та кроком 2. Макс-пулінгова операція вибирає найбільше значення з рецептивного поля для кожної комірки у вихідній матриці. Цей підхід спрямований на збереження важливих характеристик у зображенні та виключення непотрібної інформації, особливо коли модель опрацьовує великі зображення [12].

1.3.1.3 Повноз'єднаний шар

Як вже було вказано, значення пікселів на вхідному зображенні не мають прямого зв'язку з вихідним шаром у частково з'єднаних шарах. У повнозв'язному шарі кожен вузол вихідного шару безпосередньо з'єднується з вузлом попереднього шару.

Цей шар виконує завдання класифікації на основі ознак, які витягнуті через попередні шари та їх різні фільтри. Тоді, коли рівні згорток та об'єднань зазвичай використовують функцію активації ReLU (англ. *rectified linear unit*), рівні повнозв'язних шарів, як правило, використовують функцію активації softmax для коректної класифікації вхідних даних та створення ймовірностей від 0 до 1 [13].

1.4 Постановка задачі

Для досягнення поставленої мети потрібно виконати наступне:

– дослідження теоретичної бази та аналіз існуючих досліджень на пов’язану тем;

– вивчення та вибір доступних бібліотек машинного і глибинного навчання;

– тестування різних алгоритмів машинного навчання для класифікації стадії хвороби Альцгеймера на знімках МРТ головного мозку;

– оптимізація класифікації при покращені роботи існуючих архітектур, або ж запровадити нові використання методів для збільшення точності.

Висновки до розділу 1

У цьому розділі дано загальний опис хворобі Альцгеймера та застосуванню комп’ютерного зору в біомедицині. Також були досліджені види машинного та глибинного навчання та проаналізовані існуючі дослідження по цій темі. Також були сформовані потрібні для проведення дослідження кроки: вивчення та вибір доступних бібліотек для роботи в сфері машинного та глибинного навчання, тестування алгоритмів машинного та глибинного навчання, оптимізація класифікації хвороби Альцгеймера при покращені роботи існуючих архітектур.

2 МОДЕЛЮВАННЯ ТА ДОСЛІЖЕННЯ АРХІТЕКТУР ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ

2.1 Аналіз обраної задачі і вхідних даних

Набір даних доступний на платформі Kaggle та призначений для дослідження хвороби Альцгеймера через зображення мозку, отримані за допомогою технології обстеження МРТ. Основні характеристики набору даних:

– розмір: набір даних містить 6400 (5121 тренувальних і 1279 тестових) зображень;

– типи зображень: зображення представлені у форматі JPG розміром 176x208 пікселів. Декілька прикладів зображені на рис. 2.1;

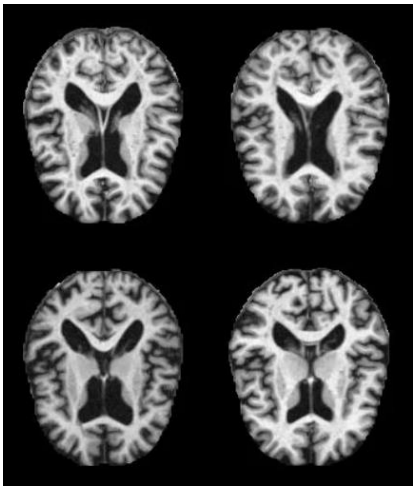

Рисунок 2.1 – Знімки МРТ головного мозку

– класи: датасет містить 4 класи: "Non Demented" (без деменції), "Very Mild Demented" (дуже слабка деменція), "Mild Demented" (легка деменція) та "Moderate Demented" (помірна деменція). Розподіл даних по класам наведений на рис. 2.2.

Data type	Training set	Test set
Non Demented	2560	640
Very Mild Dementia	1792	448
Mild Dementia	717	179
Moderate Dementia	52	12

Рисунок 2.2 – Розподіл даних у вхідному датасеті

Цей набір даних є корисним ресурсом для дослідження та розробки моделей машинного навчання для класифікації хвороби Альцгеймера на основі зображень МРТ головного мозку. Він дозволяє виконувати завдання класифікації з високою точністю та надійністю, а також досліджувати залежність між характеристиками зображень та станом хвороби.

2.2 Обрання алгоритмів і методів для вирішення задачі класифікації

Спираючись на попередній розділ та аналіз вхідних даних виберемо алгоритми та методи для вирішення задачі класифікації хвороби Альцгеймера.

Протягом теоретичних досліджень дуже явно виокремлюються алгоритми глибинного навчання. Для вирішення задачі класифікації хвороби Альцгеймера будуть застосовані різні варіації архітектур ЗНМ: VGG, ResNet, DenseNet, EfficientNet.

2.2.1 VGG

Повна назва VGG — Visual Geometry Group, яка належить Департаменту науки та техніки Оксфордського університету. Він випустив серію згорткових мережевих моделей, починаючи з VGG, які можна застосовувати для розпізнавання облич і класифікації зображень, від VGG16 до VGG19. Початкова мета дослідження VGG щодо глибини згорткових мереж полягає в тому, щоб зрозуміти, як глибина згорткових мереж впливає на точність і точність класифікації та розпізнавання великомасштабних зображень [14]. Щоб збільшити кількість мережевих рівнів і

уникнути занадто великої кількості параметрів, на всіх рівнях використовується невелике ядро згортки 3x3. У VGGNet кожен рівень згортки містить від 2 до 4 операцій згортки. Розмір кроку згортки — 1, ядра об’єднання — 2x2, а розмір кроку — 2. Найбільш очевидним удосконаленням VGGNet є зменшення розміру ядра згортки та збільшення кількості шарів згортки [15].

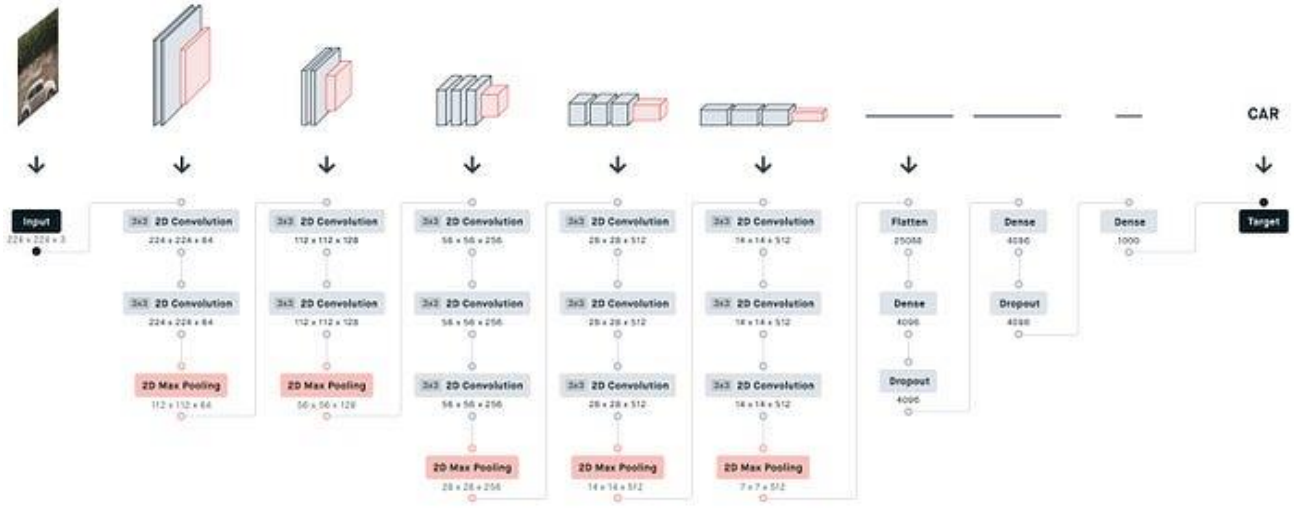

Рис. 2.2 – Архітектура VGG19

2.2.2 ResNet

ResNet, або залишкова нейромережа, представляє собою одну з модифікацій згорткових нейромереж. Коли мережа стає глибше, виникає проблема: зі збільшенням її глибини точність спочатку підвищується, але потім стрімко зменшується. Це погіршення точності вказує на особливості оптимізації деяких мереж.

Для подолання цієї проблеми введено концепцію "залишкової" структури навчання, в якій використовують з'єднання швидкого доступу. З'єднання швидкого доступу дозволяють обходити один або кілька шарів, виконуючи порівняння ідентифікаторів. Виходи цих швидкодіючих з'єднань додаються до виходів накладених шарів [16]. Приклад імплементації даного типу мереж зображений на рисунку 2.4.

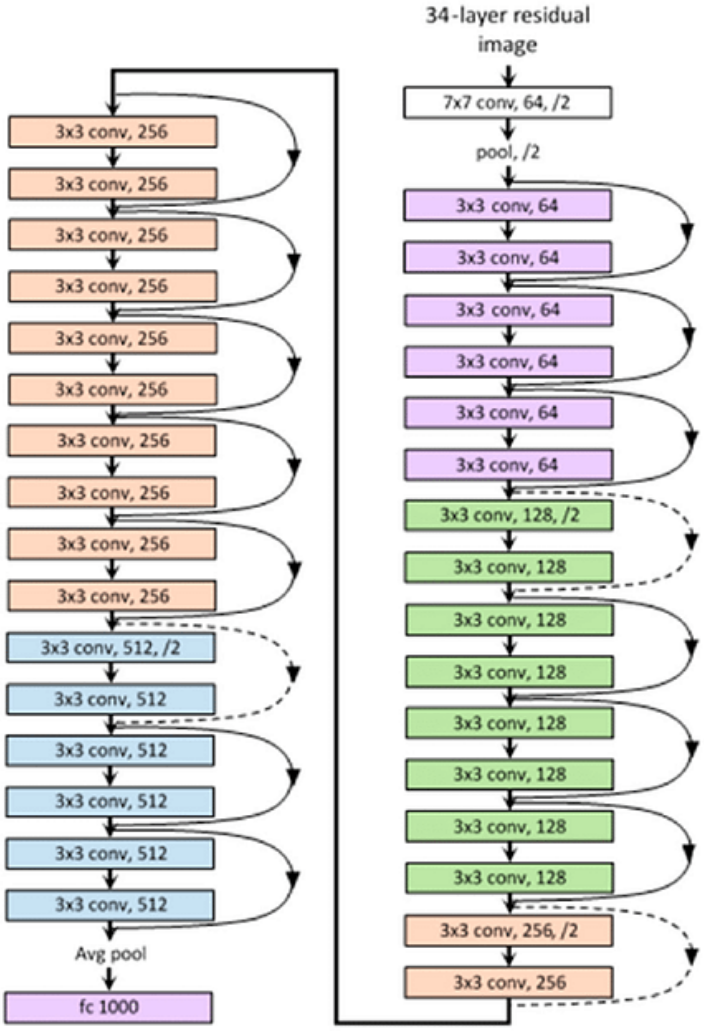

Рисунок 2.3 – Архітектура ResNet34

На зображенні ілюструється, що архітектура моделей ResNet складається з послідовних блоків з'єднань швидкого доступу, які формуються на базі основних згорткових шарів. Використовуючи ResNet, можна вирішувати різноманітні завдання, оскільки:

– ResNet виявляється відносно легким у оптимізації: у "простих" мережах (що просто об'єднують шари) помітно високий рівень навчання, коли глибина зростає;

– завдяки ResNet можна відносно просто підвищити точність за рахунок збільшення глибини, що може бути більш важким завданням для інших типів мереж.

2.2.3 DenseNet

Щільно зв’язані згорткові мережі (**DenseNet**) — це архітектура згорткової нейронної мережі із прямим зв’язком, яка пов’язує кожен рівень з усіма іншими. Це дозволяє мережі навчатися більш ефективно, повторно використовуючи функції, отже, зменшуючи кількість параметрів і покращуючи градієнтний потік під час навчання. У 2016 році Gao Huang та ін. представили архітектуру у своїй статті DenseNet «Щільні згорткові мережі».

Конструкція DenseNet базується на простому та базовому принципі: завдяки об’єднанню карт функцій усіх попередніх рівнів щільний блок дозволяє кожному шару отримувати доступ до функцій усіх попередніх рівнів. У класичних ЗНМ кожен шар має доступ лише до характеристик шару, що знаходиться безпосередньо перед ним [17].

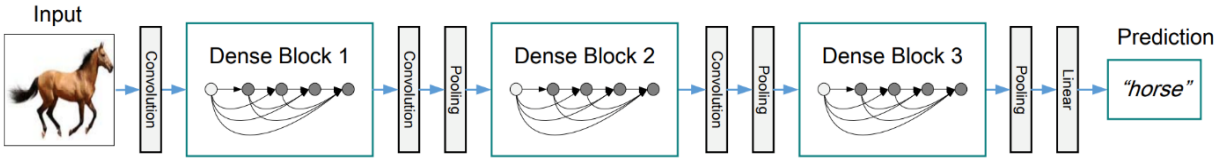

Рисунок 2.4 – Приклад DenseNet з трьома щільними блоками

Архітектура DenseNet складається з перехідних шарів і щільних блоків. Кожен згортковий шар всередині щільного блоку пов’язаний з усіма іншими шарами в блоці. Це досягається шляхом підключення виходу кожного шару до входу наступного шару, утворюючи «скорочене» посилання. Перехідні шари мінімізують розмір карт функцій у щільних блоках, що дозволяє ефективно розвивати мережу.

Класифікація зображень, розпізнавання об’єктів і семантична сегментація – це лише деякі програми комп’ютерного зору, де архітектура DenseNet, як було показано, досягає найсучаснішої продуктивності завдяки своїй здатності ефективно використовувати повторне використання функцій і зменшувати кількість параметрів [18, 19].

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Рисунок 2.5 – Опис будови архітектур DenseNet

2.2.4 EfficientNet

EfficientNet — це архітектура згорткової нейронної мережі та метод масштабування, який рівномірно масштабує всі розміри глибини/ширини/роздільної здатності за допомогою складеного коефіцієнта. На відміну від звичайної практики, яка довільно масштабує ці фактори, метод масштабування EfficientNet рівномірно масштабує ширину, глибину та роздільну здатність мережі за допомогою набору фіксованих коефіцієнтів масштабування. Наприклад, якщо ми хочемо використати в 2^N разів більше обчислювальних ресурсів, тоді ми можемо просто збільшити глибину мережі на α^N, ширину на β^N і розмір зображення на γ^N, де α, β, γ — постійні коефіцієнти, визначені пошуком за малою сіткою на оригінальній маленькій моделі. EfficientNet використовує складений коефіцієнт ϕ для рівномірного принципового масштабування ширини, глибини та роздільної здатності мережі [20, 21].

Метод складного масштабування виправдовується інтуїцією, що якщо вхідне зображення більше, то мережі потрібно більше шарів, щоб збільшити сприйнятливе

поле, і більше каналів, щоб захопити більш дрібнозернисті візерунки на більшому зображенні [22].

EfficientNet також добре передає та досягає найсучаснішої точності на CIFAR-100 (91,7%), Flowers (98,8%) та інших наборах даних для навчання передачі з на порядок меншою кількістю параметрів [23].

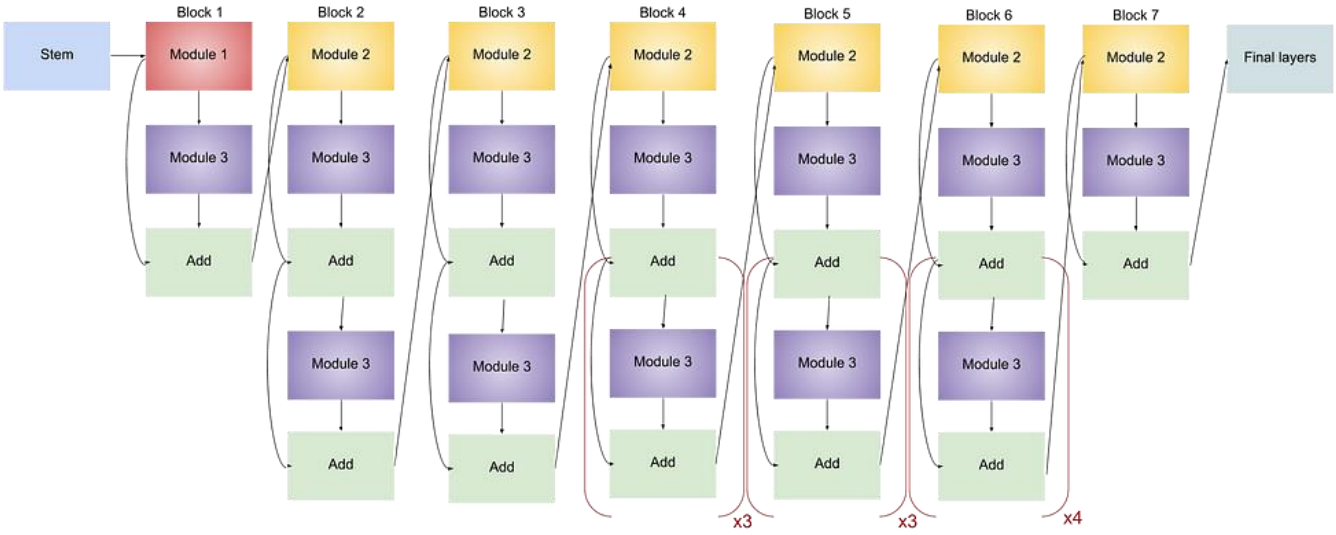

Рисунок 2.6 – Архітектура EfficientNetB3

2.3 Обрання технологій для розробки рішень обраної задачі

Для вирішення обраної задачі був обраний певний стек технологій. Датасет зображень МРТ головного мозку був розглянутий на Kaggle – це платформа для спільної роботи над проектами з обробки та аналізу даних, а також для участі в конкурсах машинного навчання. Заснована в 2010 році, Kaggle швидко стала однією з провідних спільнот у сфері даних та машинного навчання. Основна ідея полягає в тому, щоб збирати, розповсюджувати та аналізувати дані, а також вирішувати завдання машинного навчання Для роботи із даними, Kaggle пропонує використовувати Kaggle Kernels – це віртуальні середовища, де можна виконувати код у середовищі з попередньо встановленими бібліотеками та зручним інтерфейсом. Kernels дозволяють інтерактивно досліджувати та обмінюватися кодом, що спрощує співпрацю. [24].

Jupyter Notebook - це інтерактивне середовище для програмування та аналізу даних, яке дозволяє об'єднати код, текст та візуалізацію у одному документі. Jupyter (від назви трьох основних мов програмування: Julia, Python, та R) дозволяє виконувати код почергово та спостерігати за результатами, що робить його ідеальним для вивчення, експериментів та спільної роботи. Весь код та текст у Jupyter розділяється на "комірки". Кожна комірка може містити або код для виконання, або текст (Markdown), що надає коментарі та описи. Кожна комірка коду може бути виконана окремо. Виконання коду виводить результати прямо під коміркою, що спрощує відлагодження та аналіз крок за кроком. Jupyter підтримує не тільки Python, але й інші мови програмування, такі як R та Julia. Кожна комірка може бути використана для коду різних мов. Jupyter інтегрується з багатьма бібліотеками для візуалізації, такими як Matplotlib та Seaborn. Графіки можуть бути побудовані прямо в середовищі та відображені в ноутбуці. Використання Jupyter дозволяє створювати інтерактивні додатки та візуалізації, що дозволяє користувачам взаємодіяти з кодом та даними в режимі реального часу. Ноутбуки можуть бути легко збережені у форматах .ipynb та експортовані до різних форматів, таких як HTML, PDF, або використовувати сервіси для обміну ноутбуків, такі як GitHub або nbviewer. Jupyter має розширену систему плагінів, що дозволяє розширювати його функціональність та інтегрувати додаткові інструменти для роботи із Python [25].

Python - це високорівнева, інтерпретована мова програмування з простим та читабельним синтаксисом, легким для розуміння, що робить його ідеальним для початківців та досвідчених програмістів. Вона широко використовується для розробки програмного забезпечення у різних областях, таких як веб-розробка, аналіз даних, штучний інтелект, наукові дослідження та інше. Python - інтерпретована мова програмування, що означає, що код виконується по одному рядку за раз, без потреби в компіляції. Це робить процес розробки швидшим та більш гнучким. Код Python може бути виконаний на різних операційних системах, таких як Windows, macOS та Linux, що робить його платформонезалежним. Python дозволяє підключати написані на C/C++ модулі для оптимізації продуктивності та

взаємодії з системними функціями. Python має велику та активну спільноту користувачів, яка підтримує сталий розвиток мови та створення нових бібліотек та інструментів.

Python є однією з найпопулярніших мов програмування в світі та є потужним інструментом для розв'язання різноманітних завдань у програмуванні та аналізі даних [26].

Система IPython (англ. Interactive Python) спрощує процес аналізу та візуалізації даних, використовується для інтерактивної взаємодії з даними та виводу результатів у зручному форматі у середовищі Jupyter Notebook.

Для візуалізації та обробки даних використовуються інші технології. Для статистичного аналізу даних використовується бібліотека pandas – вона надає можливості виконувати операції з вибіркою, фільтрацією, групуванням та агрегацією даних і дозволяє легко проаналізувати великий та задокументований масив даних [27]. Також для обробки масивів та роботи з матрицями використовується бібліотека для наукових обчислень NumPy, вона містить функції для виконання математичних операцій, лінійної алгебри, статистики та інших операцій [28]. Для відображення графіків використовуються бібліотеки matplotlib та seaborn. Matplotlib надає широкі можливості для створення зображень, графіків, гістограм, діаграм та інших візуалізаційних елементів [29]. Бібліотека seaborn побудована поверх Matplotlib і дозволяє створювати привабливі та інформативні графіки та діаграми [30].

Для обробки зображень буде використовуватися бібліотека OpenCV – це бібліотека комп'ютерного зору та обробки зображень, яка містить багато функцій для роботи з цифровими зображеннями, такими як зчитування, запис, маніпуляція та аналіз [31]. Також використана бібліотека PIL для роботи з зображеннями у Python, яка надає функції для відкриття, обробки та зберігання зображень [32].

Слід вибрати технології для роботи із методами машинного і глибинного навчання. Бібліотека scikit-learn містить різноманітні алгоритми для класифікації, регресії, кластеризації, відбору ознак та оцінки моделей [33]. Проте, для глибинного навчання був обраний інший набір технологій. Для роботи була обрана

популярна бібліотека TensorFlow для машинного навчання та глибокого навчання. Вона надає інструменти для побудови та тренування нейронних мереж, включаючи конволюційні, рекурентні та повністю зв'язані мережі [34]. Разом з TensorFlow використовується Keras – високорівнева бібліотека для глибокого навчання, яка надає простий та зручний інтерфейс для побудови та тренування нейронних мереж. Вона працює поверх TensorFlow та інших бібліотек глибокого навчання [35].

Висновки до розділу 2

У цьому розділі було розглянуто методи та технології, які можуть бути використані для вирішення поставленої задачі класифікації хвороби Альцгеймера. Була також проаналізована обрана задача та вибраний набір даних з платформи Kaggle, що містить 5121 тренувальне і 1279 тестувальних зображень МРТ головного мозку, розподілених на 4 класи.

3 ІМПЛЕМЕНТАЦІЯ ОБРАНИХ АЛГОРИТМІВ

У цьому розділі розкривається процес вирішення завдання класифікації хвороби Альцгеймера на знімках МРТ головного мозку. Під час виконання дослідження було створено різні архітектури нейронних мереж, такі як VGG19, DenseNet, ResNet, EfficientNet. Також розроблено механізм завантаження вхідних даних, виконано попередню обробку зображень та представлено результати у вигляді графіків та консольного виводу.

3.1 Підключення бібліотек і зчитування даних

Для початку роботи в середовищі потрібно завантажити та підключити основні бібліотеки, цей процес показаний на рис. 3.1.

```
import tensorflow as tf
import pandas as pd
import os
import cv2
import numpy as np
from tqdm import tqdm

from keras.models import load_model
from keras.callbacks import TensorBoard, ModelCheckpoint,ReduceLROnPlateau, CSVLogger
from keras.utils.np_utils import to_categorical

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report,confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt
from PIL import Image

import ipywidgets as widgets
import io
from IPython.display import clear_output,display
```

Рисунок 3.1 – Підключення основних бібліотек для роботи

Далі треба зчитати завантажений з платформи Kaggle датасет для подальшої обробки і використання для тренування моделей, цей процес і кількість зображень в папці із тренувальними даними наведено на рис. 3.2.

Read Data

In [2]:
```
image_path="Alzheimers Dataset/train"
```

In [3]:
```
cls_name=os.listdir(image_path)
print(cls_name)
```

```
['MildDemented', 'ModerateDemented', 'NonDemented', 'VeryMildDemented']
```

In [4]:
```
print("Number of classes : {}".format(len(cls_name)))
```

```
Number of classes : 4
```

In [5]:
```
number_of_images={}
for class_name in cls_name:
    number_of_images[class_name]=len(os.listdir(image_path+"/"+class_name))
images_each_class=pd.DataFrame(number_of_images.values(),index=number_of_images.keys(),columns=["Nu
images_each_class
```

Out[5]:

	Number of images
MildDemented	717
ModerateDemented	52
NonDemented	2560
VeryMildDemented	1792

In [6]:
```
label_name=["MildDemented","ModerateDemented","NonDemented","VeryMildDemented"]
```

Рисунок 3.2 – Зчитування даних і виведення кількості зображень для кожного класу

3.2 Підготовка даних

Більшість із зазначених у розділі 2 архітектур налаштовані так, аби приймати на вхід зображення розміром 224x224. Для цього виконаємо попередню обробку набору даних таким чином, аби змінити розмір всіх зображень на бажаний (див. рис. 3.2).

Рисунок 3.3 – Попередня обробка даних

One-hot encoding (унітарний код) використовується в згорткових нейронних мережах для представлення категоріальних змінних у вигляді векторів бінарних значень. В контексті задачі класифікації, де ми маємо фіксовану кількість класів, one-hot encoding дозволяє представити ці класи у вигляді бінарних векторів, де кожен елемент вектора відповідає одному класу, і лише один елемент має значення 1, вказуючи належність до конкретного класу, а всі інші елементи мають значення 0. У контексті ЗНМ, коли ми маємо задачу класифікації, one-hot encoding використовується для кодування міток класів відповідно до їх категорій. Це дозволяє моделі ефективно взаємодіяти з категоріальними даними та правильно інтерпретувати ці дані під час тренування та передбачення. Такий підхід є стандартним у багатьох задачах класифікації, оскільки дозволяє легко робити порівняння між прогнозами моделі та правильними мітками класів [36].

Split Data And Label

```
In [10]: X_train,X_test,Y_train,Y_test=train_test_split(image_data,label_data,test_size=0.2,random_state=42)
```

apply onehot encoding

```
In [11]: train_label_data_new=[]
         test_label_data_new=[]
         for n in Y_train:
             train_label_data_new.append(label_name.index(n))
         Y_train=train_label_data_new
         Y_train=to_categorical(Y_train)

         for n in Y_test:
             test_label_data_new.append(label_name.index(n))
         Y_test=test_label_data_new
         Y_test=to_categorical(Y_test)
```

```
In [12]: Y_train
```

```
Out[12]: array([[0., 0., 1., 0.],
                [0., 0., 1., 0.],
                [0., 0., 1., 0.],
                ...,
                [0., 0., 0., 1.],
                [0., 0., 0., 1.],
                [0., 0., 1., 0.]], dtype=float32)
```

Рисунок 3.4 – Розділення на тренувальну та тестову вибірки і використання one-hot encoding

3.3 Тренування, тестування та валідація розроблених моделей

Після розробки каркасу проекту, слід визначити вхідні значення гіперпараметрів.

Для тренування моделей були використані набори даних, які містять зображення МРТ головного мозку пацієнтів з різними стадіями хвороби Альцгеймера. Дані були підготовлені та розділені на тренувальний, тестовий та валідаційний набори.

Для тренування моделей були використані такі параметри:

– оптимізатор: Adam;
– функція втрат: категоріальна крос-ентропія;
– кількість епох: 40;
– розмір пакета: 10.

Моделі були треновані на тренувальному наборі та моніторилися за допомогою втрат та точності на валідаційному наборі для запобігання перенавчанню.

Після завершення тренування моделі були протестовані на тестовому наборі даних, який не використовувався під час тренування. Для кожної моделі були обчислені метрики ефективності, такі як точність, відгук, F1-оцінка та матриця невідповідностей. Результати були порівняні та проаналізовані для вибору найефективнішої моделі для подальшого використання.

Цей процес дозволив здійснити оцінку та вибір оптимальної моделі для класифікації хвороби Альцгеймера на знімках МРТ головного мозку.

3.3.1 VGG19

Create Model

```
vgg19=tf.keras.applications.vgg19.VGG19(weights="imagenet",include_top=False,input_shape=
model=vgg19.output
model=tf.keras.layers.GlobalAveragePooling2D()(model)
model=tf.keras.layers.Dropout(0.5)(model)
model=tf.keras.layers.Dense(4,activation="softmax")(model)
model=tf.keras.models.Model(inputs=vgg19.input,outputs=model)
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 176, 208, 3)]     0

 block1_conv1 (Conv2D)       (None, 176, 208, 64)      1792

 block1_conv2 (Conv2D)       (None, 176, 208, 64)      36928

 block1_pool (MaxPooling2D)  (None, 88, 104, 64)       0

 block2_conv1 (Conv2D)       (None, 88, 104, 128)      73856

 block2_conv2 (Conv2D)       (None, 88, 104, 128)      147584

 block2_pool (MaxPooling2D)  (None, 44, 52, 128)       0

 block3_conv1 (Conv2D)       (None, 44, 52, 256)       295168

 block3_conv2 (Conv2D)       (None, 44, 52, 256)       590080

 block3_conv3 (Conv2D)       (None, 44, 52, 256)       590080

 block3_conv4 (Conv2D)       (None, 44, 52, 256)       590080

 block3_pool (MaxPooling2D)  (None, 22, 26, 256)       0
```

Рисунок 3.5 – Створення моделі VGG19

```
history = model.fit(X_train, Y_train,
                    validation_data =(X_test,Y_test),
                    epochs = 40,
                    verbose =1,
                    batch_size = 10,
                    callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])
```

```
Epoch 38: val_accuracy did not improve from 0.60781

Epoch 38: ReduceLROnPlateau reducing learning rate to 1.667718253294226e-21.
512/512 [==============================] - 130s 255ms/step - loss: 0.8047 - accuracy: 0.6086 - val_loss: 0.8259 - val_ac
curacy: 0.6070 - lr: 5.5591e-21
Epoch 39/40
512/512 [==============================] - ETA: 0s - loss: 0.8024 - accuracy: 0.6080
Epoch 39: val_accuracy did not improve from 0.60781

Epoch 39: ReduceLROnPlateau reducing learning rate to 5.003154881051713e-22.
512/512 [==============================] - 129s 253ms/step - loss: 0.8024 - accuracy: 0.6080 - val_loss: 0.8259 - val_ac
curacy: 0.6070 - lr: 1.6677e-21
Epoch 40/40
512/512 [==============================] - ETA: 0s - loss: 0.8024 - accuracy: 0.6057
Epoch 40: val_accuracy did not improve from 0.60781

Epoch 40: ReduceLROnPlateau reducing learning rate to 1.500946434023255e-22.
512/512 [==============================] - 130s 255ms/step - loss: 0.8024 - accuracy: 0.6057 - val_loss: 0.8259 - val_ac
curacy: 0.6070 - lr: 5.0032e-22
```

Рисунок 3.6 – Процес навчання моделі VGG19

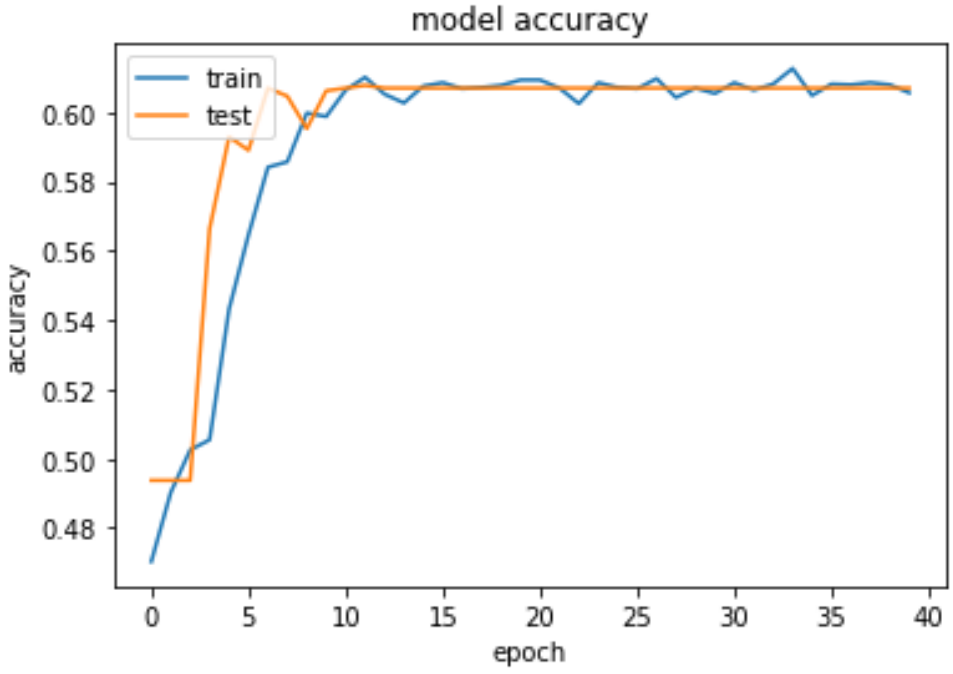

Рисунок 3.7 – Графік точності моделі VGG19

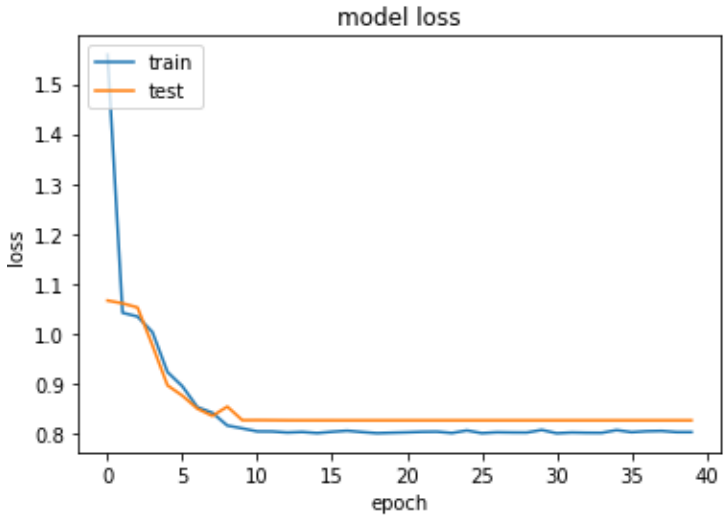

Рисунок 3.8 – Графік втрат моделі VGG19

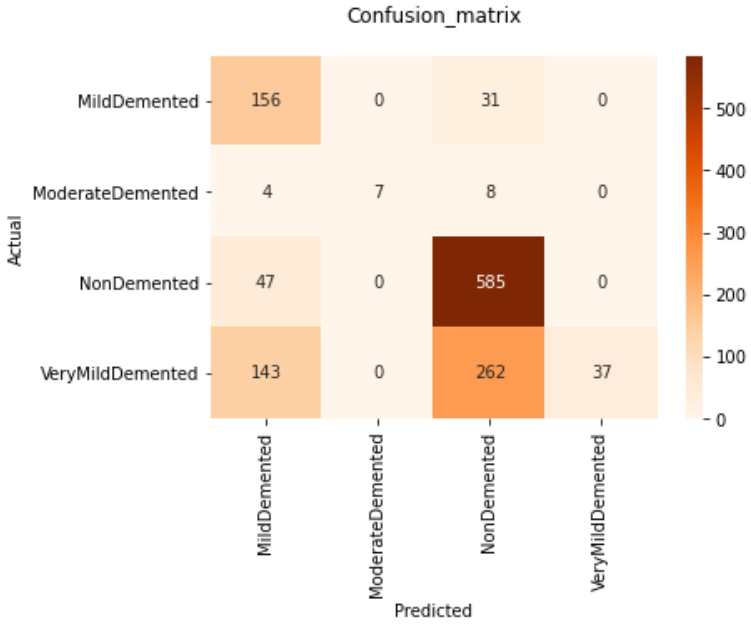

Рисунок 3.9 – Матриця невідповідностей для моделі VGG19

	precision	recall	f1-score	support
MildDemented	0.45	0.83	0.58	187
ModerateDemented	1.00	0.37	0.54	19
NonDemented	0.66	0.93	0.77	632
VeryMildDemented	1.00	0.08	0.15	442
accuracy			0.61	1280
macro avg	0.78	0.55	0.51	1280
weighted avg	0.75	0.61	0.53	1280

Рисунок 3.10 – Розрахунок метрик для кожного класу моделі VGG19

3.3.2 ResNet

3.3.2.1 ResNet101

Create Model

```
resnet101=tf.keras.applications.resnet.ResNet101(weights="imagenet",include_top=False,input_shape=(224,224,3))
model=resnet101.output
model=tf.keras.layers.GlobalAveragePooling2D()(model)
model=tf.keras.layers.Dropout(0.5)(model)
model=tf.keras.layers.Dense(4,activation="softmax")(model)
model=tf.keras.models.Model(inputs=resnet101.input,outputs=model)
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_2[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
```

Рисунок 3.11 – Створення моделі ResNet101

```
history = model.fit(X_train, Y_train,
                    validation_data =(X_test,Y_test),
                    epochs = 40,
                    verbose =1,
                    batch_size = 10,
                    callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])
```

```
Epoch 38: val_accuracy did not improve from 0.93828

Epoch 38: ReduceLROnPlateau reducing learning rate to 2.0589113778655536e-19.
512/512 [==============================] - 160s 312ms/step - loss: 0.0120 - accuracy: 0.9977 - val_loss: 0.2238 - val_ac
curacy: 0.9375 - lr: 6.8630e-19
Epoch 39/40
512/512 [==============================] - ETA: 0s - loss: 0.0106 - accuracy: 0.9977
Epoch 39: val_accuracy did not improve from 0.93828

Epoch 39: ReduceLROnPlateau reducing learning rate to 6.176734056048479e-20.
512/512 [==============================] - 160s 312ms/step - loss: 0.0106 - accuracy: 0.9977 - val_loss: 0.2204 - val_ac
curacy: 0.9367 - lr: 2.0589e-19
Epoch 40/40
512/512 [==============================] - ETA: 0s - loss: 0.0101 - accuracy: 0.9979
Epoch 40: val_accuracy did not improve from 0.93828

Epoch 40: ReduceLROnPlateau reducing learning rate to 1.8530202168145436e-20.
512/512 [==============================] - 182s 356ms/step - loss: 0.0101 - accuracy: 0.9979 - val_loss: 0.2262 - val_ac
curacy: 0.9367 - lr: 6.1767e-20
```

Рисунок 3.12 – Процес навчання моделі ResNet101

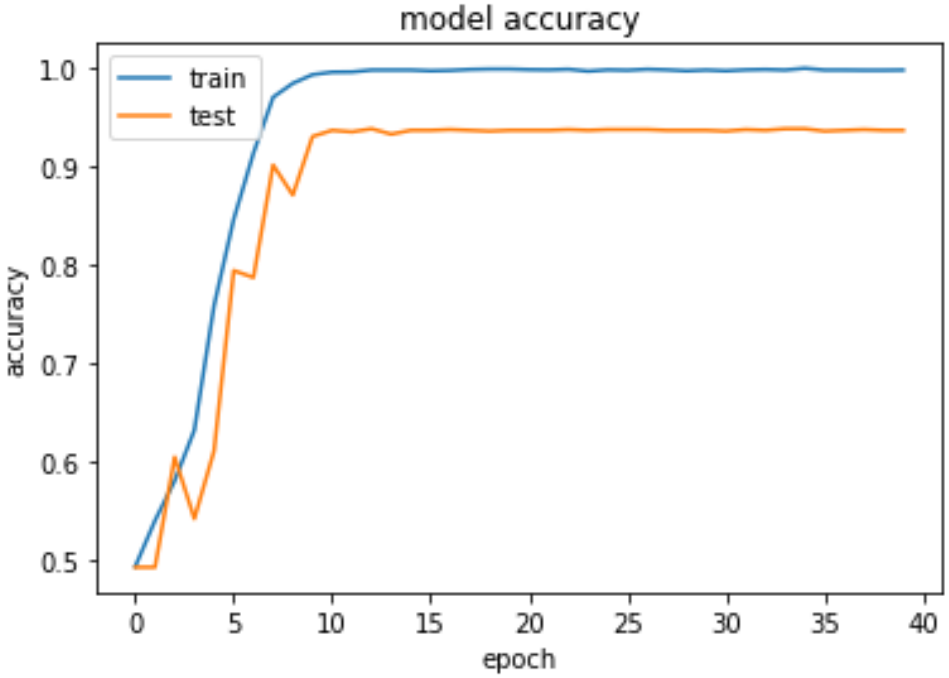

Рисунок 3.13 – Графік точності моделі ResNet101

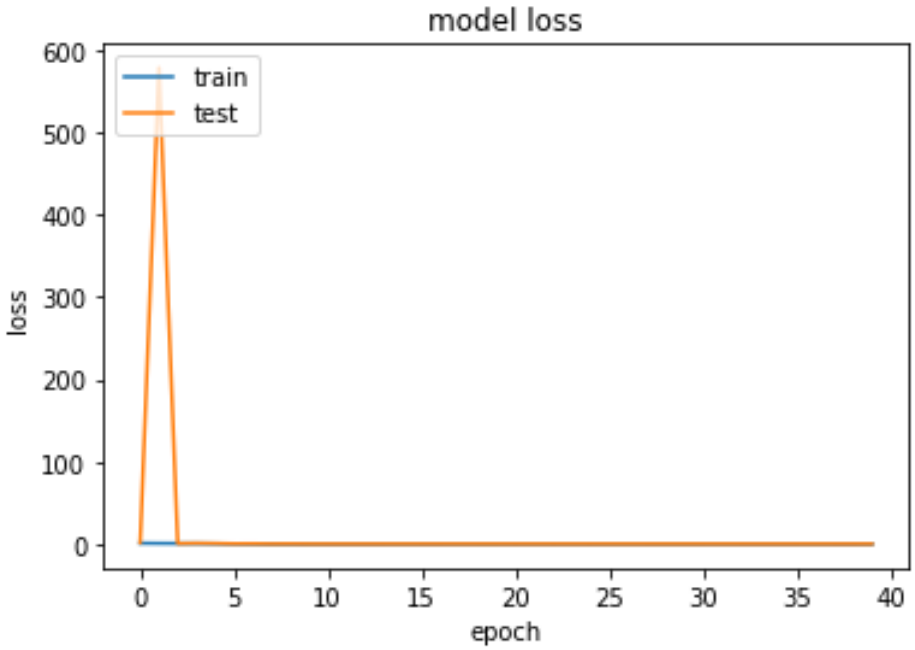

Рисунок 3.14 – Графік втрат моделі ResNet101

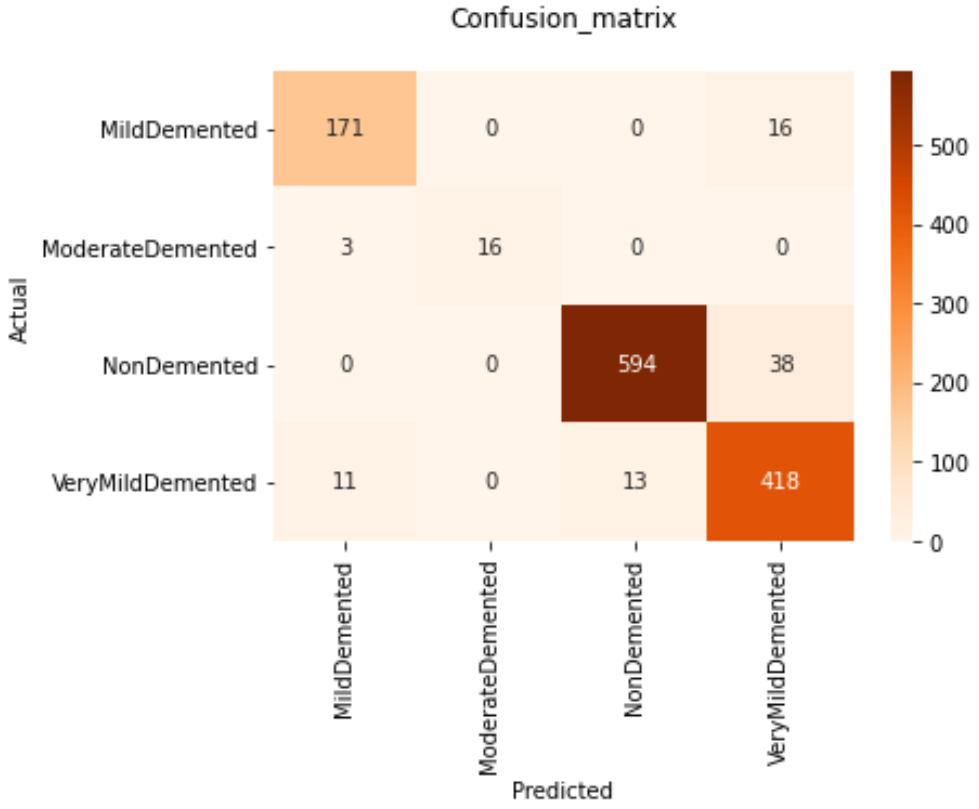

Рисунок 3.15 – Матриця невідповідностей для моделі ResNet101

	precision	recall	f1-score	support
MildDemented	0.92	0.91	0.92	187
ModerateDemented	1.00	0.84	0.91	19
NonDemented	0.98	0.94	0.96	632
VeryMildDemented	0.89	0.95	0.91	442
accuracy			0.94	1280
macro avg	0.95	0.91	0.93	1280
weighted avg	0.94	0.94	0.94	1280

Рисунок 3.16 – Розрахунок метрик для кожного класу моделі ResNet101

3.3.2.2 ResNet152

Create Model

```
resnet152=tf.keras.applications.resnet.ResNet152(weights="imagenet",include_top=False,input_shape=(224,224,3))
model=resnet152.output
model=tf.keras.layers.GlobalAveragePooling2D()(model)
model=tf.keras.layers.Dropout(0.5)(model)
model=tf.keras.layers.Dense(4,activation="softmax")(model)
model=tf.keras.models.Model(inputs=resnet152.input,outputs=model)
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_4[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
```

Рисунок 3.17 – Створення моделі ResNet152

```
history = model.fit(X_train, Y_train,
                    validation_data =(X_test,Y_test),
                    epochs = 40,
                    verbose =1,
                    batch_size = 10,
                    callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])
```

```
Epoch 38: val_accuracy did not improve from 0.97812

Epoch 38: ReduceLROnPlateau reducing learning rate to 6.863037719423359e-19.
512/512 [==============================] - 100s 196ms/step - loss: 0.0061 - accuracy: 0.9988 - val_loss: 0.0899 - val_ac
curacy: 0.9773 - lr: 2.2877e-18
Epoch 39/40
512/512 [==============================] - ETA: 0s - loss: 0.0034 - accuracy: 0.9996
Epoch 39: val_accuracy did not improve from 0.97812

Epoch 39: ReduceLROnPlateau reducing learning rate to 2.0589113778655536e-19.
512/512 [==============================] - 101s 196ms/step - loss: 0.0034 - accuracy: 0.9996 - val_loss: 0.0880 - val_ac
curacy: 0.9773 - lr: 6.8630e-19
Epoch 40/40
512/512 [==============================] - ETA: 0s - loss: 0.0036 - accuracy: 0.9994
Epoch 40: val_accuracy did not improve from 0.97812

Epoch 40: ReduceLROnPlateau reducing learning rate to 6.176734056048479e-20.
512/512 [==============================] - 100s 196ms/step - loss: 0.0036 - accuracy: 0.9994 - val_loss: 0.0897 - val_ac
curacy: 0.9773 - lr: 2.0589e-19
```

Рисунок 3.18 – Процес навчання моделі ResNet152

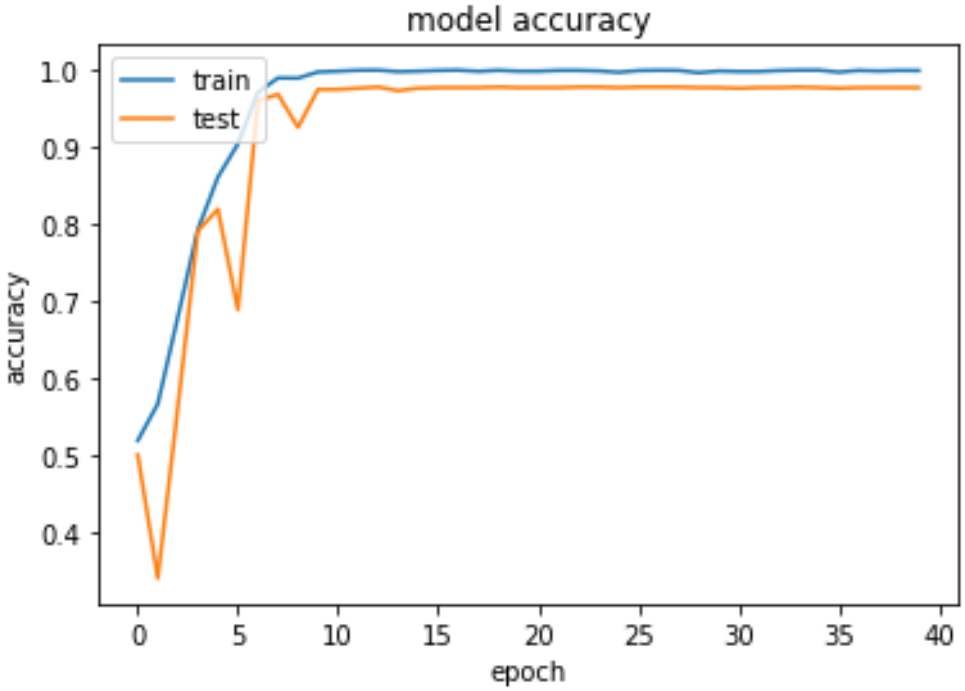

Рисунок 3.19 – Графік точності моделі ResNet152

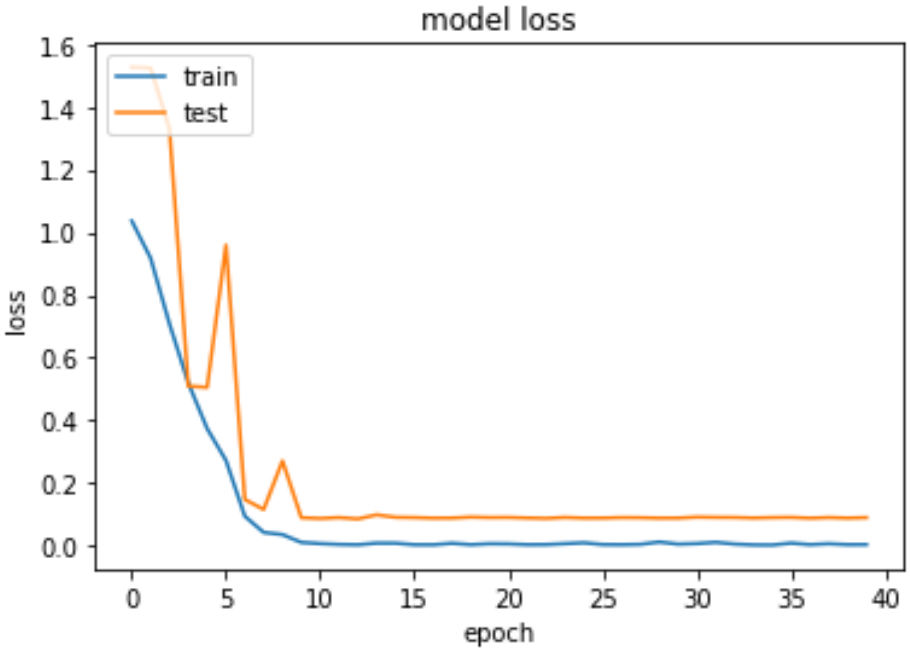

Рисунок 3.20 – Графік втрат моделі ResNet152

3.3.3 DenseNet121

Create Model

```
densenet121=tf.keras.applications.densenet.DenseNet121(weights="imagenet",include_top=False,input_shape=(224,224,3))
model=densenet121.output
model=tf.keras.layers.GlobalAveragePooling2D()(model)
model=tf.keras.layers.Dropout(0.5)(model)
model=tf.keras.layers.Dense(4,activation="softmax")(model)
model=tf.keras.models.Model(inputs=densenet121.input,outputs=model)
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 zero_padding2d (ZeroPadding2D)  (None, 230, 230, 3)  0          ['input_1[0][0]']

 conv1/conv (Conv2D)            (None, 112, 112, 64  9408        ['zero_padding2d[0][0]']
                                )

 conv1/bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1/conv[0][0]']
                                )

 conv1/relu (Activation)        (None, 112, 112, 64  0           ['conv1/bn[0][0]']
                                )

 zero_padding2d_1 (ZeroPadding2  (None, 114, 114, 64  0          ['conv1/relu[0][0]']
```

Рисунок 3.21 – Створення моделі DenseNet121

```
history = model.fit(X_train, Y_train,
                    validation_data =(X_test,Y_test),
                    epochs = 40,
                    verbose =1,
                    batch_size = 10,
                    callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])
```

```
512/512 [==============================] - 150s 293ms/step - loss: 4.0009e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 5.9049e-09
Epoch 38/40
512/512 [==============================] - ETA: 0s - loss: 4.6807e-04 - accuracy: 1.0000
Epoch 38: val_accuracy did not improve from 0.99609
512/512 [==============================] - 147s 288ms/step - loss: 4.6807e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 1.7715e-09
Epoch 39/40
512/512 [==============================] - ETA: 0s - loss: 3.5342e-04 - accuracy: 1.0000
Epoch 39: val_accuracy did not improve from 0.99609

Epoch 39: ReduceLROnPlateau reducing learning rate to 5.314410245205181e-10.
512/512 [==============================] - 153s 300ms/step - loss: 3.5342e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 1.7715e-09
Epoch 40/40
512/512 [==============================] - ETA: 0s - loss: 8.4522e-04 - accuracy: 0.9996
Epoch 40: val_accuracy did not improve from 0.99609
512/512 [==============================] - 149s 290ms/step - loss: 8.4522e-04 - accuracy: 0.9996 - val_loss: 0.0222 - va
l_accuracy: 0.9961 - lr: 5.3144e-10
```

Рисунок 3.22 – Процес навчання моделі DenseNet121

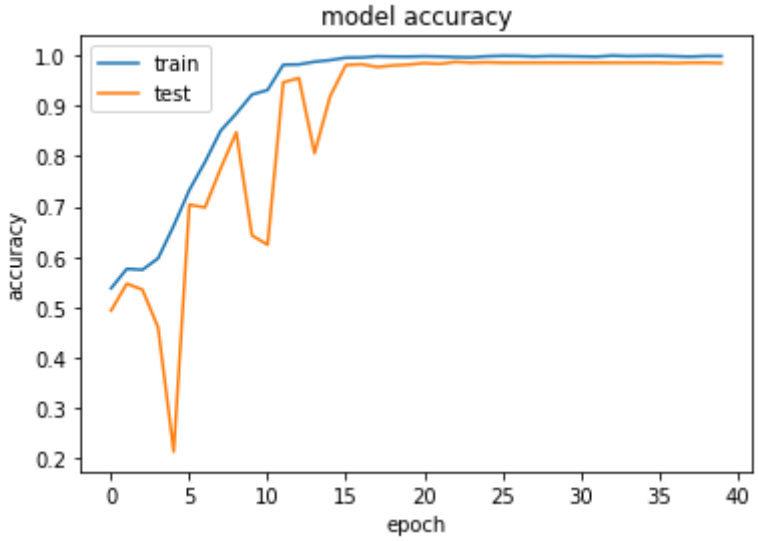

Рисунок 3.23 – Графік точності моделі DenseNet121

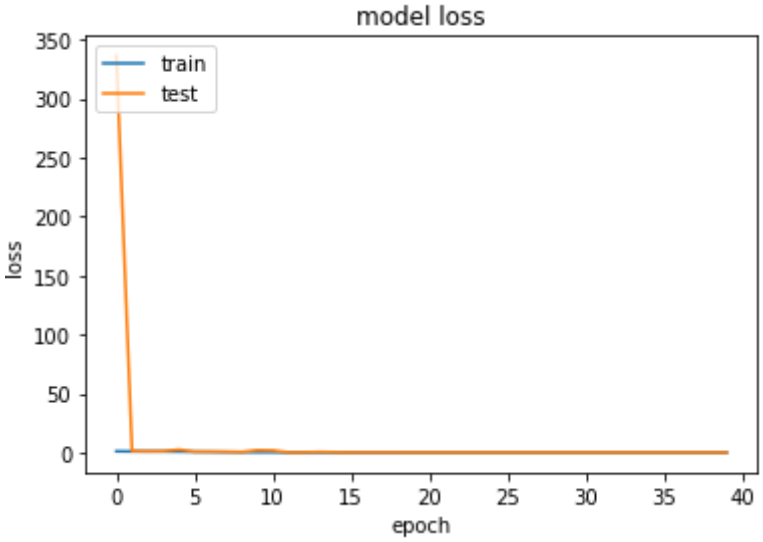

Рисунок 3.24 – Графік втрат моделі DenseNet121

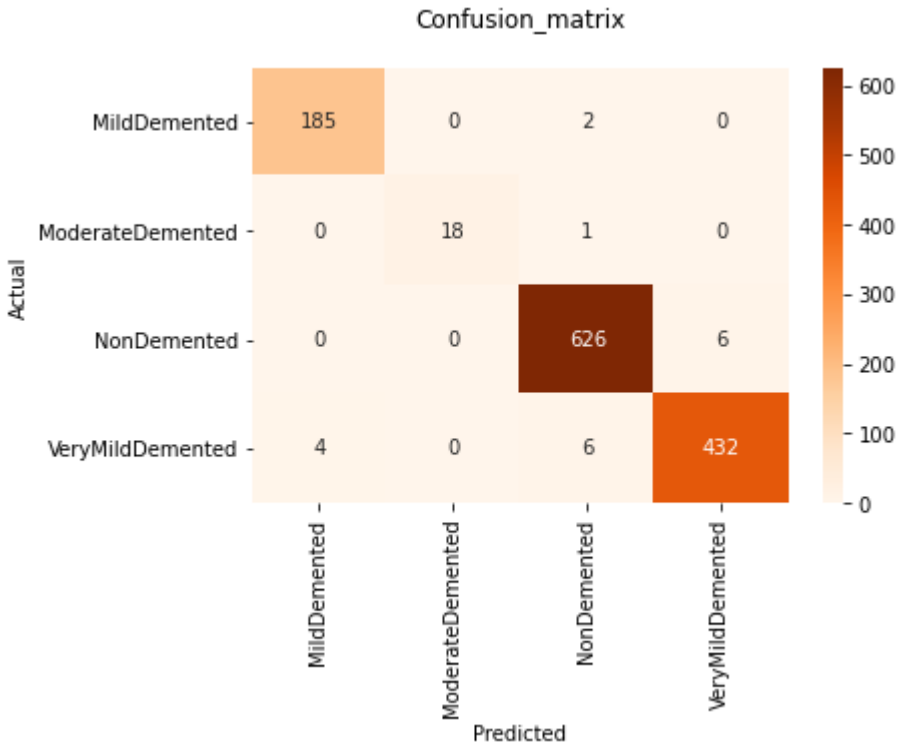

Рисунок 3.25 – Матриця невідповідностей для моделі DenseNet121

Classification Report

```
In [23]: print(classification_report(y_pred, pred_label,target_names=["MildDemented","
```

	precision	recall	f1-score	support
MildDemented	0.98	0.99	0.98	187
ModerateDemented	1.00	0.95	0.97	19
NonDemented	0.99	0.99	0.99	632
VeryMildDemented	0.99	0.98	0.98	442
accuracy			0.99	1280
macro avg	0.99	0.98	0.98	1280
weighted avg	0.99	0.99	0.99	1280

Рисунок 3.26 – Розрахунок метрик для кожного класу моделі DenseNet121

3.3.4 EfficientNetB3

Create Model

```
In [18]: efficientnetb3=tf.keras.applications.efficientnet.EfficientNetB3(weights="imagenet",include_top=False,input_shape=(224,224,3
         model=efficientnetb3.output
         model=tf.keras.layers.GlobalAveragePooling2D()(model)
         model=tf.keras.layers.Dropout(0.5)(model)
         model=tf.keras.layers.Dense(4,activation="softmax")(model)
         model=tf.keras.models.Model(inputs=efficientnetb3.input,outputs=model)
         model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
         model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_5 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 rescaling (Rescaling)          (None, 224, 224, 3)  0           ['input_5[0][0]']

 normalization (Normalization)  (None, 224, 224, 3)  7           ['rescaling[0][0]']

 stem_conv_pad (ZeroPadding2D)  (None, 225, 225, 3)  0           ['normalization[0][0]']

 stem_conv (Conv2D)             (None, 112, 112, 40  1080        ['stem_conv_pad[0][0]']
                                )

 stem_bn (BatchNormalization)   (None, 112, 112, 40  160         ['stem_conv[0][0]']
                                )
```

Рисунок 3.27– Створення моделі EfficientNetB3

```
In [20]: history = model.fit(X_train, Y_train,
                             validation_data =(X_test,Y_test),
                             epochs = 40,
                             verbose =1,
                             batch_size = 10,
                             callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])
```

```
512/512 [==============================] - 150s 293ms/step - loss: 4.0009e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 5.9049e-09
Epoch 38/40
512/512 [==============================] - ETA: 0s - loss: 4.6807e-04 - accuracy: 1.0000
Epoch 38: val_accuracy did not improve from 0.99609
512/512 [==============================] - 147s 288ms/step - loss: 4.6807e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 1.7715e-09
Epoch 39/40
512/512 [==============================] - ETA: 0s - loss: 3.5342e-04 - accuracy: 1.0000
Epoch 39: val_accuracy did not improve from 0.99609

Epoch 39: ReduceLROnPlateau reducing learning rate to 5.314410245205181e-10.
512/512 [==============================] - 153s 300ms/step - loss: 3.5342e-04 - accuracy: 1.0000 - val_loss: 0.0225 - va
l_accuracy: 0.9961 - lr: 1.7715e-09
Epoch 40/40
512/512 [==============================] - ETA: 0s - loss: 8.4522e-04 - accuracy: 0.9996
Epoch 40: val_accuracy did not improve from 0.99609
512/512 [==============================] - 149s 290ms/step - loss: 8.4522e-04 - accuracy: 0.9996 - val_loss: 0.0222 - va
l_accuracy: 0.9961 - lr: 5.3144e-10
```

Рисунок 3.28 – Процес навчання моделі EfficientNetB3

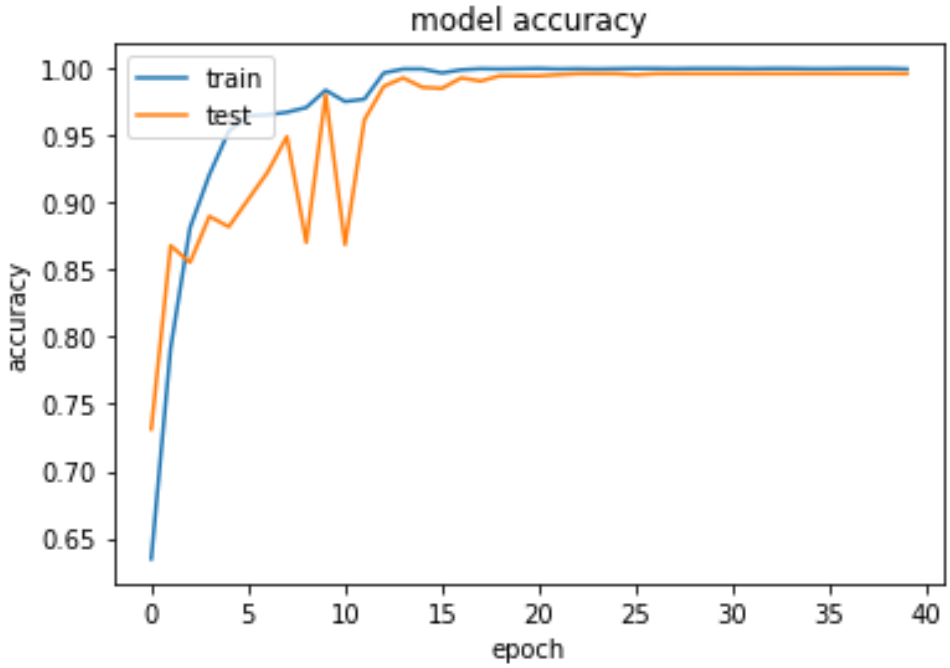

Рисунок 3.29 – Графік точності моделі EfficientNetB3

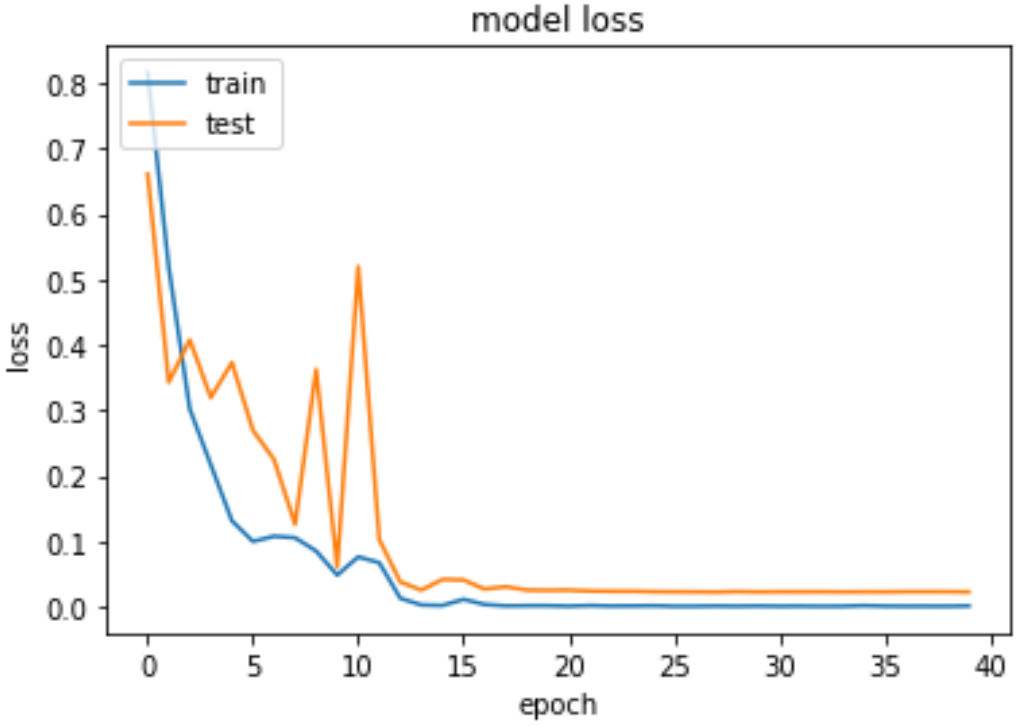

Рисунок 3.30 – Графік втрат моделі EfficientNetB3

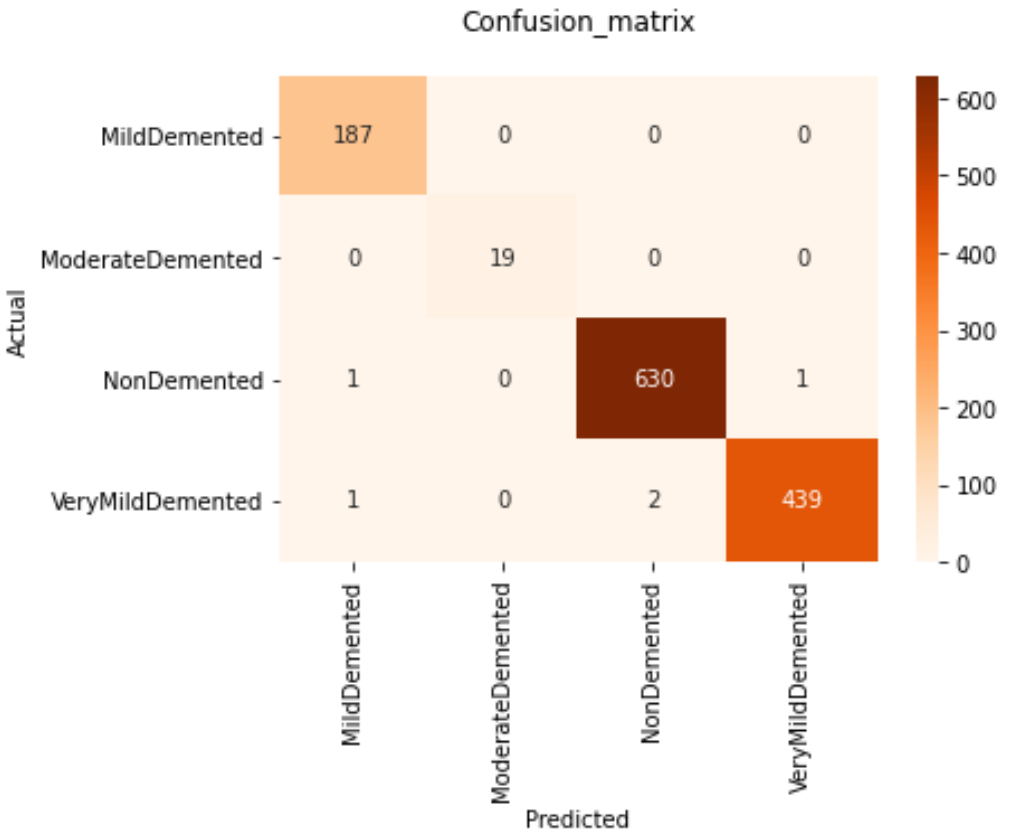

Рисунок 3.31 – Матриця невідповідностей для моделі EfficientNetB3

Classification Report

```
In [28]: print(classification_report(y_pred, pred_label,target_names=["Mi
```

	precision	recall	f1-score	support
MildDemented	0.99	1.00	0.99	187
ModerateDemented	1.00	1.00	1.00	19
NonDemented	1.00	1.00	1.00	632
VeryMildDemented	1.00	0.99	1.00	442
accuracy			1.00	1280
macro avg	1.00	1.00	1.00	1280
weighted avg	1.00	1.00	1.00	1280

Рисунок 3.32 – Розрахунок метрик для кожного класу моделі EfficientNetB3

3.4 Перевірка на тестових даних

Створимо функцію для завантаження власного файлу зображення і вибору моделі із відсотком передбачуваності для кожного класу хвороби Альцгеймера (див. рис. 3.33). Результати передбачень на тестових даних показані на рис. 3.34 – 3.38.

```
model_select = widgets.Select(
    options={'resnet101.h5', 'resnet152.h5', 'densenet121.h5', 'vgg19.h5', 'efficient_B3.h5'},
    description='Select Model:',
    disabled=False,
)

uploader=widgets.FileUpload()
display(model_select, uploader)

def image_prediction(uploader):
    for name, fileinfo in uploader.value.items():
        image_file=Image.open(io.BytesIO(fileinfo["content"]))
    model = load_model(model_select.value)
    predict_image=cv2.cvtColor(np.array(image_file),cv2.COLOR_RGB2BGR)
    predict_image=cv2.resize(predict_image,(208, 176))
    predict_image=predict_image.reshape(1,176, 208, 3)

    predict_probabilities = model(predict_image)[0]
    predicted_class = np.argmax(predict_probabilities)
    class_labels = ["Mild Demented", "Moderate Demented", "Non Demented", "Very Mild Demented"]

    # Виводимо всі ймовірності класів
    for i, prob in enumerate(predict_probabilities):
        print(f"Probability of {class_labels[i]}: {prob * 100:.2f}%")

    # Виводимо клас з найбільшою ймовірністю
    print(f"Model prediction is {class_labels[predicted_class]}")


button=widgets.Button(description="Predict")
out_put=widgets.Output()

def button_click(_):
    with out_put:
        clear_output()
        try:
            image_prediction(uploader)
        except:
            print("Enter Correct Image File")

button.on_click(button_click)
widgets.VBox([button,out_put])
```

Рисунок 3.33 – Функція вибору моделі, завантаження зображення знімку МРТ і передбачення класу захворювання Альцгеймера

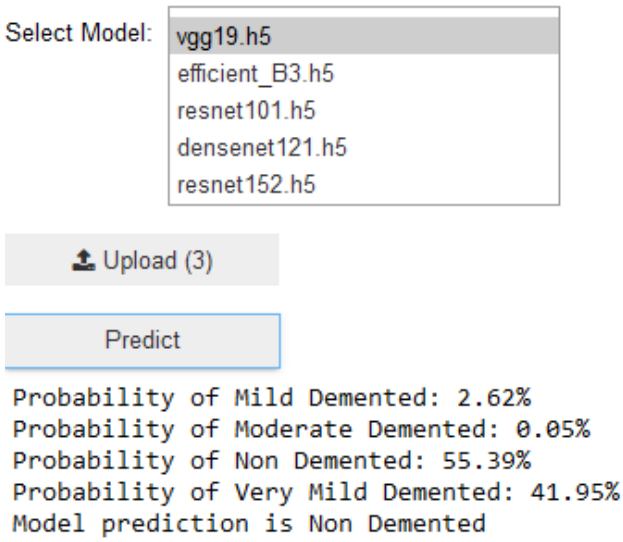

Рисунок 3.34 – Результат передбачення моделі VGG19

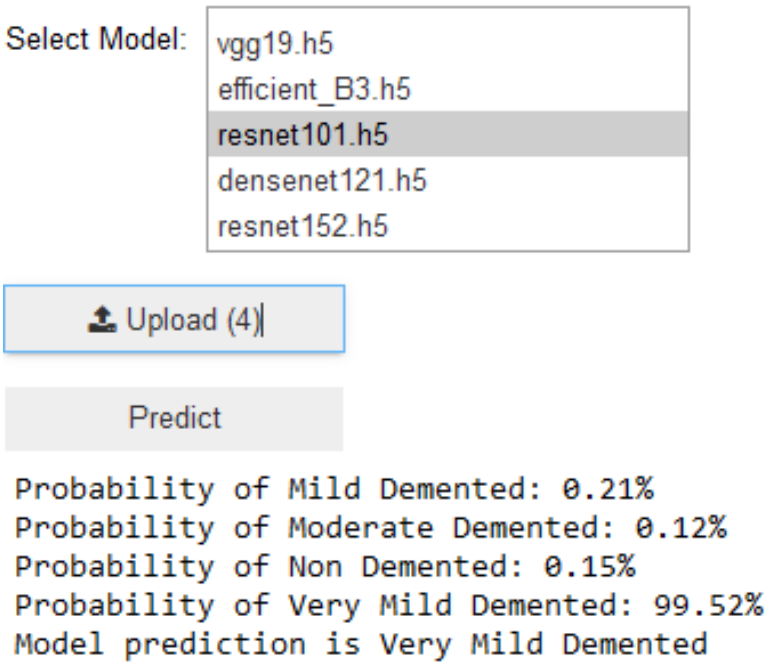

Рисунок 3.35 – Результат передбачення моделі ResNet101

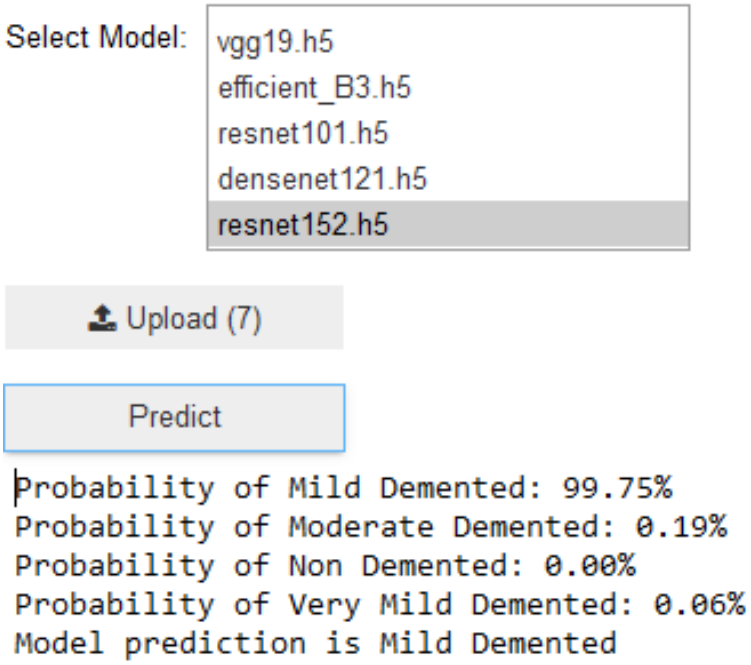

Рисунок 3.36 – Результат передбачення моделі ResNet152

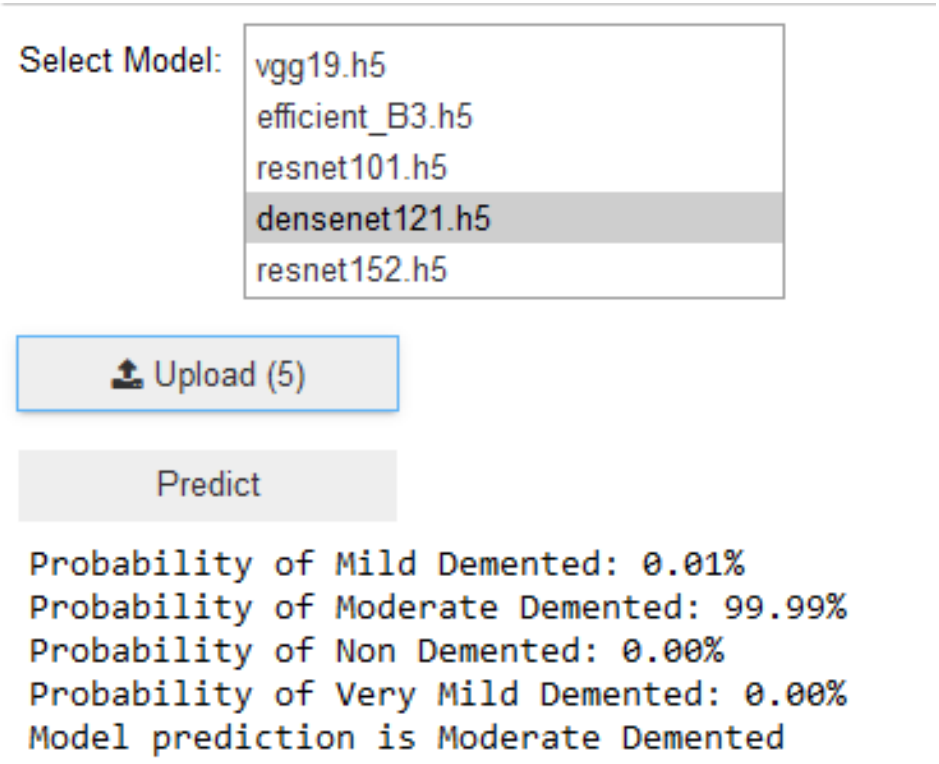

Рисунок 3.37 – Результат передбачення моделі DenseNet121

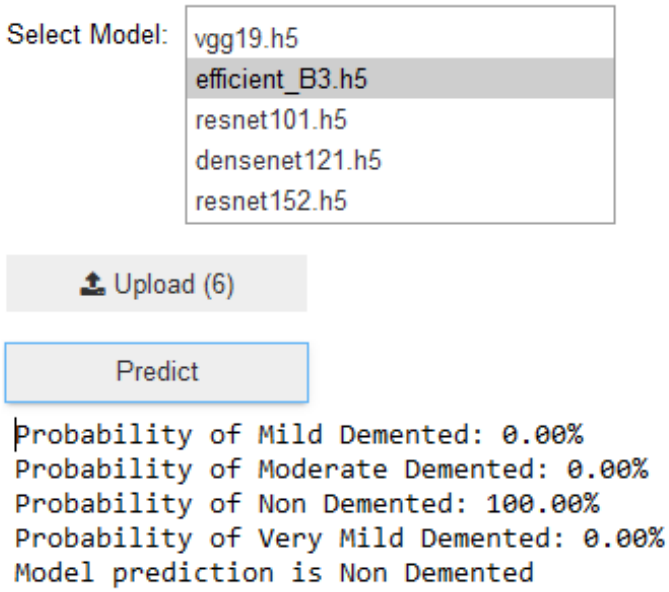

Рисунок 3.38 – Результат передбачення моделі EfficientNetB3

3.5 Вибір метрик для порівняння натренованих моделей

При оцінці класифікатора важливо враховувати проблему дисбалансу класів, що виникає, коли більшість записів у наборі даних належать до одного класу. Цей аспект також стосується досліджуваного набору даних. Існує кілька методів для вимірювання ефективності класифікатора, проте найбільш об'єктивним є той, який враховує, наскільки класифікатор успішно виконує свою основну функцію. Важливо встановити показники ефективності, які відображатимуть корисність моделі, а не просто її точність. Альтернативні показники ефективності можна отримати з матриці невідповідностей.

Матриця невідповідностей - це таблиця, що відображає класифікацію прогнозів у відповідності до того, чи вони збігаються з фактичними значеннями. Одне з напрямків таблиці - це різні категорії прогнозованих значень, а інший - ті самі категорії для фактичних значень. Якщо прогнозоване значення збігається з фактичним, класифікація вважається правильною, і ці прогнози розташовуються у матриці невідповідностей вздовж діагоналі. Інші комірки матриці вказують на випадки, коли прогнозоване значення відрізняється від фактичного, що відповідає

неправильним прогнозам. Показники ефективності моделей класифікації базуються на кількості прогнозів, які потрапили на діагональ та поза нею [37].

Найпоширеніші показники ефективності враховують здатність моделі відрізняти один клас від іншого. У цьому контексті цільовий клас вважається позитивним, а інші - негативними. Залежність між прогнозами позитивного та негативного класів може бути представлена у вигляді матриці невідповідностей розміром 2×2 (див. табл. 3.1), у якій показано, чи належать прогнози до різних категорій:

– True Positive (TP): коректно визначений об'єкт, що відноситься до цільового класу;

– True Negative (TN): правильно визначений об'єкт, який не входить до класу, що вивчається;

– False Positive (FP): некоректно визначений об'єкт, помилково віднесений до цільового класу;

– False Negative (FN): некоректно визначений об'єкт, який має відноситися до цільового класу, але цього не враховано.

Таблиця 3.1 – Матриця невідповідностей

		Фактична мітка	
		Істина	**Хибність**
Передбачення	**Істина**	Істинно-позитивний (TP)	Хибно-позитивний (FP)
	Хибність	Хибно-негативний (FN)	Істинно-негативний (TN)

Подібна матриця невідповідностей є основою багатьох найважливіших показників ефективності моделі.

3.5.1 Точність прогнозування

Шляхом використання матриці невідповідностей розміром 2×2 можна формалізувати визначення точності прогнозування (accuracy), іноді відомої як коефіцієнт успішності:

$$accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{TP + TN + FP + FN}. \quad (3.1)$$

У цьому виразі TP, TN, FP та FN вказують на кількість випадків, коли прогнози моделі попадають у кожну відповідну категорію. Таким чином, точність представляє собою відношення суми істинно-позитивних і істинно-негативних значень до загальної кількості прогнозів [38].

3.5.2 Прецизійність та повнота

Ці статистичні показники використовуються в основному в задачах пошуку інформації і призначені для того, щоб показати, наскільки цікаві та актуальні результати моделі або прогнози розведені безглуздим шумом.

Прецизійність визначається як відношення істинно-позитивних прикладів, які були правильно прогнозовані як позитивні, до загальної кількості прикладів, які модель визначила як позитивні. Іншими словами, це вимірює, наскільки часто модель правильно ідентифікує позитивний клас. Точна модель буде прогнозувати позитивний клас тільки у випадках, коли це дійсно відповідає реальності.

$$precision = \frac{TP}{TP + FP}. \quad (3.2)$$

Повнота вказує на те, наскільки вичерпною є модель у виявленні всіх можливих відповідей. Цей показник визначає відношення істинно позитивних прогнозів до загальної кількості позитивних прогнозів. Модель з високим рівнем

повноти зафіксує значну частину істинних позитивних прикладів, що свідчить про її вміння виявляти широкий спектр позитивних випадків [39].

$$recall = \frac{TP}{TP + FN} \tag{3.3}$$

3.5.3 F-міра

F-міра комбінує точність і повноту, використовуючи середнє гармонійне – це тип середнього, який враховує швидкість зміни величини. Обчислення F-міри виконується за допомогою наступної формули:

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.4}$$

Щоб обчислити F-міру, можна скористатися розрахованими раніше значеннями точності та повноти. Вийде точно те саме, як при використанні розрахунків за матрицею невідповідностей.

Оскільки F-міра описує ефективність моделі одним числом, вона є зручним способом порівняння кількох моделей між собою [40].

3.6 Порівняльний аналіз результатів

Використовуючи пакет sklearn можна автоматично розрахувати коефіцієнти на основі векторів спрогнозованих та реальних класів. Для обчислення коефіцієнтів і їх виведення у вигляді таблиці використовується функція classification_report(). Отримані результати дозволяють зробити висновок про ефективність різних архітектур ЗНМ для задачі класифікації хвороби Альцгеймера на знімках МРТ головного мозку. Результат порівняння розроблених архітектур ЗНМ між собою за обчисленими метриками наведено в табл. 3.2.

Таблиця 3.2 – Аналіз архітектур ЗНМ для класифікації хвороби Альцгеймера

	Accuracy	Precision	Recall	F1-score
VGG19	0.61	0.75	0.61	0.53
ResNet101	0.94	0.94	0.94	0.94
ResNet152	0.98	0.98	0.98	0.98
DenseNet121	0.99	0.99	0.98	0.98
EfficientNetB3	0.99	0.99	0.99	0.99

Висновки до розділу 3

У даному розділі були використані та розроблені обрані алгоритми машинного навчання для вирішення задачі класифікації хвороби Альцгеймера на знімках МРТ головного мозку. Під час створення даного розділу були імплементовані та натреновані архітектури обраних моделей. Після цього в розділі зазначені результати метрик у вигляді порівняльної таблиці. Врешті решт, під час проведення порівняння була обрана створена варіація архітектури EfficientNetB3. Її точність досягла 99%, що є досить значним результатом у порівнянні з іншими архітектурами моделей ЗНМ.

ВИСНОВКИ

Матеріал даної роботи зосереджений на дослідженні алгоритмів машинного навчання для автоматизованої класифікації хвороби Альцгеймера на знімках МРТ головного мозку.

У першому розділі дано загальний опис хворобі Альцгеймера та застосуванню комп'ютерного зору в біомедицині. Також були досліджені види машинного та глибинного навчання та проаналізовані існуючі дослідження по цій темі. Також були сформовані потрібні для проведення дослідження кроки: вивчення та вибір доступних бібліотек для роботи в сфері машинного та глибинного навчання, тестування алгоритмів машинного та глибинного навчання, оптимізація класифікації хвороби Альцгеймера при покращені роботи існуючих архітектур.

У другому розділі було розглянуто методи і технології, які можуть бути застосовані для вирішення поставленої задачі класифікації хвороби Альцгеймера. Була також проаналізована обрана задача та вибраний набір даних з платформи Kaggle. Цей набір даних було проаналізовано та оброблено для подальшої роботи.

У третьому розділі були використані та розроблені обрані алгоритми машинного навчання для вирішення задачі класифікації хвороби Альцгеймера на знімках МРТ головного мозку. Під час створення даного розділу були імплементовані та натреновані архітектури обраних моделей. Після цього в розділі зазначені результати метрик у вигляді порівняльної таблиці. Врешті решт, під час проведення порівняння була обрана створена варіація архітектури EfficientNetB3. Її точність досягла 99%, що є досить значним результатом у порівнянні з іншими архітектурами моделей ЗНМ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. G. Gupta, A. Gupta, P. Barura, V. Jaiswal, C. S. Engineering, and N. Delhi, “Mobile Health Applications and Android Toolkit for Alzheimer Patients, Caregivers and Doctors,” vol. 11, no. 1, pp. 199–205, 2019.

2. S. H. Wang, P. Phillips, Y. Sui, B. Liu, M. Yang, and H. Cheng, “Classification of Alzheimer’s Disease Based on Eight-Layer Convolutional Neural Network with Leaky Rectified Linear Unit and Max Pooling,” J. Med. Syst., vol. 42, no. 5, p. 85, 2018, doi: 10.1007/s10916-018-0932-7.

3. 7 Types of Classification Algorithms. Analytics India Magazine. URL: https://analyticsindiamag.com/7-types-classification-algorithms/ (дата звернення: 02.02.2024).

4. 1. D. Hand, H. Mannila, P. Smyth.: Principles of Data Mining. The MIT Press, 2001.

5. N. Cristianini, J. Shawe-Taylor. An Introduction to Support Vector Machines, Cambridge: Cambridge University Press, 2000.

6. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature. URL: https://www.nature.com/articles/nature14539/ (дата звернення: 02.02.2024).

7. Szeliski R. Computer Vision: Algorithms and Applications. Springer Nature, 2022.

8. Introduction to convolutional neural networks (CNN). Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ (date of access: 02.02.2024).

9. What are convolutional neural networks?. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/convolutional-neural-networks (дата звернення: 02.02.2024).

10. Craig L., Awati R. What is a convolutional neural network (CNN)?. Enterprise AI. URL: https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network/ (дата звернення: 02.02.2024).

11. A gentle introduction to pooling layers for convolutional neural networks - machinelearningmastery.com. MachineLearningMastery.com. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (дата звернення: 02.02.2024).

12. DeepAI. Max pooling. DeepAI. URL: https://deepai.org/machine-learning-glossary-and-terms/max-pooling/ (дата звернення: 02.02.2024).

13. Fully connected layer vs. convolutional layer: explained. Built In. URL: https://builtin.com/machine-learning/fully-connected-layer/ (дата звернення: 02.02.2024).

14. Bangar S. VGG-Net Architecture explained. Medium. URL: https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f/ (дата звернення: 02.02.2024).

15. Anwar A. Difference between AlexNet, VGGNet, ResNet, and Inception. Towards Data Science. URL: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96/ (дата звернення: 02.02.2024).

16. ResNet: The basics and 3 ResNet extensions. Datagen. URL: https://datagen.tech/guides/computer-vision/resnet/ (дата звернення: 02.02.2024).

17. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q., “Densely Connected Convolutional Networks” Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

18. Tsang S.-H. Review: DenseNet – dense convolutional network (image classification). Towards Data Science. URL: https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803/ (дата звернення: 02.02.2024).

19. Bouchard L. State-of-the-Art convolutional neural networks explained - DenseNet. Louis-François Bouchard, aka What's AI. URL: https://www.louisbouchard.ai/densenet-explained/ (дата звернення: 02.02.2024).

20. Tan M., Le Q., "EfficientNet: Rethinking model scaling for convolutional neural networks." International conference on machine learning. PMLR, 2019.

21. Malingan N. EfficientNet - scaler topics. Scaler Topics. URL: https://www.scaler.com/topics/deep-learning/efficientNet/ (дата звернення: 02.02.2024).

22. Potrimba P. What is EfficientNet? The ultimate guide. Roboflow Blog. URL: https://blog.roboflow.com/what-is-efficientnet/ (дата звернення: 02.02.2024).

23. Sarkar A. Understanding EfficientNet – The most powerful CNN architecture. Medium. URL: https://medium.com/mlearning-ai/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad/ (дата звернення: 02.02.2024).

24. Uslu Ç. What is Kaggle?. Learn Data Science and AI Online | DataCamp. URL: https://www.datacamp.com/blog/what-is-kaggle/ (дата звернення: 02.02.2024).

25. What is jupyter notebook?. Domino Data Lab | Unleash Data Science at Scale. URL: https://domino.ai/data-science-dictionary/jupyter-notebook/ (дата звернення: 02.02.2024).

26. What is Python used for? A beginner's guide. Coursera. URL: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python/ (дата звернення: 02.02.2024).

27. pandas. PyPI. URL: https://pypi.org/project/pandas/ (дата звернення: 02.02.2024).

28. What is numpy? – numpy v1.26 manual. NumPy. URL: https://numpy.org/doc/stable/user/whatisnumpy.html (дата звернення: 16.02.2024).

29. Matplotlib documentation – Matplotlib 3.8.3 documentation. Matplotlib – Visualization with Python. URL: https://matplotlib.org/stable/index.html#learn (дата звернення: 02.02.2024).

30. An introduction to seaborn – seaborn 0.13.2 documentation. seaborn: statistical data visualization – seaborn 0.13.2 documentation. URL: https://seaborn.pydata.org/tutorial/introduction.html (дата звернення: 02.02.2024).

31. About. OpenCV. URL: https://opencv.org/about/ (дата звернення: 02.02.2024).

32. Pillow. Pillow (PIL Fork). URL: https://pillow.readthedocs.io/en/stable/ (дата звернення: 02.02.2024).

33. Getting started. scikit-learn. URL: https://scikit-learn.org/stable/getting_started.html (дата звернення: 02.02.2024).

34. TensorFlow. TensorFlow. URL: https://www.tensorflow.org/ (дата звернення: 02.02.2024).

35. Keras documentation: About Keras 3. Keras: Deep Learning for humans. URL: https://keras.io/about/ (дата звернення: 02.02.2024).

36. One hot encoding in machine learning. GeeksforGeeks. URL: https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/ (дата звернення: 02.02.2024).

37. Narkhede S. Understanding confusion matrix. Towards Data Science. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (дата звернення: 02.02.2024).

38. Parashar N. What is an accuracy score and how to check it?. Medium. URL: https://medium.com/@niitwork0921/what-is-an-accuracy-score-and-how-to-check-it-13b23eeed6a3 (дата звернення: 02.02.2024).

39. Precision and Recall | Essential Metrics for Machine Learning Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/ (дата звернення: 02.02.2024).

40. F1 score in machine learning: intro & calculation. V7 | The AI Data Engine for Computer Vision & Generative AI. URL: https://www.v7labs.com/blog/f1-score-guide (дата звернення: 02.02.2024).

41. Alzheimer disease classification through transfer learning approach / N. Raza et al. Diagnostics. 2023. Т. 13, № 4. С. 801. URL: https://doi.org/10.3390/diagnostics13040801 (дата звернення: 02.02.2024).

42. Deep learning for computer vision: a brief review / A. Voulodimos et al. Computational intelligence and neuroscience. 2018. Т. 2018. С. 1–13. URL: https://doi.org/10.1155/2018/7068349 (дата звернення: 02.02.2024).

43. Multi-classification of Alzheimer disease on magnetic resonance images (MRI) using deep convolutional neural network (DCNN) approaches / S. A. Ajagbe et al. International journal of advanced computer research. 2021. Т. 11, № 53. С. 51–60. URL: https://doi.org/10.19101/ijacr.2021.1152001 (дата звернення: 02.02.2024).

44. Yamanakkanavar N., Choi J. Y., Lee B., MRI segmentation and classification of human brain using deep learning for diagnosis of Alzheimer's disease: a survey. Sensors. 2020. Т. 20, № 11. С. 3243. URL: https://doi.org/10.3390/s20113243 (дата звернення: 02.02.2024).

45. Alzheimer disease classification through transfer learning approach / N. Raza et al. Diagnostics. 2023. Т. 13, № 4. С. 801. URL: https://doi.org/10.3390/diagnostics13040801 (дата звернення: 02.02.2024).

46. Lazarova S., Grigorova D., Petrova-Antonova D. Detection of Alzheimer's disease using logistic regression and clock drawing errors. Brain sciences. 2023. Т. 13, № 8. С. 1139. URL: https://doi.org/10.3390/brainsci13081139 (дата звернення: 02.02.2024).

ДОДАТОК А

Код програми

```
import tensorflow as tf
import pandas as pd
import os
import cv2
import numpy as np
from tqdm import tqdm
from keras.models import load_model
from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLROnPlateau,
CSVLogger
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import ipywidgets as widgets
import io
from IPython.display import clear_output, display

model_select = widgets.Select(
    options={'resnet101.h5', 'resnet152.h5', 'densenet121.h5', 'vgg19.h5',
'efficient_B3.h5'},
    description='Select Model:',
    disabled=False,
)

uploader=widgets.FileUpload()
display(model_select, uploader)

def image_prediction(uploader):
    for name, fileinfo in uploader.value.items():
        image_file=Image.open(io.BytesIO(fileinfo["content"]))
    model = load_model(model_select.value)
    predict_image=cv2.cvtColor(np.array(image_file),cv2.COLOR_RGB2BGR)
    predict_image=cv2.resize(predict_image,(176, 208))
    predict_image=predict_image.reshape(1,208, 176, 3)

    predict_probabilities = model(predict_image)[0]
    predicted_class = np.argmax(predict_probabilities)
    class_labels = ["Mild Demented", "Moderate Demented", "Non Demented", "Very Mild
Demented"]

    # Виводимо всі ймовірності класів
    for i, prob in enumerate(predict_probabilities):
        print(f"Probability of {class_labels[i]}: {prob * 100:.2f}%")
```

```
    # Виводимо клас з найбільшою ймовірністю
    print(f"Model prediction is {class_labels[predicted_class]}")


button=widgets.Button(description="Predict")
out_put=widgets.Output()

def button_click(_):
    with out_put:
        clear_output()
        try:
            image_prediction(uploader)
        except:
            print("Enter Correct Image File")

button.on_click(button_click)
widgets.VBox([button,out_put])
image_path="Alzheimers Dataset/train"
cls_name=os.listdir(image_path)
print(cls_name)

print("Number of classes : {}".format(len(cls_name)))
number_of_images={}

for class_name in cls_name:
    number_of_images[class_name]=len(os.listdir(image_path+"/"+class_name))
images_each_class=pd.DataFrame(number_of_images.values(),index=number_of_images.keys(
),columns=["Number of images"])
images_each_class
print("Preprocess train data\n")

image_data=[]
label_data=[]

for i in label_name:
    data_path=os.path.join("Alzheimers Dataset","train",i)
    for m in tqdm(os.listdir(data_path)):
        image=cv2.imread(os.path.join(data_path,m))
        image=cv2.resize(image,(208,176))
        image_data.append(image)
        label_data.append(i)

for i in label_name:
    data_path=os.path.join("Alzheimers Dataset","test",i)
    for m in tqdm(os.listdir(data_path)):
        image=cv2.imread(os.path.join(data_path,m))
        image=cv2.resize(image,(208,176))

        image_data.append(image)
```

```
        label_data.append(i)

image_data=np.array(image_data)
label_data=np.array(label_data)
image_data.shape

image_data,label_data=shuffle(image_data,label_data,random_state=42)
X_train,X_test,Y_train,Y_test=train_test_split(image_data,label_data,test_size=0.2,ra
ndom_state=42)

train_label_data_new=[]
test_label_data_new=[]

for n in Y_train:
    train_label_data_new.append(label_name.index(n))
Y_train=train_label_data_new
Y_train=to_categorical(Y_train)

for n in Y_test:
    test_label_data_new.append(label_name.index(n))
Y_test=test_label_data_new
Y_test=to_categorical(Y_test)
Y_train

vgg19=tf.keras.applications.vgg19.VGG19(weights="imagenet",include_top=False,input_sh
ape=(176,208,3))
model=vgg19.output
model=tf.keras.layers.GlobalAveragePooling2D()(model)
model=tf.keras.layers.Dropout(0.5)(model)
model=tf.keras.layers.Dense(4,activation="softmax")(model)
model=tf.keras.models.Model(inputs=vgg19.input,outputs=model)

model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
model.summary()

csvlogger = CSVLogger("vgg19.log")
tensorboard = TensorBoard(log_dir="logs")
checkpoint = ModelCheckpoint("vgg19.h5",
                             monitor='val_accuracy',
                             mode = "auto",
                             verbose = 1,
                             save_best_only =True)

reduce_LR = ReduceLROnPlateau(monitor="val_accuracy",
                             factor = 0.3,
                             patience = 1,
                             min_delta=0.001,
                             mode = "auto",
                             verbose = 1)
history = model.fit(X_train, Y_train,
```

```
                    validation_data =(X_test,Y_test),
                    epochs = 40,
                    verbose =1,
                    batch_size = 10,
                    callbacks = [tensorboard, csvlogger, checkpoint, reduce_LR])

result = model.evaluate(X_test,Y_test,batch_size=10)
print("Accuracy is\n",result)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

prediction = model.predict(X_test)
pred_label = np.argmax(prediction,axis=1)
y_pred = np.argmax(Y_test,axis=1)

cm = confusion_matrix(y_pred,pred_label)
sns.heatmap(cm,  annot=True, fmt="d"
,cmap="Oranges",xticklabels=label_name,yticklabels=label_name)
plt.title("Confusion_matrix\n")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

print(classification_report(y_pred, pred_label,
target_names=["MildDemented","ModerateDemented","NonDemented","VeryMildDemented"]))
```