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

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

СИСТЕМА КЛАСИФІКАЦІЇ ЗАХВОРЮВАНЬ ДИХАЛЬНИХ ШЛЯХІВ НА ОСНОВІ РЕНТГЕНІВСЬКИХ ЗНІМКІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Яна ТРОФИМЕНКО

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент _____ Євген СІДЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Трофименко Яна Олександрівна
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система класифікації захворювань дихальних шляхів».

Керівник роботи: Сіденко Євген Вікторович, доцент кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент.
(прізвище, ім'я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2025 р. № 111.

2. Строк представлення кваліфікаційної роботи « 11 » червня 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: система класифікації захворювань дихальних шляхів на основі рентгенівських знімків із використанням нейронних мереж, з використанням набору даних NIH ChestX-ray14.

4. Перелік питань, що підлягають розробці:

– дослідження теоретичних засад використання нейронних мереж для класифікації захворювань дихальних шляхів за рентгенівськими знімками;

– аналіз існуючих архітектур глибокого навчання, що застосовуються для обробки медичних зображень;

– обґрунтування вибору інструментальних засобів для побудови моделі класифікації на основі згорткових нейронних мереж;

– підготовка та попередня обробка медичних зображень набору даних для навчання нейронної мережі;

– навчання та тестування моделі класифікації захворювань дихальних шляхів;

– оцінка ефективності розробленої.

5. Перелік графічних матеріалів: презентація, рисунки, таблиці.

Керівник роботи _____ ___Євген СІДЕНКО__
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ _Яна ТРОФИМЕНКО_
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «__20__» ___грудня___ 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Система класифікації захворювань дихальних шляхів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	Виконано
3	Огляд літературних джерел за темою використання нейронних мереж у медичній сфері та аналіз існуючих систем з використання нейромережевих технологій для рентгенівських знімків	31.01.2025	01.03.2025	Виконано
4	Огляд існуючих архітектур штучних нейронних мереж для для обробки медичних зображень	02.03.2025	01.04.2025	Виконано
5	Навчання нейронних мереж з аналізом отриманих результатів	02.04.2025	15.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	17.06.2025	Виконано

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Яна ТРОФИМЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 402 ЧНУ ім. Петра Могили

Трофименко Яни Олександрівни

на тему: **“СИСТЕМА КЛАСИФІКАЦІЇ ЗАХВОРЮВАНЬ ДИХАЛЬНИХ ШЛЯХІВ”**

Захворювання дихальних шляхів, зокрема пневмонія, ателектаз та пневмоторакс, залишаються одними з найпоширеніших причин смертності в усьому світі. Рання діагностика цих хвороб є вирішальним фактором для успішного лікування та запобігання ускладнень. Однак аналіз рентгенівських знімків легень є складною задачею, яка вимагає високої кваліфікації медичних працівників.

Об'єктом роботи є процеси класифікації захворювань дихальних шляхів.

Предметом роботи є архітектури нейронних мереж для класифікації захворювань дихальних шляхів.

Метою роботи є дослідження ефективності класифікації захворювань дихальних шляхів за рахунок створення інформаційної системи з використанням різних архітектур нейронних мереж.

У даній роботі досліджується застосування нейронних мереж для автоматизації процесу класифікації захворювань дихальних шляхів на основі рентгенівських знімків. Основна увага приділяється розробці та оптимізації архітектури згорткових нейронних мереж (CNN), які дозволяють виявляти ознаки хвороб з високою точністю. Очікується, що запропонована система допоможе медичним працівникам у швидкому та точному виявленні захворювань, що сприятиме зменшенню смертності.

Загальний обсяг роботи – 83 сторінки. Кваліфікаційна робота містить два додатока, 50 рисунків, 1 таблицю та 38 джерел посилання.

Ключові слова: захворювання дихальних шляхів, машинне навчання, згорткова нейронна мережа, модель нейронної мережі, класифікація.

ABSTRACT

to the qualification work by the student of the group 402 of Petro Mohyla Black Sea National University

Trofymenko Yana

"A CLASSIFICATION SYSTEM FOR RESPIRATORY TRACT DISEASES BASED ON X-RAY IMAGES"

Respiratory diseases, including pneumonia, atelectasis, and pneumothorax, remain among the most common causes of mortality worldwide. Early diagnosis of these diseases is a crucial factor for successful treatment and prevention of complications. However, the analysis of lung X-ray images is a complex task that requires highly qualified medical professionals.

The object of the work is the processes of classifying respiratory diseases.

The subject of the work is the architecture of neural networks for classifying respiratory diseases.

The aim of the work is to study the effectiveness of classifying respiratory diseases by creating an information system using different neural network architectures.

This work investigates the use of neural networks to automate the process of classifying respiratory diseases based on X-ray images. The main focus is on the development and optimization of the architecture of convolutional neural networks (CNN), which allow detecting signs of diseases with high accuracy. It is expected that the proposed system will help medical professionals in quickly and accurately detecting diseases, which will contribute to reducing mortality.

The total volume of the work is 83 pages. The qualification work contains two appendices, 50 figures, 1 table and 38 references.

Keywords: respiratory diseases, machine learning, convolutional neural network, neural network model, classification.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

CNN – convolutional neural network

ЗНМ – згорткова нейронна мережа

ReLU – rectified linear unit

CV – computer vision

ResNet – residual network

GPU – graphics processing unit

КТ – комп'ютерна томографія

МРТ – магнітно-резонансна томографія

ВСТУП

У сучасному світі, де машинне навчання та штучний інтелект знаходять дедалі ширше застосування, особливої уваги набувають методи автоматизованої діагностики захворювань. Захворювання дихальних шляхів, зокрема пневмонія, ателектаз та пневмоторакс, залишаються одними з найпоширеніших причин смертності у всьому світі. Їхня рання діагностика є критично важливим фактором для успішного лікування та зниження рівня летальних випадків.

Аналіз рентгенівських знімків грудної клітини є складним завданням, що потребує значного досвіду та високої кваліфікації лікарів. Однак розвиток технологій глибокого навчання, зокрема згорткових нейронних мереж (CNN), відкриває можливості для створення ефективних автоматизованих систем діагностики. Такі системи здатні аналізувати великі обсяги медичних зображень та виявляти патології з високою точністю, що може значно полегшити роботу медичних працівників та підвищити якість діагностики.

Об'єктом роботи є процеси класифікації захворювань дихальних шляхів.

Предметом роботи є архітектури нейронних мереж для класифікації захворювань дихальних шляхів.

Метою роботи є дослідження ефективності класифікації захворювань дихальних шляхів за рахунок створення інформаційної системи з використанням різних архітектур нейронних мереж.

Дана робота присвячена дослідженню можливостей застосування нейронних мереж для класифікації захворювань дихальних шляхів на основі рентгенівських знімків. Очікується, що результати дослідження сприятимуть розвитку автоматизованих систем діагностики, що допоможе зменшити навантаження на лікарів, підвищити точність діагностування та покращити загальну якість медичного обслуговування.

Для досягнення поставленої мети необхідно виконати наступні етапи дослідження:

– аналіз сучасних методів машинного навчання;

– вибір та підготовка навчального набору даних;
– розробка та навчання моделей нейронних мереж;
– оцінка продуктивності моделей та їх порівняльний аналіз;
– впровадження покращених підходів до класифікації;
– висновки та рекомендації щодо використання запропонованих моделей у медичній практиці.

1 АНАЛІЗ ПРОБЛЕМИ КЛАСИФІКАЦІЇ ЗАХВОРЮВАНЬ ДИХАЛЬНИХ ШЛЯХІВ

1.1 Методи діагностики захворювань дихальної системи

Діагностика захворювань дихальної системи є дуже складним та багаторівневим процесом, який основується на клінічному обстеженні пацієнта, аналізі його скарг, даних анамнезу і використанні різноманітних лабораторних методів дослідження [1]. Добре побудований діагностичний алгоритм може дозволити не лише встановити наявність патологічного процесу, а й визначити його характер, стадію та рівень ураження органів.

Одним із основних етапів діагностики є фізичний метод обстеження, який передбачає огляд, пальпацію, постукування та прослуховування. Під час обстеження лікар звертає увагу на наявність задухи, зміни контуру грудей та опорою на додаткові м’язи для дихання. Перкуторний звук і характер дихальних шумів під час прослуховування допомагають здогадатися про наявність патологічних процесів у легенях або області грудей [1]. Однак фізичне обстеження часто має обмежену чутливість і специфічність, тому вимагає перевірки за допомогою подальших наукових досліджень.

Лабораторна діагностика також займає важливе місце у виявленні запальних процесів, інфекційних агентів, а також у визначенні загального стану організму. Вона включає загальний аналіз крові, біохімічні дослідження, тести на проби слизу, скринінг на конкретні антитіла або мікроби та тести на алергію, якщо існує підозра на алергічні захворювання [2]. Вимірювання рівня кисню в крові та оцінка газового складу артеріальної крові допомагають діагностувати порушення газообміну.

Незважаючи на це вирішальне значення у діагностиці захворювань дихальної системи відіграють інструментальні методи, серед яких особливу увагу приділяють рентгенологічному дослідженню. Широке використання такого підходу пояснюється його легким доступом, швидким процесом дослідження та доступною ціною, що робить його важливим на початку оцінки

пацієнтів, які можуть мати проблеми з легенями. Подібне дослідження дозволяє швидко оцінити стан легеневої тканини, виявити ознаки запальних, дегенеративних, пухлинних або обструктивних процесів [3].

При оцінці рентгенівського знімка лікар-радіолог звертає увагу на кілька ключових аспектів [3]. Зазвичай рентгенограми аналізуються з урахуванням таких параметрів:

– інтенсивність та характер затемнень (вогнищеві, дифузні, інфільтративні);

– деформація легеневого малюнка (посилення, ослаблення, розмитість);

– зміщення органів середостіння (може вказувати на пневмоторакс або пухлинний процес);

– стан діафрагми (високе стояння, згладженість, нерівномірність).

Рентгенівські візуалізації мають вирішальне значення для раннього виявлення шкідливих захворювань, таких як туберкульоз та рак легенів. При туберкульозі можна спостерігати одну або кілька локалізованих змін, кальцифікатів або порожнин. Онкологічні захворювання характеризуються щільними вузловими утвореннями, які можуть мати нерівні контури [4].

Крім того, рентген має вирішальне значення для виявлення раптових проблем зі здоров’ям, таких як рідина в грудній порожнині або пневмоторакс. Наприклад, при пневмотораксі спостерігається відсутність легеневого малюнка в певній зоні, а плевральний випіт проявляється горизонтальним рівнем рідини [3].

Рентгенографія має кілька ключових переваг: доступність, швидкість проведення та невисока вартість. Це дозволяє швидко отримати доступ до даних про оцінку здоров'я, особливо під час невідкладних медичних ситуацій. Проте існують і обмеження. Наприклад, незначні відхилення можуть залишатися непоміченими, оскільки техніці не вистачає ясності. Крім того, рентген надає плоску картину, яка може ускладнити розказування різних хвороб.

У складних ситуаціях для кращого обстеження застосовуються додаткові методи, такі як комп’ютерна томографія (КТ) або магнітно-резонансна томографія

(МРТ) [2]. Тим не менш, рентгенівські промені є вирішальним початковим кроком у виявленні легеневих хвороб завдяки своїй ефективності та простоті.

Зазвичай ідентифікація легеневих захворювань покладається на поєднання фізичних оглядів, лабораторних тестів та різних інструментів, при цьому рентгенівські промені є ключовим та загальним методом для первинної візуалізації патологій органів грудної клітки [1]. Саме завдяки рентгенівським знімкам можливо швидко, доступно та досить інформативно оцінити стан дихальної системи, що є основою для подальшого діагностичного та лікувального планування.

1.2 Основи глибокого навчання та згорткових нейронних мереж

Глибоке навчання – це підрозділ машинного навчання, що використовує штучні нейронні мережі великої глибини для аналізу даних [5]. Воно імітує спосіб, яким людський мозок обробляє інформацію, що дозволяє алгоритмам зрозуміти складні зв’язки між інформацією за допомогою багаторівневої обробки. Основна ідея глибокого навчання полягає у поступовому виділенні ознак на різних рівнях складності, що дозволяє системі автоматично визначати вирішальні дані для прийняття рішень.

1.2.1 Поняття та принципи глибокого навчання

На відміну від традиційних методів машинного навчання, які часто потребують ознак виставлених людиною, глибокі нейронні мережі здатні самостійно виявляти важливі характеристики даних на різних рівнях складності. Це робить використання глибокого навчання надзвичайно ефективним для обробки великих і складних наборів даних, таких як зображення, звук, текст та відео.

Основна ідея глибокого навчання передбачає створення багатоярусної системи для обробки інформації, де кожен наступний шар нейронної мережі витягує дедалі складніші та абстрактніші ознаки на основі даних, отриманих з попереднього шару [6]. У аналізі зображень початкові шари ідентифікують

основні компоненти, такі як лінії та кути, проміжні шари виявляють більш складні форми або предмети, а заключні шари розуміють загальні ідеї, такі як ідентифікація певного предмета.

Навчання глибоких нейронних мереж відбувається за допомогою оптимізації функції втрат (loss function), яка відображає наскільки далеко передбачення моделі від фактичного результату [7]. Для оновлення ваг мережі використовується алгоритм зворотного поширення помилки (backpropagation) у поєднанні з методами оптимізації, такими як стохастичний градієнтний спуск (SGD) або його сучасні модифікації (наприклад, Adam, RMSprop) [7]. У процесі навчання мережа поступово мінімізує функцію втрат, покращуючи свої прогнози з кожною ітерацією.

Однією з важливих характеристик глибокого навчання є здатність працювати із сирими (неструктурованими) даними без необхідності їх попередньої обробки. Це надає можливість використання великих обсягів даних, що надходять безпосередньо зі сканерів, сенсорів, камер або інших джерел.

Крім того, для успішного функціонування глибокі нейронні мережі зазвичай потребують великої кількості навчальних даних та значних обчислювальних потужностей. У цьому контексті їх зростання було тісно пов'язане з просуванням апаратного забезпечення, особливо з зростанням ємності графічного процесора (GPU) та еволюцією хмарних обчислень.

Глибоке навчання сьогодні є основою для багатьох сучасних технологій, включаючи автоматичне розпізнавання мови, комп'ютерний зір, машинний переклад, системи рекомендацій, автономне водіння та,звичайно, медичну діагностику на основі аналізу складних медичних даних [8].

1.2.2 Архітектура штучних нейронних мереж

Архітектура штучної нейронної мережі базується на базі трьох основних типів шарів: вхідного шару, прихованих шарів і вихідного шару. Кожен із цих шарів виконує певні функції у процесі обробки інформації та навчання моделі.

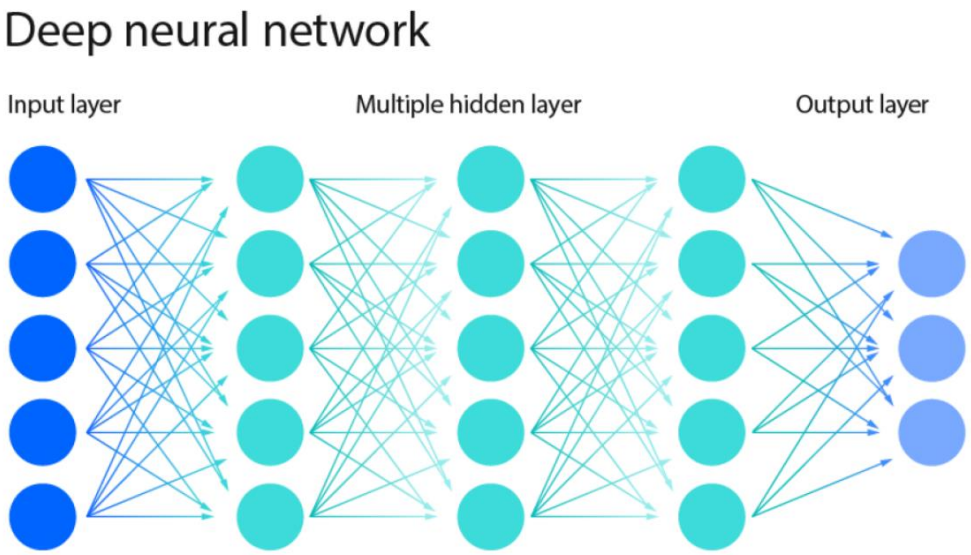

Рисунок 1.1 – Типи шарів штучної нейронної мережі [9]

Вхідний шар є першим шаром нейронної мережі і відповідає за прийом даних. На цьому етапі кожен нейрон вхідного шару представляє одну ознаку (або характеристику) вхідного вектора даних [7]. Наприклад коли мережа вивчає зображення, яке становить 28 на 28 пікселів, перший шар матиме 784 нейрони, кожен з яких отримує значення яскравості відповідного пікселя.

Приховані шари розташовані між вхідним і вихідним шарами і є основним елементом, що забезпечує складність і потужність нейронної мережі. Кожен нейрон прихованого шару отримує сигнали з попереднього шару, обробляє їх із використанням вагових коефіцієнтів та надсилає результати обрахунку до наступного шару. У прихованих шарах відбувається поступова трансформація і виділення якихось ознак для передбачення результату із початкових даних [7]. Кількість прихованих шарів і нейронів у них визначає «глибину» нейронної мережі. Саме завдяки багатьом прихованим шарам мережа може навчитися розпізнавати дуже складні залежності та виявляти закономірності у даних.

Вихідний шар є останнім у нейронній мережі й відповідає за формування фінального передбачення від роботи моделі. Структура вихідного шару залежить від типу завдання, яке вирішується: у задачах класифікації кількість нейронів вихідного шару зазвичай співпадає з кількостю класів, у задачах регресії вихідний

шар може містити один нейрон, що видає числове значення.

Кожен нейрон наступного шару отримує на вхід сигнали від нейронів попереднього шару. Ці вхідні сигнали множаться на відповідні вагові коефіцієнти – числові значення, які відображають силу та важливість кожного зв'язку. Вага визначає, наскільки сильно один нейрон впливає на інший: чим вища вага, тим більший вплив має сигнал, і навпаки.

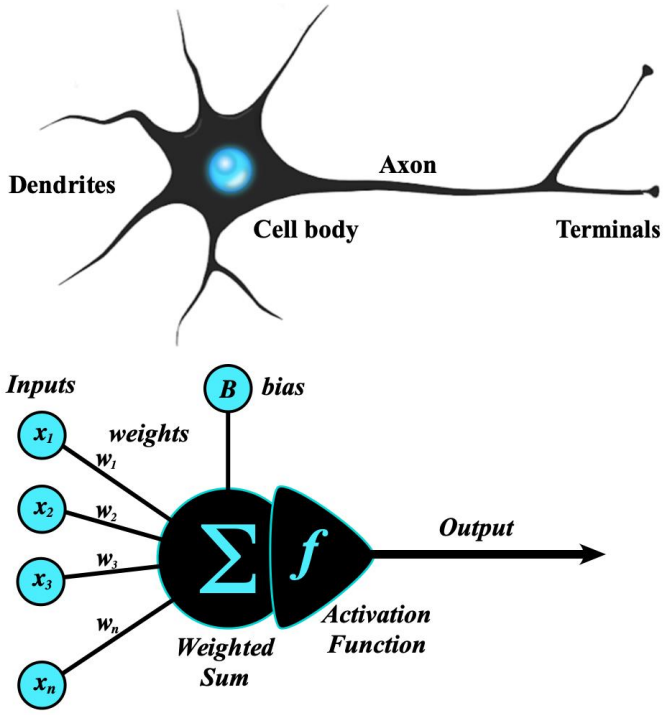

Рисунок 1.2 – Структура та принцип роботи нейрона [10]

Після множення сигналів на ваги всі отримані значення сумуються, і до цієї суми додається зміщення (bias) – додатковий параметр, який допомагає моделі зміщувати функцію активації і краще пристосовуватися до даних. Отриманий результат передається через функцію активації, яка визначає вихідне значення нейрона. Таким чином, кожен нейрон приймає рішення, чи потрібно активуватися (передавати сигнал далі), і наскільки сильним повинен бути цей сигнал.

Основна мета функції активації - перетворення суми зважених входів з нейронів у значення для наступного шару [7]. Вибір певної функції активації суттєво впливає на загальну ефективність мережі, її здатність до навчання та

точність прогнозів.

Існує багато різних функцій активації, кожна з яких має свої особливості та сфери застосування.

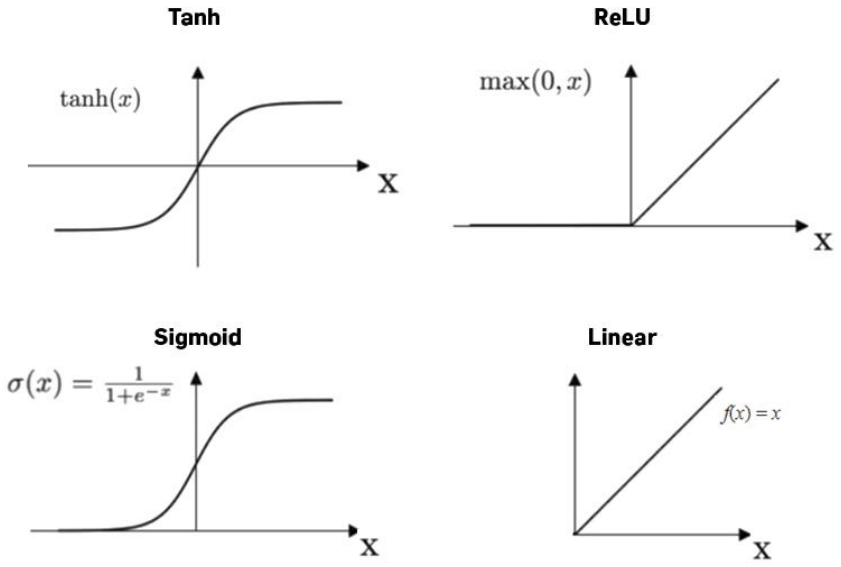

Рисунок 1.3 – Основні типи функцій активації [11]

Процес передачі сигналу від вхідного шару до вихідного через всі приховані шари називається прямим проходженням (forward propagation). Саме під час цього процесу нейронна мережа робить свої прогнози, аналізуючи нові дані.

На етапі навчання нейронної мережі вагові коефіцієнти та зміщення оновлюються так, щоб зменшити різницю між фактичним виходом мережі та правильним результатом. Для цього застосовується метод зворотного поширення помилки (backpropagation), який обчислює, як потрібно змінити кожну вагу, спираючись на похідну функції втрат [12].

Глибина нейронної мережі визначається кількістю шарів, через які проходять дані від вхідного шару до вихідного. Чим більше прихованих шарів має мережа, тим більш «глибокою» вона вважається. Глибина мережі є критичним фактором, який безпосередньо впливає на її здатність виявляти складні залежності та вирішувати непрості задачі.

Глибокі нейронні мережі мають перевагу в тому, що вони можуть будувати

багаторівневі представлення даних. На перших шарах мережа може виявляти прості ознаки, як контури якоїсь фігури на зображенні, на наступних шарах – більш складні патерни, наприклад, частини об'єктів, а на фінальних вже цілісні об'єкти [12]. Завдяки такій багаторівневій обробці даних мережа стає здатною вирішувати завдання, які важко або навіть нереально розв'язати простими моделями.

Однак збільшення глибини мережі може також створювати певні труднощі. По-перше, глибші мережі вимагають більше обчислювальних ресурсів для тренування та прогнозування, що також може потребувати і багато часу на навчання. По-друге, зростає ризик виникнення проблеми затухання або вибуху градієнтів, коли зміни ваг у початкових шарах стають занадто незначними або завеликими, що ускладнює навчання. Для вирішення цих проблем було розроблено спеціальні техніки, зокрема нормалізація пакетів (Batch Normalization), спеціальні функції активації та архітектури з прямими з’єднаннями між шарами (наприклад, мережі типу ResNet).

1.2.3 Згорткові нейронні мережі

Згорткові нейронні мережі (ЗНМ), відомі також як Convolutional Neural Networks (CNN), являють собою специфічний різновид штучних нейронних мереж, що найбільше застосовується для аналізу структурованих даних, наприклад, зображень чи відео [12]. Ключовою концепцією згорткових мереж є їх здатність автоматично визначати важливі ознаки з вхідних даних за допомогою спеціальних фільтрів, які самостійно удосконалюються в процесі навчання.

На відміну від традиційних нейронних мереж, де кожен нейрон сполучений з усіма нейронами попереднього рівня, у згорткових мережах нейрони з'єднані лише з невеликим фрагментом вхідних даних. Цей принцип називається локальним зв'язком та значно зменшує кількість параметрів у мережі, що робить її більш продуктивною при роботі з великими зображеннями.

Принцип роботи згорткової мережі полягає у використанні операції згортки. Маленький фільтр (або згорткове ядро) методично ковзає по всій вхідній

картинці, визначаючи локальні властивості, на кшталт контурів, кутів чи візерунків текстури. Кожен фільтр налаштований на розпізнавання певного типу характеристик, що сприяє створенню розуміння даних на різноманітних рівнях узагальнення. Після обробки фільтрами, виявлені характеристики передаються далі через мережу для глибшого аналізу.

Типова структура згорткової нейронної мережі (CNN) містить послідовність шарів, кожен з яких відіграє власну роль у перетворенні вхідних даних на високорівневе представлення, що підходить для класифікації, сегментації чи іншого поставленого завдання. Архітектура CNN має ієрархічну структуру, що дозволяє моделі поступово виокремлювати все складніші особливості.

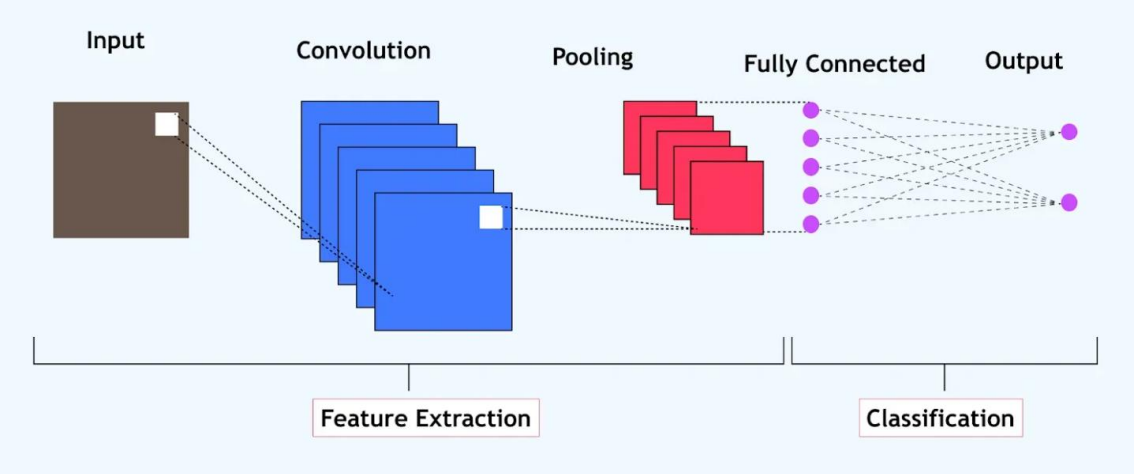

Рисунок 1.4 – Архітектура згорткових нейронних мереж [13]

Першим етапом в архітектурі згорткових нейронних мереж є вхідний шар, який приймає необроблені дані – наприклад, зображення у вигляді тривимірного тензора (висота × ширина × кількість каналів) [12]. Ці дані надсилаються до згорткових шарів, котрі виконують згорткові операції за допомогою фільтрів, котрі підлягають навчанню. Кожен такий шар формує набір карт ознак, що представляють конкретні властивості вхідного зображення, такі як краї, кути, контури або текстури.

Після одного чи кількох згорткових шарів зазвичай розміщуються шари субдискретизації (пулінгу), які зменшують просторову розмірність карт ознак,

водночас зберігаючи ключову інформацію. Це не лише зменшує обчислювальні ресурси, але й знижує ймовірність перенавчання, оскільки модель концентрується на найважливіших частинах даних.

Зазвичай у CNN застосовують декілька модулів, кожен з яких містить згортковий рівень, функцію активації (скажімо, ReLU) та шар пулінгу [12]. Ці модулі дають можливість поступово трансформувати зображення: від базових ознак до більш узагальнених образів. Зі збільшенням глибини мережі, виокремлені властивості стають все складнішими, охоплюючи не тільки структуру об'єкта, але й його форму, компоненти, або ж контекст.

На завершальних стадіях мережа переходить до шарів з повним зв'язком, що відповідають за класифікацію на основі розпізнаних ознак. Тут кожен нейрон сполучений з усіма нейронами попереднього шару, що дає мережі змогу зібрати всю узагальнену інформацію для прийняття рішення. На кінцевому етапі архітектури, як правило, міститься вихідний шар, що визначає фінальний результат, наприклад, прогнозування класу за допомогою функції softmax.

Залежно від поставленої задачі, може варіюватися кількість блоків згортки, типи фільтрів, величина вікон пулінгу та інші показники. Однак, базова структура архітектури, яка включає почергове використання згорткових і пулінгових шарів з подальшою класифікацією, залишається характерною для більшості CNN.

Згорткові нейронні мережі (CNN) мають ряд суттєвих переваг, що сприяло їхньому перетворенню на ключовий елемент для сучасного комп'ютерного зору (CV). Вони здійснюють автоматичне вилучення ключових характеристик з даних, усуваючи потребу в ручному налаштуванні, при цьому зберігаючи просторову незалежність та оптимально задіюючи параметри через локальні зв'язки і спільне використання вагових коефіцієнтів. Це забезпечує високу точність у розв'язанні задач класифікації, сегментації та розпізнавання об'єктів.

Водночас, CNN не позбавлені недоліків. Ефективне навчання часто потребує значних обсягів розмічених даних, а також потужних обчислювальних ресурсів для їх обробки. Крім того, ці моделі важко піддаються інтерпретації, що створює проблеми у критичних галузях, як-от медицина, де критично важливо

розуміти, на підставі чого було прийнято те чи інше рішення.

1.3 Використання нейромереживих технологій у медицині

Протягом останнього десятиліття нейромережеві технології інтегрувалися в авангард інноваційного руху в медичній сфері. Стрімкий розвиток обчислювальних потужностей, алгоритмів машинного навчання та легкий доступ до великих масивів медичних відомостей дали поштовх до активного використання штучних нейронних мереж у діагностиці, прогнозуванні та терапевтичних підходах. Їхня здатність розпізнавати складні патерни в інформації відкриває нові обрії точності й результативності в клінічному просторі.

Нейромережі, зокрема згорткові нейронні мережі (CNN), отримали широке застосування в аналізі медичних зображень, таких як рентгенівські знімки, КТ, МРТ та гістологічні зразки [6]. Завдяки автоматизованому виділенню ключових характеристик, ці моделі дозволяють знизити суб'єктивність оцінок, прискорити діагностику та поліпшити якість виявлення патологій на ранніх етапах. До того ж, нейронні мережі використовуються в персоналізованій медицині, прогнозуванні перебігу хвороб, аналізі електронних медичних записів та оптимізації процесів лікування.

У статті "COVID-19 classification of X-ray images using deep neural networks" [14] дослідники представили модель глибокого навчання для автоматичного виявлення COVID-19 на рентгенограмах грудної клітки. Використовуючи ансамбль нейронних мереж (ResNet та VGG), автори досягли високих показників точності (понад 90%) при класифікації зображень як позитивних або негативних щодо COVID-19. Перед обробкою зображення проходили сегментацію легень і аугментацію, що допомогло моделі краще узагальнювати інформацію.

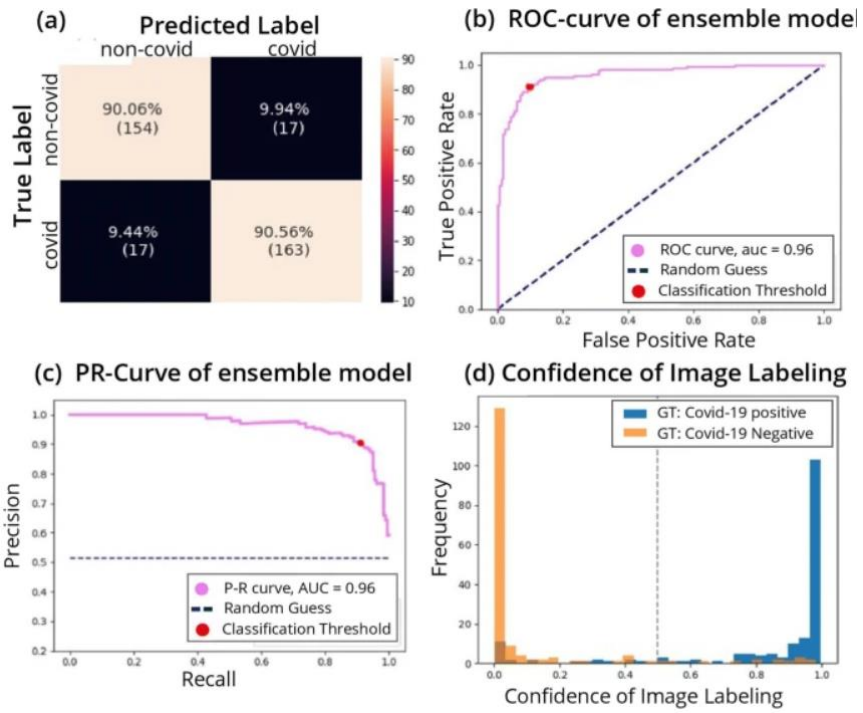

Рисунок 1.5 – Результати навчання мережі для виявлення COVID-19 [14]

Модель також реалізувала функцію пошуку схожих зображень, що може бути корисним у клінічній практиці. Однак автори наголошують на обмеженнях: навчання проводилося на даних з одного регіону, що може впливати на універсальність результатів. Крім того, модель є малопояснюваною, що обмежує її впровадження в лікарській практиці.

У статті "Deep Convolutional Neural Networks for Endotracheal Tube Position and X-ray Image Classification: Challenges and Opportunities" [15] (Journal of Digital Imaging, 2017) досліджується застосування глибоких згорткових нейронних мереж (DCNN) для автоматичної класифікації рентгенівських зображень, зокрема для виявлення наявності та правильності розташування ендотрахеальної трубки (ET) на грудних рентгенограмах.

Автори використовували як попередньо навчені, так і ненавчені моделі, зокрема AlexNet та GoogLeNet, для трьох завдань класифікації: розрізнення грудних і абдомінальних рентгенограм, виявлення наявності або відсутності ET та визначення її положення (нормальне чи занадто низьке). Для покращення навчання застосовували аугментацію даних.

Результати показали, що для простих завдань, таких як розрізнення грудних і абдомінальних зображень, моделі досягли ідеальної точності (AUC = 1.00), навіть з обмеженим навчальним набором. Для складніших завдань, таких як виявлення наявності ET, моделі продемонстрували високу точність (AUC = 0.99). Однак при визначенні неправильного положення ET точність знизилася (AUC = 0.81), що вказує на складність цього завдання для автоматичних систем.

У статті “Convolutional Neural Network Based Classification of Chest X-ray Images Using Transfer Learning” [16] (ASTESJ, 2020) автори досліджують ефективність згорткових нейронних мереж із перенесеним навчанням для автоматичної класифікації рентгенівських зображень грудної клітки. Основна мета – автоматично розпізнавати пневмонію та інші патології з CXR-зображень, використовуючи моделі глибокого навчання.

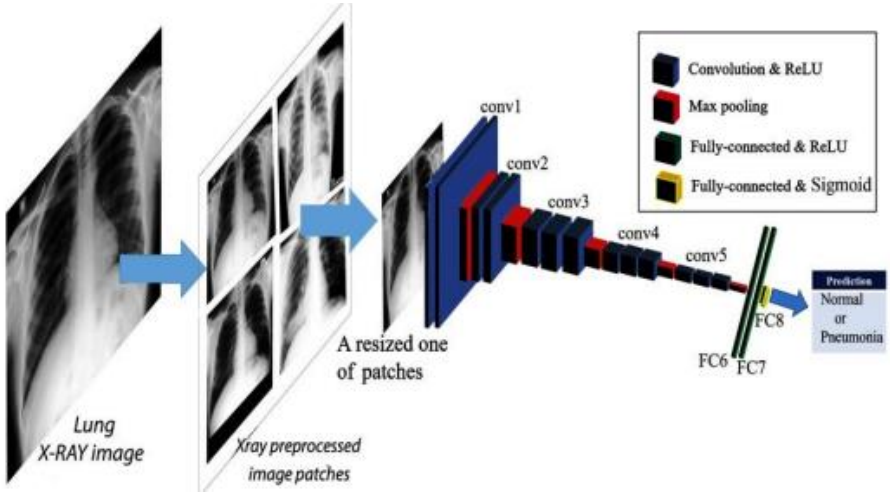

Рисунок 1.6 – Архітектура нейромережі використана для виявлення пневмонії [16]

Для реалізації було використано популярні архітектури, зокрема VGG16, VGG19, InceptionV3 та ResNet50, попередньо навчені на ImageNet. Після тонкого налаштування на медичному датасеті, найкращі результати показала ResNet50 з точністю 96%, що свідчить про високу придатність методів трансферного навчання у цій галузі.

У статті "A deep learning architecture for multi-class lung diseases classification" [17] (Journal of King Saud University – Computer and Information

Sciences, 2022) представлено архітектуру глибокого навчання для багатокласової класифікації захворювань легень, зокрема пневмонії, раку легень, туберкульозу, затемнень легень та COVID-19.

Автори запропонували спеціально розроблену CNN-архітектуру для класифікації п’яти типів легеневих захворювань. Модель була протестована на зібраному датасеті, що включав 6000 зображень, розподілених між класами: нормальні легені, пневмонія, туберкульоз, рак легень і COVID-19. Запропонована модель досягла загальної точності 96.7%.

У дослідженні "Classification of Images of Childhood Pneumonia using Convolutional Neural Networks" [18] (BIOSTEC 2019) автори досліджують застосування згорткових нейронних мереж (CNN) для автоматичної діагностики пневмонії у дітей за рентгенівськими зображеннями грудної клітки. Використовуючи відкритий датасет Chest X-Ray Images for Classification (Kermany et al., 2018), що містить 5863 зображення, дослідники розділили дані на дві категорії: "норма" та "пневмонія".

Автори розробили згорткову нейронну мережу (CNN), спеціально адаптовану до задачі класифікації рентгенівських зображень грудної клітки у дітей. Їхня архітектура була створена з нуля – це не була переднавчена модель на зразок ResNet чи VGG, а натомість власна легка CNN, зосереджена на простоті та високій продуктивності при обмежених обчислювальних ресурсах. Результати показали, що запропонована модель досягла середньої точності 95.3%, перевершуючи попередні дослідження, де точність становила 92.8%.

1.4 Постановка задачі

Актуальність теми дослідження зумовлена широким розповсюдженням захворювань органів дихання та необхідністю їх своєчасної діагностики, що часто базується на аналізі рентгенівських знімків. У багатьох регіонах, особливо з обмеженим доступом до кваліфікованих лікарів, ефективна інтерпретація рентгенограм залишається складним завданням. Тому автоматизація процесу діагностики за допомогою методів штучного інтелекту, зокрема згорткових

нейронних мереж (CNN), є перспективним напрямом, який може підвищити точність та швидкість постановки діагнозу.

Об’єктом дослідження є процес автоматичної класифікації рентгенівських зображень органів грудної клітки.

Предметом дослідження є застосування методів глибокого навчання, зокрема згорткових нейронних мереж, для виявлення патологій дихальної системи на рентгенівських знімках.

Метою роботи є аналіз ефективності сучасних архітектур згорткових нейронних мереж для задачі класифікації захворювань дихальних шляхів за зображеннями рентгенограм, з подальшою підготовкою до реалізації відповідного інтелектуального програмного рішення.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– дослідження теоретичних матеріалів та аналіз існуючих досліджень пов’язаних з обраною тематикою;

– огляд та дослідження технологій глибокого навчання в сфері медичної діагностики;

– використання різних типів архітектур нейронної мережі для задачі класифікації легеневих хвороб на арентгенівських знімках;

– порівняння ефективності використання різних типів архітектур нейронної мережі для потавленої задачі.

Висновки до розділу 1

У цьому розділі було надано загальний опис проблеми класифікації захворювань дихальних шляхів, зокрема на основі рентгенівських знімків органів грудної клітки. Розкрито теоретичні основи глибокого навчання та згорткових нейронних мереж. Проаналізовано способи застосування нейромережевих технологій у медицині. Також було сформульовано мету, об’єкт і предмет дослідження, визначено конкретні завдання, які необхідно виконати для досягнення поставленої мети.

2 АНАЛІЗ ОБРАНИХ ТЕХНОЛОГІЙ ДЛЯ КЛАСИФІКАЦІЇ ЗАХВОРЮВАНЬ ДИХАЛЬНИХ ШЛЯХІВ

2.1 Вибір набору даних для задачі класифікації

Для вирішення задачі класифікації медичних зображень ключовим є вибір якісного, збалансованого і надійного набору даних. Досить відомим і часто використовуваним у дослідженнях у галузі медичної візуалізації є NIH ChestX-ray14, розроблений Національним інститутом здоров'я США (National Institutes of Health) [19].

Цей набір даних містить понад 112 тисяч рентгенівських знімків грудної клітки, що належать понад 30 тисячам пацієнтів. Для кожного зображення зазначено, чи присутня одна або декілька із 14 патологій, зокрема пневмонія, плевральний випіт, кардіомегалія, легеневі інфільтрати, фіброз, пневмоторакс, пухлини тощо. Також кожен знімок супроводжується метаданими, включаючи вік, стать пацієнта та положення тіла при зйомці (наприклад, PA або AP проєкція).

Набір NIH ChestX-ray14 добре знаний як один із найбільших та найбільш стандартизованих для глибинного навчання, адже всі рентгенівські знімки зроблені в межах однієї медичної установи, з застосуванням стандартного обладнання. Це мінімізує коливання даних, спричинені різноманітними джерелами або типами пристроїв. До того ж, мультикласове маркування дозволяє не тільки вирішувати задачу бінарної класифікації (норма/патологія), але й виконувати мультимітку, коли для одного зображення можливо визначити кілька діагнозів одночасно.

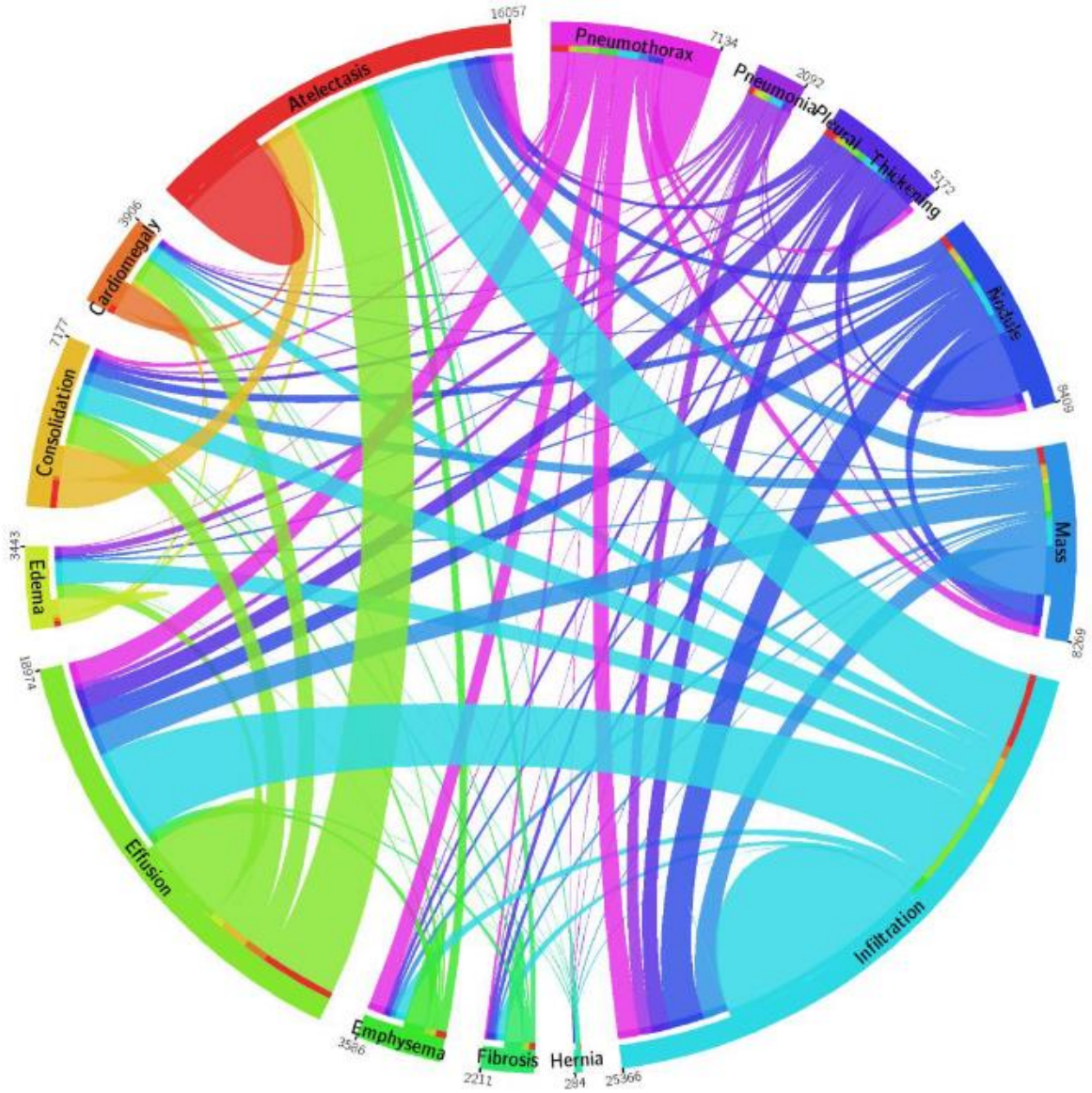

Рисунок 2.1 – Статистика розподілу 14 категорій захворювань у наборі даних [19]

Маркування патологій було автоматизовано на основі аналізу текстових медичних записів, що потенційно могло призвести до деяких похибок у визначенні класів. Не зважаючи на це, завдяки великому обсягу зображень, великій кількості різних патологій та широкому використанню у науковому співтоваристві, цей набір лишається стандартом для оцінки нейромережевих моделей в сфері медичної діагностики.

2.2 Архітектури нейронних мереж, що використовуються для класифікації медичних зображень

Класифікація медичних зображень представляє собою надзвичайно складне завдання, що вимагає найвищої точності, безвідмовності та здатності розпізнавати навіть незначні відхилення. З появою глибокого навчання, зокрема згорткових нейронних мереж (CNN), відкрилася можливість автоматизувати аналіз складних візуальних моделей, що присутні на таких зображеннях, як рентгенівські знімки, комп'ютерна томографія, магнітно-резонансна томографія та ультразвукові дослідження. Водночас, традиційні CNN-архітектури не завжди демонструють необхідну ефективність на великих або глибоких моделях, через що виникла необхідність у розробці складніших та оптимізованих архітектур [20].

Серед найбільш продуктивних і широко застосовуваних у сфері медичної візуалізації слід виділити архітектури ResNet, DenseNet та EfficientNet. Кожна з них характеризується унікальними властивостями, що дають змогу досягти кращих результатів за рахунок глибшої структури, ефективного використання параметрів або оптимізації обчислювальних ресурсів.

2.2.1 ResNet

Архітектура ResNet (Мережі з залишковими блоками) була запропонована, щоб вирішити складність навчання надзвичайно глибоких нейронних мереж [25]. Один з ключових викликів при збільшенні глибини мережі – це проблема "вибуху" або "згасання" градієнта, що негативно впливає на ефективність навчання. Ця проблема була подолана завдяки використанню залишкових зв'язків (residual connections), які створюють прямий шлях для градієнтів під час зворотного поширення.

Основна концепція ResNet полягає у додаванні "залишкового" зв'язку в кожному блоці мережі, що дозволяє інформації обходити шар практично без змін. Завдяки цьому, кожен блок може "навчатися" моделі, зосереджуючись на різниці між вхідними та вихідними даними. Це сприяє кращому збереженню інформації мережею, навіть при великій глибині. Залишкові зв'язки роблять процес навчання

глибших моделей набагато простішим, значно зменшуючи проблему деградації точності, яка часто виникає в глибоких мережах.

Архітектура ResNet побудована на основі численних залишкових блоків. Кожен з цих блоків складається з двох або більше згорткових шарів, між якими інтегрований залишковий зв'язок. Така структура дозволяє мережі ефективно навчатися на глибших шарах, при цьому зберігаючи ключову інформацію, що передається через мережу. Завдяки цим залишковим зв'язкам, що дають можливість зберігати інформацію навіть після численних трансформацій, модель досягає кращих результатів у розпізнаванні складних особливостей зображень.

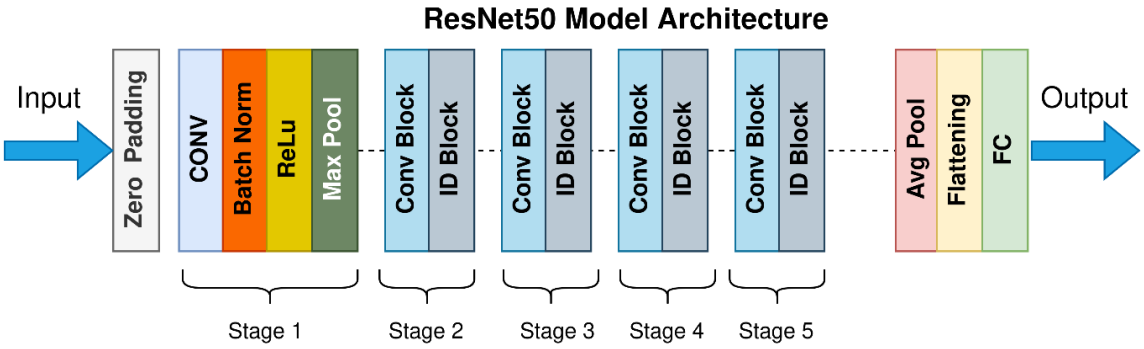

Рисунок 2.2 – Архітектура ResNet50 [25]

Завдяки здатності долати проблему згіршення ефективності у глибоких мережах, ResNet набула статусу ключової архітектури у багатьох завданнях комп'ютерного зору (CV), зокрема при класифікації медичних зображень. Моделі ResNet демонструють високу точність, навіть працюючи з надзвичайно глибокими мережами, що робить їх потужним інструментом для аналізу зображень.

Характеристики ResNet також відкривають можливості для трансферного навчання, де попередньо навчена мережа, яка вже обробила великий набір даних, може бути пристосована до конкретних медичних потреб з мінімальною кількістю додаткових етапів навчання. Це критично важливо в медичній галузі, де доступ до великих, розмічених даних часто обмежений.

2.2.2 DenseNet

DenseNet (Густо З'єднані Згорткові Мережі) – це еволюція ідеї вдосконалення глибоких згорткових мереж, але з застосуванням ще тіснішої взаємодії між шарами [26]. Ключова риса архітектури DenseNet полягає у щільному зв'язку шарів: кожен шар використовує на вході не лише вихід попереднього шару, а й виходи з усіх попередніх шарів. Цей спосіб сприяє повторному використанню ознак, що відчутно покращує ефективність мережі, як в аспекті точності, так і в економії параметрів.

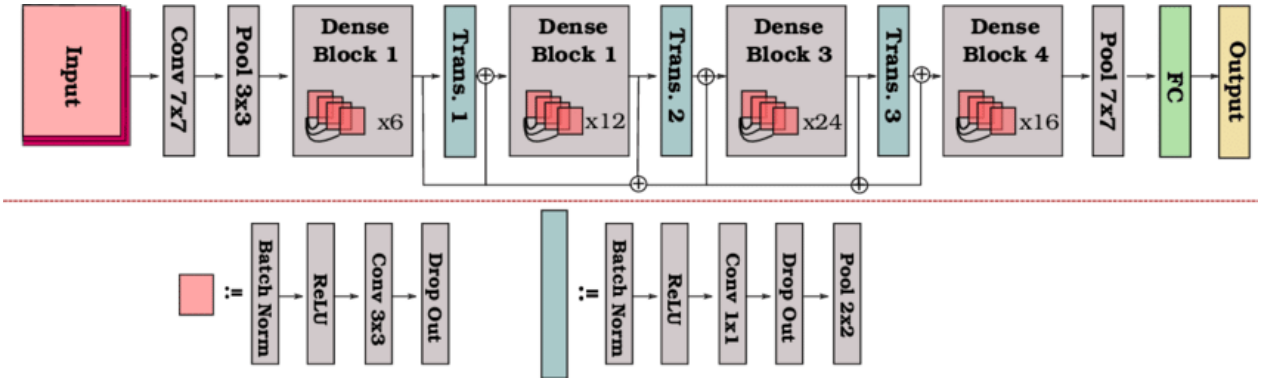

Рисунок 2.3 – Архітектура DenseNet-121 [26]

Завдяки міцним з'єднанням, DenseNet допомагає уникнути повторів вже вивчених рис та полегшує проходження градієнтів назад, що позитивно впливає на процес навчання. Це є особливо важливим під час роботи з медичними зображеннями, де кожен, навіть незначний, елемент може мати діагностичне значення. Крім того, цей підхід сприяє глибшому розповсюдженню інформації через мережу, що покращує виявлення складних і тонких структур на зображеннях.

Кожна група щільно з'єднаних шарів у DenseNet іменується щільним блоком, між якими розташовуються перехідні шари (transition layers), котрі зменшують розмірність та запобігають надмірному збільшенню обчислювальних ресурсів. Незважаючи на велику кількість з'єднань, DenseNet показує себе економічнішою за кількістю параметрів у порівнянні з традиційними мережами,

оскільки не потребує значної кількості фільтрів на кожному шарі.

DenseNet продемонструвала відмінні результати в медичних задачах, таких як автоматична діагностика пневмонії на рентгенівських знімках, виявлення пухлин, сегментація анатомічних структур та інші. Її здатність працювати з меншими обсягами даних, ефективно навчаючись навіть на складних наборах, робить її привабливою архітектурою для медичних застосувань, де обсяг доступних даних часто обмежений.

2.2.3 EfficientNet

EfficientNet – це сучасна архітектура для глибоких нейронних мереж, що була створена для досягнення найвищої точності з мінімальними обчислювальними витратами [27]. Ключовою концепцією є збалансоване масштабування трьох основних параметрів нейронної мережі: глибини, ширини та розмірності вхідного зображення. Раніше моделі масштабувались, зазвичай, лише одним із цих параметрів, що вело до неефективного використання ресурсів. EfficientNet розв'язує цю проблему, застосовуючи єдиний коефіцієнт масштабування, який оптимально поєднує всі три параметри.

Архітектура базується на простішій моделі під назвою EfficientNet-B0, яка була створена автоматизованим способом за допомогою AutoML – інструменту для автоматичного проєктування архітектур. Далі ця базова модель масштабувалася до більших версій (B1–B7), зберігаючи баланс між продуктивністю та ефективністю.

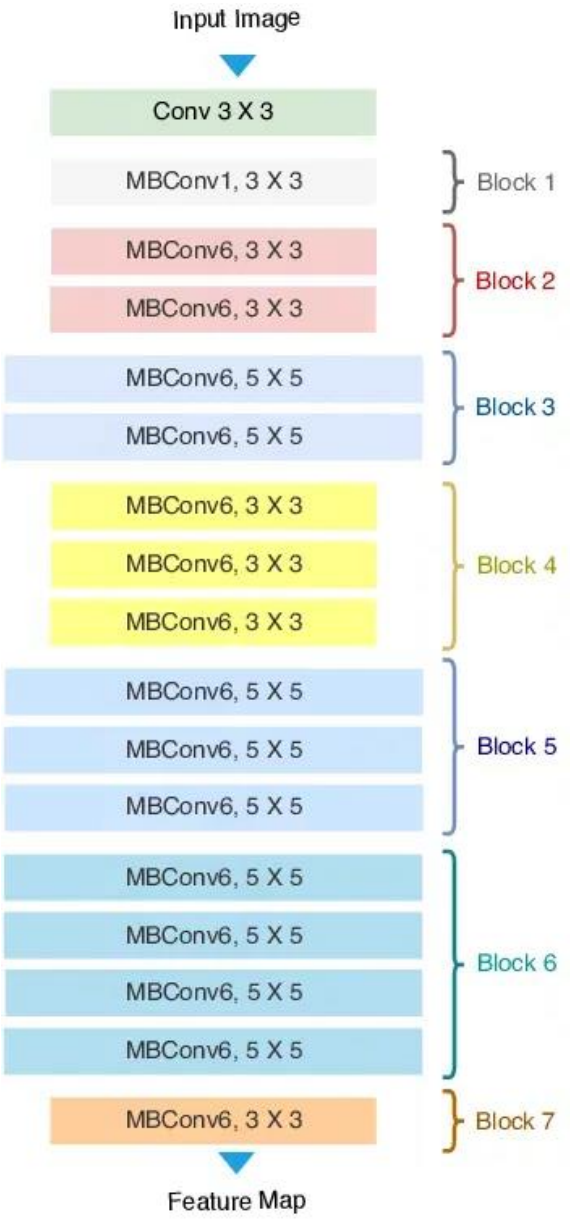

Рисунок 2.4 – Архітектура EfficientNet [27]

У медичних завданнях, де критично важливо досягати найвищої точності, але з обмеженими обчислювальними можливостями, EfficientNet показує свою особливу ефективність. Скажімо, ця архітектура дозволяє з високою точністю класифікувати медичні зображення (рентгенівські знімки, комп'ютерна томографія, ультразвук), навіть на пристроях з меншою продуктивністю, на кшталт мобільних систем або вбудованих діагностичних апаратів. До того ж, завдяки здатності до хорошого узагальнення, модель рідше перенавчається, що є суттєвим плюсом під час роботи з невеликими або нерівномірно розподіленими медичними наборами даних.

EfficientNet також чудово адаптується для трансферного навчання, що дає змогу пристосовувати вже навчену модель до специфічних медичних завдань з мінімальними витратами. Це робить її практичною для використання в реальних клінічних умовах, де потрібна оперативна та достовірна діагностика за обмежених ресурсів.

2.3 Вибір технологій для розробки рішень поставленої задачі

Python є однією з найулюбленіших мов програмування у галузі штучного інтелекту, машинного навчання та обробки медичних зображень [28]. Його поширеність пояснюється вдалим поєднанням простого синтаксису, численної спільноти розробників, широкого вибору бібліотек та гнучкості в інтеграції з іншими програмними засобами. Для вирішення завдань класифікації медичних зображень, Python надає всі необхідні інструменти: від обробки даних до розробки та навчання нейронних мереж.

Однією з ключових переваг Python є доступ до потужних бібліотек глибокого навчання, на кшталт TensorFlow, PyTorch та Keras, котрі дозволяють легко створювати, навчати та тестувати нейронні мережі різноманітної складності. Разом з цим, бібліотеки OpenCV та Pillow активно використовуються для попередньої обробки зображень, зокрема, для масштабування, нормалізації, аугментації та перетворення форматів.

Python відзначається також підтримкою у сфері біоінформатики та медичних досліджень. Бібліотеки, такі як scikit-learn, використовуються для реалізації традиційних алгоритмів машинного навчання та аналізу результатів, а NumPy, Pandas і Matplotlib - для обробки, візуалізації та статистичного аналізу даних. Цей набір інструментів робить Python ідеальним для розробки повноцінних рішень у сфері інтелектуальної обробки медичних зображень, включаючи як побудову моделей, так і інтеграцію їх у прикладні системи.

TensorFlow - це відкритий фреймворк для машинного навчання від Google. Його основне призначення - створення, тренування та розгортання моделей глибинного навчання, зокрема згорткових нейронних мереж, що особливо

важливо для аналізу зображень, таких як розпізнавання патологій на медичних рентген-знімках [29]. У рамках роботи було використано TensorFlow з високорівневим API Keras, що забезпечило просту й ефективну реалізацію моделі на базі попередньо натренованої архітектури ResNet50, а також моделі EfficientNetB3.

TensorFlow забезпечує широкі можливості для масштабування: моделі можна запускати як на одному процесорі або відеокарті, так і на кластерах серверів. У бібліотеці доступні численні попередньо навчені моделі, які легко адаптувати до нових завдань через перенавчання. У цьому дослідженні такі моделі було використано для створення класифікатора рентгенівських знімків грудної клітки, що дозволило значно скоротити час підготовки та навчання. TensorFlow також підтримує інтерактивні інструменти для моніторингу навчання, включаючи TensorBoard, який дозволяє візуалізувати зміни значень метрик, втрат, побудову ROC-кривих тощо.

Завдяки активній спільноті, постійній підтримці, широкому вибору документації та прикладів, TensorFlow став зручною платформою для вирішення цього завдання. Його використання дозволило ефективно реалізувати генерацію даних, побудову моделі, її навчання з урахуванням багатьох параметрів, а також збереження найкращого варіанта за допомогою зворотних викликів, таких як EarlyStopping і ModelCheckpoint. Отже, TensorFlow є оптимальним вибором для реалізації системи автоматичної класифікації медичних зображень.

Tkinter - стандартна бібліотека для створення графічного інтерфейсу користувача (GUI) в Python. Це обгортка над популярною бібліотекою Tcl/Tk, що поставляється разом з Python, тому не потребує додаткового встановлення.

Tkinter полегшує створення віконних додатків, які можуть містити кнопки, текстові поля, мітки, випадаючі списки, меню, вкладки та інші базові елементи інтерфейсу. Інтерфейс будується шляхом створення головного вікна, до якого додаються потрібні віджети. Tkinter також підтримує обробку подій, що дозволяє реагувати на дії користувача (натискання кнопок, введення тексту тощо).

Ця бібліотека зручна для створення невеликих настільних програм і

прототипів. Її перевагами є простота використання, швидкий старт розробки та сумісність з більшістю операційних систем. Хоча за зовнішнім виглядом Tkinter-програми можуть поступатися сучасним інтерфейсам, для навчальних або службових утиліт, особливо для запуску та тестування моделей машинного навчання, ця бібліотека залишається практичним інструментом.

Jupyter Notebook – інтерактивне середовище для програмування на Python, яке широко використовується в наукових дослідженнях, машинному навчанні та аналізі даних [30]. Його головною перевагою є можливість поєднувати в одному документі код, результати його виконання, візуалізації та текстові пояснення. Такий підхід особливо зручний в процесі експериментального дослідження моделей, коли необхідно поетапно перевіряти гіпотези, змінювати параметри та аналізувати вплив кожного кроку на результати.

Jupyter Notebook активно застосовується під час розробки рішень у сфері медичного аналізу зображень через свою гнучкість та зручність для візуального відображення результатів. Наприклад, виведення графіків точності та втрат під час навчання нейронної мережі, побудова матриці плутанини або демонстрація прикладів класифікованих медичних знімків – усе це можна організувати безпосередньо у ноутбуці. Цей формат підходить як для внутрішнього аналізу, так і для презентації результатів дослідження.

Крім того, Jupyter Notebook добре інтегрується з основними бібліотеками Python, такими як TensorFlow, PyTorch, Matplotlib, NumPy та Pandas, що дозволяє створювати повноцінні робочі прототипи моделей без потреби в окремому середовищі розробки. Ще однією перевагою є зручність у відстеженні помилок та редагуванні коду "на місці", що робить Jupyter незамінним інструментом для тестування моделей глибинного навчання на медичних зображеннях.

Висновки до розділу 2

У цьому розділі було проведено дослідження та обґрунтовано підбір технологій, потрібних для створення системи класифікації захворювань дихальних шляхів за рентгенівськими знімками. Розглянуто організацію та

ключові властивості використаного набору даних. Крім того, було виконано огляд сучасних архітектур згорткових нейронних мереж. Особливу увагу було приділено підбору інструментарію розробки: мови програмування Python та середовища Jupyter Notebook.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ КЛАСИФІКАЦІЇ ЗАХВОРЮВАНЬ ДИХАЛЬНИХ ШЛЯХІВ

3.1 Зчитування та підготовка даних

Першочергово, для роботи з даними та обраними архітектурами нейромереж потрібно завантажити та підключити бібліотеки (див. рис. 3.1).

```
import numpy as np
import pandas as pd
import os
from glob import glob
import matplotlib.pyplot as plt
from itertools import chain
import tensorflow as tf
import seaborn as sns
from tensorflow.keras import backend as K
from tensorflow.keras.applications import ResNet50, DenseNet121, EfficientNetB2
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.config import threading
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.metrics import AUC, BinaryAccuracy
from tensorflow.keras.models import load_model
```

Рисунок 3.1 – Підключення бібліотек

Далі здійснюється завантаження та початкова обробка набору даних ChestX-ray14 (див. рис. 3.2). Зчитується файл CSV із записами про знімки, а також створюється словник, який пов’язує назви зображень з їхніми повними шляхами на диску.

```
all_xray_df = pd.read_csv('./Chest_X-rays/Data_Entry_2017.csv')

all_image_paths = {
    os.path.basename(p): p
    for p in glob(os.path.join('.', 'Chest_X-rays', 'images*', '*', '*.png'))
}
print(f'Scans available: {len(all_image_paths)} | Metadata entries: {all_xray_df.shape[0]}')

all_xray_df['path'] = all_xray_df['Image Index'].map(all_image_paths.get)
all_xray_df['Patient Age'] = all_xray_df['Patient Age'].astype(int)
all_xray_df.sample(3)
```

```
Scans available: 112120 | Metadata entries: 112120
```

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height]
32075	00008387_013.png	No Finding	13	8387	42	M	AP	2500	2048
55265	00013844_005.png	Atelectasis	5	13844	36	M	PA	2992	2991
98863	00026137_000.png	No Finding	0	26137	45	M	PA	2992	2991

Рисунок 3.2 – Завантаження даних з набору

На рисунку 3.3 видно стовпчасту діаграму, що відображає частоту появи найбільш поширених класів діагнозів у наборі даних. Це дозволяє наочно побачити значний дисбаланс класів – деякі захворювання представлені тисячами знімків, тоді як інші зустрічаються значно рідше.

```
label_counts = all_xray_df['Finding Labels'].value_counts().iloc[:15]

fig, ax1 = plt.subplots(figsize=(12, 8))
ax1.bar(np.arange(len(label_counts)) + 0.5, label_counts.values)
ax1.set_xticks(np.arange(len(label_counts)) + 0.5)
xtick_labels = ax1.set_xticklabels(label_counts.index, rotation=90)
plt.tight_layout()
plt.show()
```

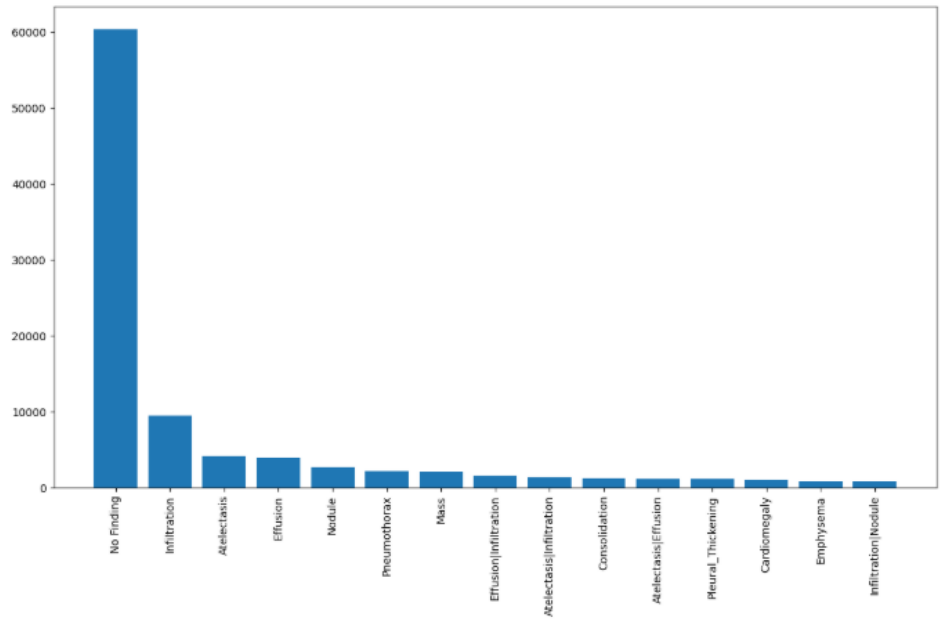

Рисунок 3.3 – Графічне відображення дисбалансу класів

Щоб зменшити негативний вплив дисбалансу під час навчання моделі, застосовується бінарна кросентропія (binary crossentropy) як функція втрат. Це стандартний підхід у задачах багатокласової багатоміткової (multi-label) класифікації, особливо коли класи можуть співіснувати в одному зразку.

Бінарна кросентропія обчислює втрату окремо для кожного класу, як для задачі двокласової класифікації. Формула:

$$Loss = -[\, y \cdot log(p) + (1 - y) \cdot log(1 - p)], \quad (3.1)$$

де y – справжнє значення (0 або 1);

p – передбачена ймовірність приналежності до класу.

Цей підхід дозволяє моделі окремо оцінювати імовірність наявності кожної хвороби та не примушує її зосереджуватися лише на більш поширених класах. Бінарна кросентропія допомагає зменшити вплив дисбалансу, надаючи кожному класу однакову значущість під час оновлення ваг моделі.

Далі виконується підготовка міток для задачі багатоміткової класифікації (див. рис. 3.4). Спочатку з записів вилучається позначка «No Finding», що означає відсутність хвороб. Потім усі діагнози, зазначені через символ «|», розбиваються й формується перелік унікальних патологій, які можуть бути наявні на знімках. Після цього для кожного діагнозу створюється окремий бінарний стовпчик у таблиці: він позначає наявність (1) або відсутність (0) відповідної патології у конкретному рентгенівському знімку.

```
all_xray_df['Finding Labels'] = all_xray_df['Finding Labels'].str.replace('No Finding', '')
all_labels = np.unique(
    list(chain.from_iterable(all_xray_df['Finding Labels'].str.split('|')))
)
all_labels = [label for label in all_labels if label.strip()]
print(f'Extracted Labels ({len(all_labels)}): {all_labels}')
for c_label in all_labels:
    if len(c_label) > 1:
        all_xray_df[c_label] = all_xray_df['Finding Labels'].map(
            lambda findings: 1.0 if c_label in findings else 0.0
        )
all_xray_df.sample(3)
```

```
Extracted Labels (14): [np.str_('Atelectasis'), np.str_('Cardiomegaly'), np.str_('Consolidation'), np.str_('Edema'), np.str_('Effusion'), np.str_('Emphysema'), np.str_('Fibrosis'), np.str_('Hernia'), np.str_('Infiltration'), np.str_('Mass'), np.str_('Nodule'), np.str_('Pleural_Thickening'), np.str_('Pneumonia'), np.str_('Pneumothorax')]
```

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position	OriginalImage[Width	Height
8857	00002339_004.png	Emphysema	4	2339	55	F	AP	2500	204
61256	00015109_000.png		0	15109	54	F	PA	2834	299
18002	00004832_021.png	Effusion|Fibrosis|Pleural_Thickening	21	4832	33	M	AP	2500	204

3 rows × 27 columns

Рисунок 3.4 – Підготовка міток для подальшої роботи

На цьому етапі виконується фільтрація міток за кількістю зразків (див. рис. 3.5). Зі списку діагнозів залишаються лише ті, що зустрічаються принаймні 1000 разів у датасеті, щоб уникнути навчання на занадто рідкісних класах, які можуть негативно вплинути на якість моделі. Таким чином було видалено мітку “Hernia”, що згадувалась лише 284 рази.

```
MIN_CASES = 1000
all_labels = [label for label in all_labels if all_xray_df[label].sum() > MIN_CASES]

print(f'Filtered Labels ({len(all_labels)}):',
      [(label, int(all_xray_df[label].sum())) for label in all_labels])
```

```
Filtered Labels (13): [(np.str_('Atelectasis'), 11559), (np.str_('Cardiomegaly'), 2776), (np.str_('Consolidation'), 4667), (np.str_('Edema'), 2303), (np.str_('Effusion'), 13317), (np.str_('Emphysema'), 2516), (np.str_('Fibrosis'), 1686), (np.str_('Infiltration'), 19894), (np.str_('Mass'), 5782), (np.str_('Nodule'), 6331), (np.str_('Pleural_Thickening'), 3385), (np.str_('Pneumonia'), 1431), (np.str_('Pneumothorax'), 5302)]
```

Рисунок 3.5 – Видалення класів з малою кількістю екземплярів

Далі формується словник, що зв’язує назви зображень з повними шляхами до відповідних файлів (див. рис. 3.6). Це дає змогу легко отримувати доступ до конкретного зображення за його індексом. До датафрейму додається новий стовпець full_path, який містить повний шлях до кожного зображення. Записи без

доступного зображення видаляються. Після цього набір даних розділяється на тренувальну та тестову вибірки у співвідношенні 80:20 для подальшого навчання та оцінки моделі.

```
all_image_paths = {
    os.path.basename(x): x
    for x in glob(os.path.join('.', 'Chest_X-rays', 'images_*', 'images', '*.png'))
}
```

```
all_xray_df['full_path'] = all_xray_df['Image Index'].map(all_image_paths.get)
all_xray_df = all_xray_df.dropna(subset=['full_path'])

train_df, test_df = train_test_split(all_xray_df, test_size=0.2, random_state=42)
```

Рисунок 3.6 – Налаштування шляхів та тренувального і тестувального наборів

Функція create_generator створює генератор зображень на основі переданого датафрейму (див. рис. 3.7). Вона приймає стовпець з шляхами до зображень (x_col) і стовпці з мітками (y_col), застосовує зазначений об’єкт аугментації datagen, і формує батчі зображень заданого розміру (target_size). Генератор повертає дані в форматі raw, тобто багатокласові або мультиміткові значення без автоматичного кодування. Зображення при цьому перемішуються на кожній епосі.

```
def create_generator(dataframe, x_col, y_col, datagen, batch_size=16, target_size=(224, 224)):
    return datagen.flow_from_dataframe(
        dataframe=dataframe,
        x_col=x_col,
        y_col=y_col,
        directory=None,
        class_mode='raw',
        batch_size=batch_size,
        target_size=target_size,
        shuffle=True
    )
```

Рисунок 3.7 – Функція генератору даних

На цьому етапі створюються генератори зображень для тренувального та тестового наборів даних (див. рис. 3.8). Спочатку визначається об’єкт ImageDataGenerator, який виконує масштабування пікселів до діапазону [0, 1] та

застосовує стандартну функцію попередньої обробки. Потім за допомогою раніше оголошеної функції create_generator формуються відповідні генератори train_generator і test_generator, які будуть використовуватись для подачі зображень у модель під час навчання та валідації.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=tf.keras.applications.resnet.preprocess_input
)
test_datagen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=tf.keras.applications.resnet.preprocess_input
)
train_generator = create_generator(train_df, 'full_path', all_labels, train_datagen, batch_size=16)
test_generator = create_generator(test_df, 'full_path', all_labels, test_datagen, batch_size=16)
```

```
Found 89696 validated image filenames.
Found 22424 validated image filenames.
```

Рисунок 3.8 – Налаштування генератору даних

Наступним етам буде налаштування та навчання моделей з різними типами архітектур.

3.2 Навчання обраних моделей

Для класифікації захворювань дихальних шляхів було створено декілька моделей на основі обраних архітектур ResNet50, DenseNet121 та EfficientNetB3. Базові моделі використовувались без верхніх (fully-connected) шарів (include_top=False), що дозволяє адаптувати їх до нового завдання багатокласової класифікації. До виходу базових моделей додається шар глобального усереднення (GlobalAveragePooling2D), дропаут-шар з імовірністю 0.5 для зменшення перенавчання, та фінальний щільний (Dense) шар з функцією активації sigmoid, що дозволяє здійснювати багатокласову класифікацію (multi-label).

Для налаштування моделей були обрані такі значення параметрів:

– розмір вхідного зображення: 224 × 224;
– функція втрат (loss): binary_crossentropy;
– оптимізатор: Adam з параметром learning_rate=1e-4;
– розмір пакета (batch size): 16;
– кількість епох навчання: 20;

- кількість кроків на епоху (steps_per_epoch): 200;
- кількість валідаційних кроків (validation_steps): 50.

Під час навчання моделі застосовувались такі метрики: бінарна точність (Binary Accuracy) та площа під ROC-кривою (AUC). Ці метрики дозволяють оцінити якість класифікації кожного класу при багатозначному результаті.

Стан навчання моделі контролювався за допомогою EarlyStopping та ModelCheckpoint. EarlyStopping автоматично припиняв навчання, якщо значення метрики AUC на валідаційній вибірці не поліпшувалось протягом 5 епох, запобігаючи перенавчанню. ModelCheckpoint зберігав найкращу модель за показником val_auc, що давало змогу зберегти оптимальний варіант моделі після завершення навчання.

Для візуалізації навчання будувались графіки зміни значень метрик AUC та функції втрат (Loss) на тренувальній і валідаційній вибірках по епохах. Також було побудовано ROC-криві для кожного класу, що дало можливість детально оцінити якість класифікації по кожному з діагностичних станів.

3.2.1 Навчання ResNet50

Спочатку завантажується базова модель ResNet50 з вагами, заздалегідь навченими на наборі даних ImageNet. Параметр include_top=False визначає, що верхній (останній) повнозв'язний рівень моделі не буде доданий, що дозволяє адаптувати модель під певну задачу. Розмір вхідних зображень задається як (224, 224, 3), що відповідає звичним налаштуванням ResNet50.

Далі, до виходу базової моделі додається глобальний рівень усереднення (GlobalAveragePooling2D), котрий зменшує просторовий вимір тензору, обчислюючи середнє значення для кожного каналу. Це дає змогу перетворити вихід базової моделі у вектор фіч, який можна застосувати для класифікації. Після цього застосовується шар дропауту (Dropout з параметром 0.5), що допомагає запобігти перенавчанню шляхом випадкового "відключення" частини нейронів під час навчання. На закінчення, додається повнозв'язний шар (Dense) з кількістю нейронів, що дорівнює кількості класів у задачі (len(all_labels)), та сигмоїдною

функцією активації, котра зазвичай застосовується для багатокласової класифікації.

Модель формується за допомогою класу Model, де вхідними даними є вхідний шар базової моделі, а вихідними – прогнози останнього шару (див. рис. 3.9). Після компіляції моделі (яка не показана у коді, але зазвичай включає вибір оптимізатора, функції втрат та метрик) починається процес навчання. Під час навчання модель проходить через епохи, де на кожній ітерації оновлюються ваги з урахуванням градієнтів функції втрат.

```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D ()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(all_labels), activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

Рисунок 3.9 – Створення моделі ResNet50

Загальна кількість параметрів моделі становить 23 614 349, з яких 23 561 229 є навчальними, а 53 120 – замороженими (не навчаються, оскільки базовий ResNet50 спочатку був заморожений для тонкого навчання). Обсяг пам’яті, необхідний для зберігання моделі, становить приблизно 90 МБ, що є відносно невеликим для такої потужної архітектури (див. рис. 3.10). Ця інформація дозволяє зрозуміти масштаб моделі та її обчислювальні потреби.

```
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=[
        BinaryAccuracy(name='binary_accuracy'),
        AUC(name='auc', multi_label=True)
    ]
)
model.summary()
```

(GlobalAveragePooling2D)			
dropout (Dropout)	(None, 2048)	0	global_average_pooling2d[…
dense (Dense)	(None, 13)	26,637	dropout[0][0]

Total params: 23,614,349 (90.08 MB)

Trainable params: 23,561,229 (89.88 MB)

Non-trainable params: 53,120 (207.50 KB)

Рисунок 3.10 – Налаштування параметрів компіляції моделі ResNet50

```
callbacks = [
    EarlyStopping(monitor='val_auc', patience=5, mode='max', verbose=1),
    ModelCheckpoint('ResNet50.keras', monitor='val_auc',
                    save_best_only=True, mode='max', verbose=1)
]


history = model.fit(
    train_generator,
    steps_per_epoch=200,
    validation_data=test_generator,
    validation_steps=50,
    epochs=20,
    verbose=1,
    callbacks=callbacks
)
```

```
Epoch 1/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.5669 - binary_accuracy: 0.8861 - loss: 0.2980
Epoch 1: val_auc improved from -inf to 0.54832, saving model to ResNet50.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 714s 3s/step - auc: 0.5672 - binary_accuracy: 0.8863 - loss: 0.2976 - val_auc: 0.5483 - val_binary_accuracy: 0.8877 - val
_loss: 0.3016
Epoch 2/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.6528 - binary_accuracy: 0.9407 - loss: 0.1992
Epoch 2: val_auc improved from 0.54832 to 0.55942, saving model to ResNet50.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 695s 3s/step - auc: 0.6528 - binary_accuracy: 0.9407 - loss: 0.1992 - val_auc: 0.5594 - val_binary_accuracy: 0.8401 - val
_loss: 0.3292
Epoch 3/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.6681 - binary_accuracy: 0.9426 - loss: 0.1883
Epoch 3: val_auc improved from 0.55942 to 0.59649, saving model to ResNet50.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 696s 3s/step - auc: 0.6682 - binary_accuracy: 0.9426 - loss: 0.1883 - val_auc: 0.5965 - val_binary_accuracy: 0.9435 - val
_loss: 0.2047
Epoch 4/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7121 - binary_accuracy: 0.9484 - loss: 0.1701
Epoch 4: val_auc improved from 0.59649 to 0.70699, saving model to ResNet50.keras
```

Рисунок 3.11 – Запуск навчання моделі ResNet50

Після завершення навчання виводиться інформація про точність на тренувальному та валідаційному наборах даних, а також графіки зміни точності та втрат протягом епох, що дозволяє оцінити якість навчання та виявити можливе перенавчання (див. рис. 3.12).

```
Epoch 15/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7507 - binary_accuracy: 0.9436 - loss: 0.1748
Epoch 15: val_auc did not improve from 0.77910
200/200 ━━━━━━━━━━━━━━━━━━━━ 688s 3s/step - auc: 0.7507 - binary_accuracy: 0.9436 - loss: 0.1748 - val_auc: 0.7691 - val_binary_accuracy: 0.9447 - val_l
oss: 0.1838
Epoch 16/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7641 - binary_accuracy: 0.9482 - loss: 0.1647
Epoch 16: val_auc improved from 0.77910 to 0.78668, saving model to ResNet50_v2.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 700s 4s/step - auc: 0.7641 - binary_accuracy: 0.9482 - loss: 0.1647 - val_auc: 0.7867 - val_binary_accuracy: 0.9434 - val_l
oss: 0.1758
Epoch 17/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7515 - binary_accuracy: 0.9429 - loss: 0.1762
Epoch 17: val_auc did not improve from 0.78668
200/200 ━━━━━━━━━━━━━━━━━━━━ 678s 3s/step - auc: 0.7515 - binary_accuracy: 0.9429 - loss: 0.1762 - val_auc: 0.7589 - val_binary_accuracy: 0.9485 - val_l
oss: 0.1610
Epoch 18/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7676 - binary_accuracy: 0.9453 - loss: 0.1698
Epoch 18: val_auc did not improve from 0.78668
200/200 ━━━━━━━━━━━━━━━━━━━━ 641s 3s/step - auc: 0.7676 - binary_accuracy: 0.9453 - loss: 0.1698 - val_auc: 0.7627 - val_binary_accuracy: 0.9451 - val_l
oss: 0.1702
Epoch 19/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7512 - binary_accuracy: 0.9444 - loss: 0.1737
Epoch 19: val_auc did not improve from 0.78668
200/200 ━━━━━━━━━━━━━━━━━━━━ 644s 3s/step - auc: 0.7513 - binary_accuracy: 0.9444 - loss: 0.1737 - val_auc: 0.7273 - val_binary_accuracy: 0.9433 - val_l
oss: 0.1820
Epoch 20/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7565 - binary_accuracy: 0.9414 - loss: 0.1785
Epoch 20: val_auc did not improve from 0.78668
200/200 ━━━━━━━━━━━━━━━━━━━━ 647s 3s/step - auc: 0.7565 - binary_accuracy: 0.9414 - loss: 0.1785 - val_auc: 0.7318 - val_binary_accuracy: 0.9459 - val_l
oss: 0.1745
```

Рисунок 3.12 – Процес навчання моделі ResNet50

На графіку з рисунку 3.13 видно, що AUC для тренувального та валідаційного наборів поступово зростає протягом епох, стабілізуючись близько до значення 0.78. Це свідчить про успішне навчання без перенавчання: валідаційна крива не демонструє значного падіння, а тримається на рівні або вище тренувальної.

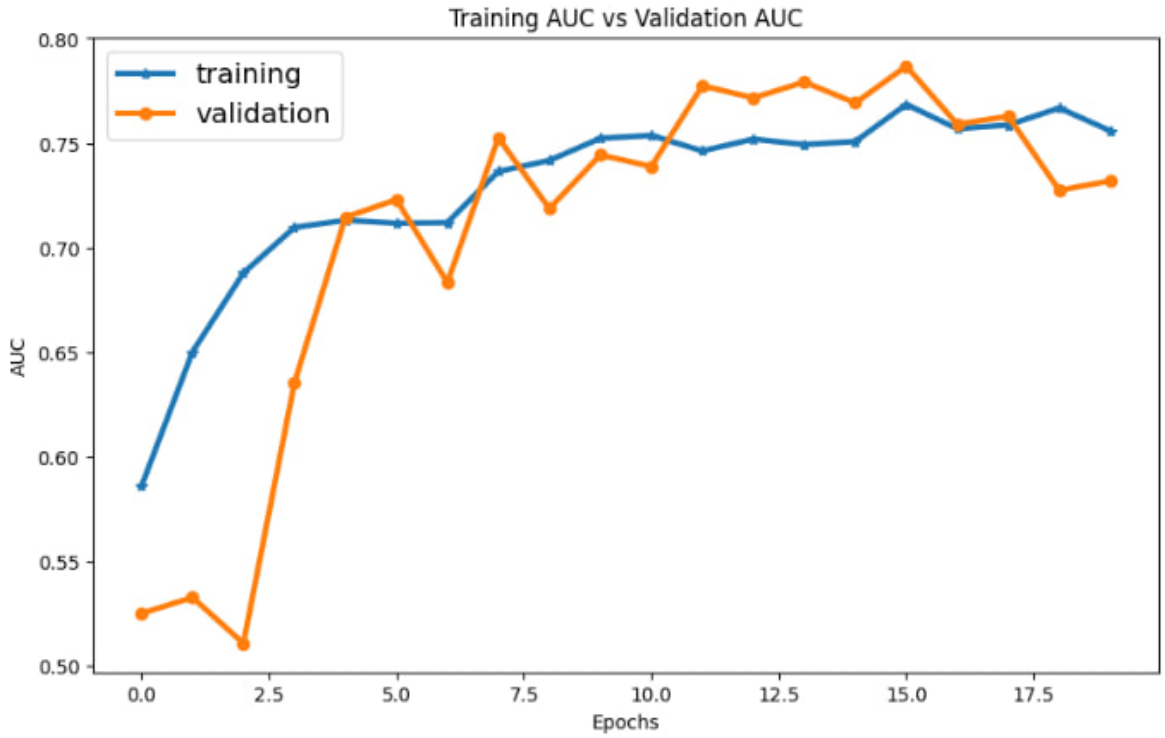

Рисунок 3.13 – Графік залежності AUC від епохи навчання моделі ResNet50

Функція втрат на рисунку 3.14, у цьому випадку бінарна кросс-ентропія, показує, наскільки прогнози моделі відрізняються від реальних значень. Тренувальні втрати поступово зменшуються до 0.18, що вказує на те, що модель ефективно оптимізується. Валідаційні втрати також знижуються, але їхні значення залишаються трохи вищими, що є нормальним явищем, оскільки модель не бачила цих даних під час навчання. Важливо відзначити, що обидві криві мають плавний спад без різких коливань, що свідчить про стабільність процесу навчання.

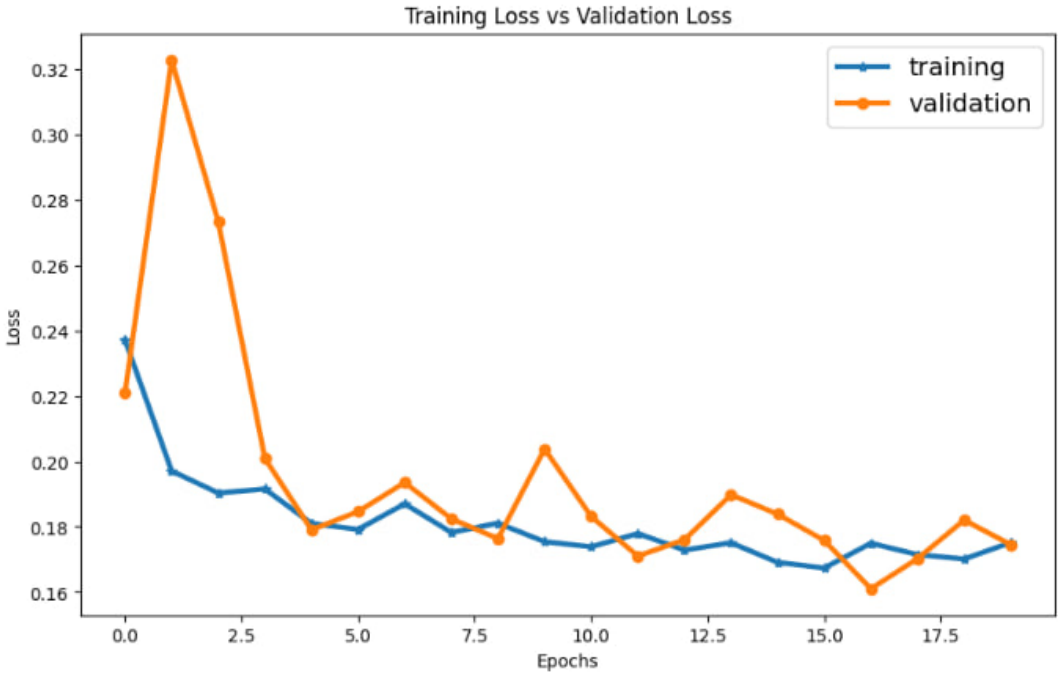

Рисунок 3.14 – Графік залежності втрат від епохи навчання моделі ResNet50

ROC-криві для кожного класу продемонстровані на рисунку 3.15 показують досить високу якість моделі. AUC (площа під кривою) варіюється від 0.67 до 0.88, що свідчить про добру здатність моделі розрізняти наявність і відсутність кожної з патологій. Найвищі значення AUC спостерігаються для таких захворювань, як Cardiomegaly, Edema, Effusion, Pneumothorax та Emphysema, що свідчить про впевненість моделі у класифікації цих станів.

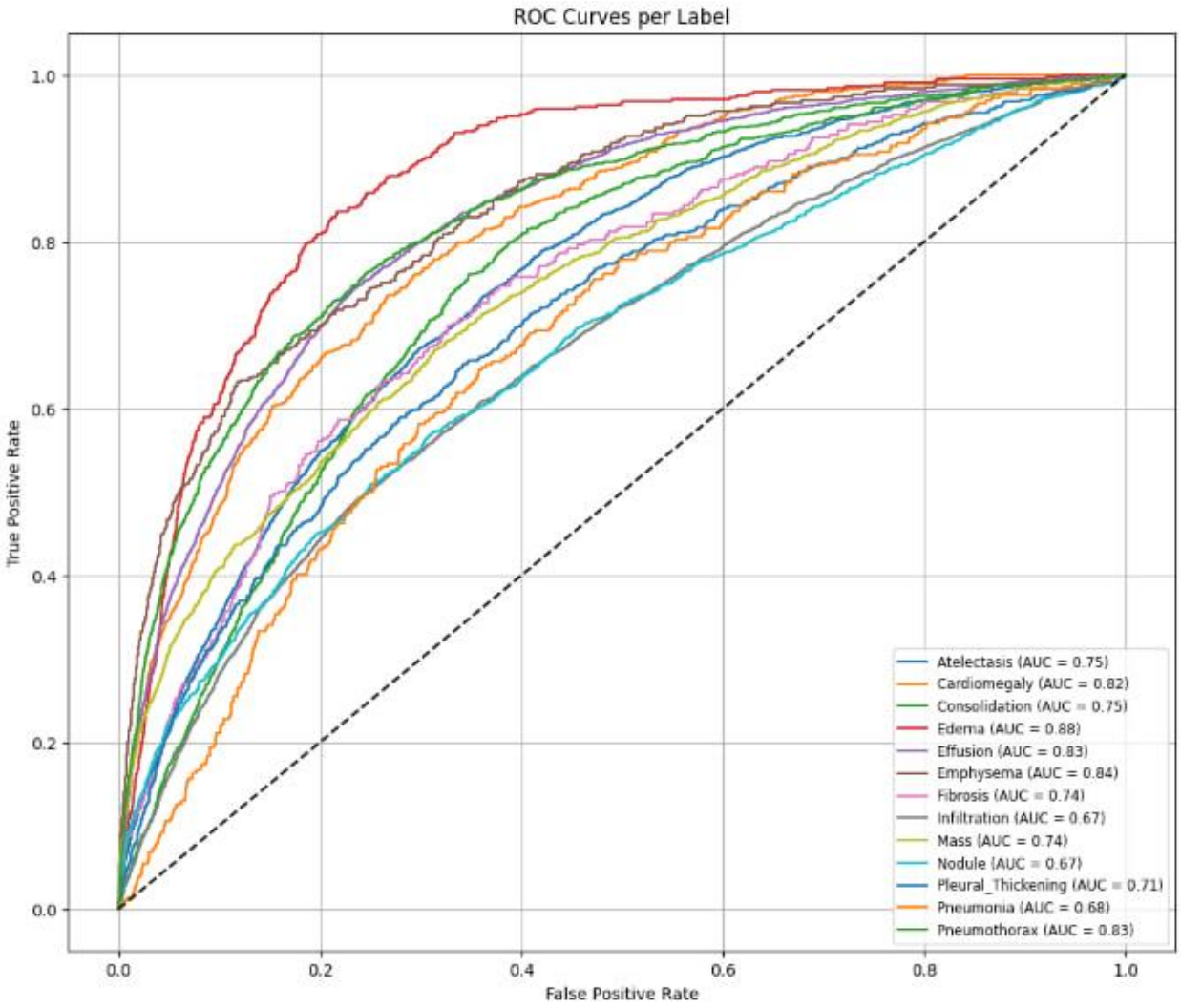

Рисунок 3.15 – Порівняння ROC-кривих для різних класів моделі ResNet50

Модель демонструє хорошу узагальнювальну здатність на нових даних. Більшість патологій класифікуються з високою точністю, а процес навчання був стабільним і ефективним. Мережа потенційно придатна для застосування у клінічному середовищі як інструмент попереднього сортування зображень.

3.2.2 Навчання DenseNet121

Створення та налаштування для моделі DenseNet121 відбувається за прикладом ResNet50.

```
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D ()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(all_labels), activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

Рисунок 3.16 – Створення моделі DenseNet121

```
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=[
        BinaryAccuracy(name='binary_accuracy'),
        AUC(name='auc', multi_label=True)
    ]
)
model.summary()
```

(GlobalAveragePooling2D)			
dropout (Dropout)	(None, 1024)	0	global_average_pooling2d[…
dense (Dense)	(None, 13)	13,325	dropout[0][0]

Total params: 7,050,829 (26.90 MB)

Trainable params: 6,967,181 (26.58 MB)

Non-trainable params: 83,648 (326.75 KB)

Рисунок 3.17 – Налаштування параметрів компіляції моделі DenseNet121

```
callbacks = [
    EarlyStopping(monitor='val_auc', patience=5, mode='max', verbose=1),
    ModelCheckpoint('densenet121.keras', monitor='val_auc',
                    save_best_only=True, mode='max', verbose=1)
]


history = model.fit(
    train_generator,
    steps_per_epoch=200,
    validation_data=test_generator,
    validation_steps=50,
    epochs=20,
    verbose=1,
    callbacks=callbacks
)
```

Рисунок 3.18 – Запуск навчання моделі DenseNet121

Під час навчання виводяться метрики: AUC, бінарна точність (binary_accuracy) та втрати (loss) для тренувальної та валідаційної вибірок. Наприклад, на 15-й епосі AUC на тренуванні становить 0.7567, а на валідації –

0.7813. На 17-й епосі val_auc покращився до 0.8059, і модель була збережена. Однак на наступних епохах цей показник не перевищувався, тому після 20-ї епохи навчання завершилося.

```
Epoch 15/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7567 - binary_accuracy: 0.9447 - loss: 0.1721
Epoch 15: val_auc did not improve from 0.78504
200/200 ━━━━━━━━━━━━━━━━━━━━ 693s 3s/step - auc: 0.7567 - binary_accuracy: 0.9447 - loss: 0.1721 - val_auc: 0.7813 - val_binary_accuracy: 0.9473 - val_l
oss: 0.1662
Epoch 16/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7464 - binary_accuracy: 0.9435 - loss: 0.1745
Epoch 16: val_auc did not improve from 0.78504
200/200 ━━━━━━━━━━━━━━━━━━━━ 703s 4s/step - auc: 0.7465 - binary_accuracy: 0.9435 - loss: 0.1745 - val_auc: 0.7676 - val_binary_accuracy: 0.9473 - val_l
oss: 0.1660
Epoch 17/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7570 - binary_accuracy: 0.9449 - loss: 0.1729
Epoch 17: val_auc improved from 0.78504 to 0.80586, saving model to densenet121.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 707s 4s/step - auc: 0.7570 - binary_accuracy: 0.9449 - loss: 0.1729 - val_auc: 0.8059 - val_binary_accuracy: 0.9491 - val_l
oss: 0.1557
Epoch 18/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7611 - binary_accuracy: 0.9462 - loss: 0.1686
Epoch 18: val_auc did not improve from 0.80586
200/200 ━━━━━━━━━━━━━━━━━━━━ 701s 4s/step - auc: 0.7611 - binary_accuracy: 0.9462 - loss: 0.1686 - val_auc: 0.7736 - val_binary_accuracy: 0.9462 - val_l
oss: 0.1709
Epoch 19/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7720 - binary_accuracy: 0.9445 - loss: 0.1707
Epoch 19: val_auc did not improve from 0.80586
200/200 ━━━━━━━━━━━━━━━━━━━━ 671s 3s/step - auc: 0.7721 - binary_accuracy: 0.9445 - loss: 0.1707 - val_auc: 0.7485 - val_binary_accuracy: 0.9466 - val_l
oss: 0.1717
Epoch 20/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - auc: 0.7769 - binary_accuracy: 0.9425 - loss: 0.1748
Epoch 20: val_auc did not improve from 0.80586
200/200 ━━━━━━━━━━━━━━━━━━━━ 662s 3s/step - auc: 0.7768 - binary_accuracy: 0.9425 - loss: 0.1748 - val_auc: 0.7751 - val_binary_accuracy: 0.9484 - val_l
oss: 0.1623
```

Рисунок 3.19 – Процес навчання моделі DenseNet121

Графік Training AUC vs Validation AUC на рисунку 3.20 демонструє поступове зростання як метрики AUC на тренувальних, так і на валідаційних даних, що свідчить про успішне навчання моделі. Вже з перших епох видно значне покращення продуктивності, а далі криві залишаються близькими одна до одної без помітного розходження, що є позитивною ознакою – модель не переобучилась і добре узагальнює.

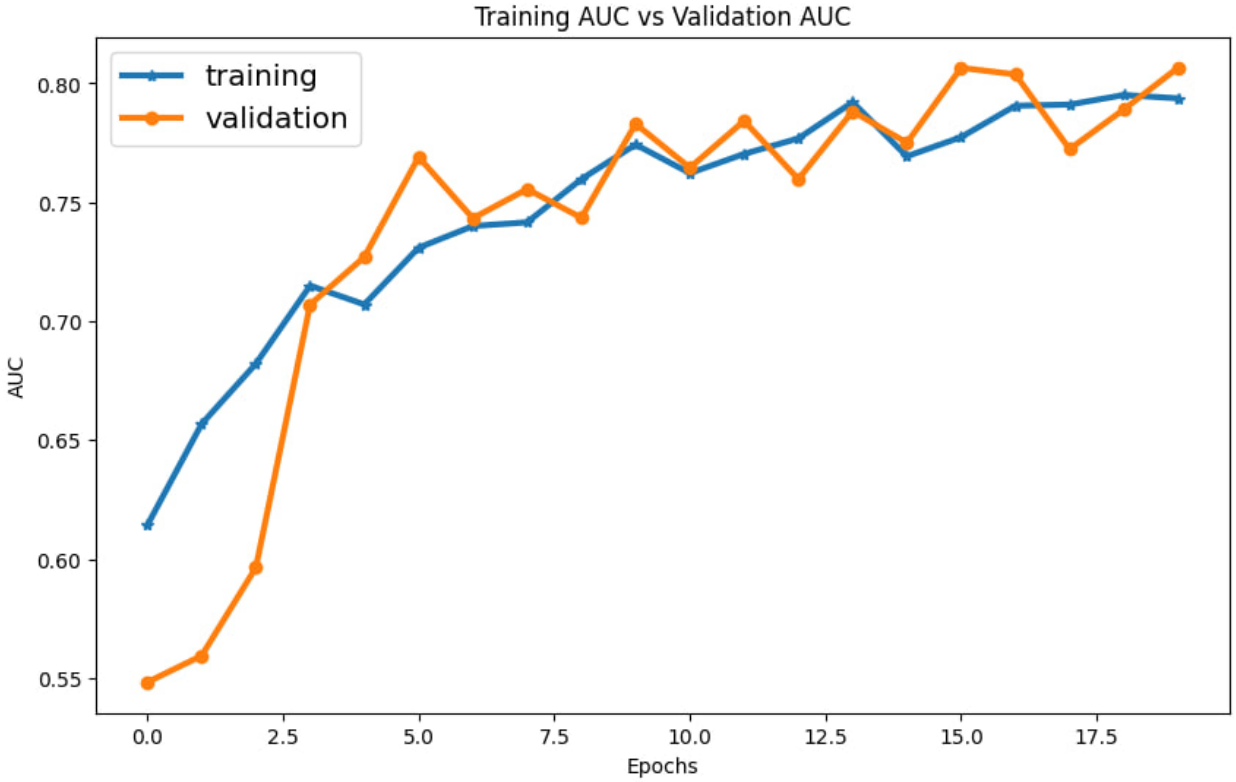

Рисунок 3.20 – Графік залежності AUC від епохи навчання моделі DenseNet121

Криві втрат на рисунку 3.21 демонструють класичну картину успішного навчання нейромережі. Початкові втрати на рівні 0.325 поступово зменшуються з кожною епохою, досягаючи 0.175 на тренувальній вибірці та 0.18 на валідаційній. Плавний спад обох кривих без різких коливань свідчить про стабільний процес оптимізації. Невеликий розрив між тренувальними та валідаційними втратами підтверджує відсутність перенавчання, що є хорошим знаком для подальшого використання моделі.

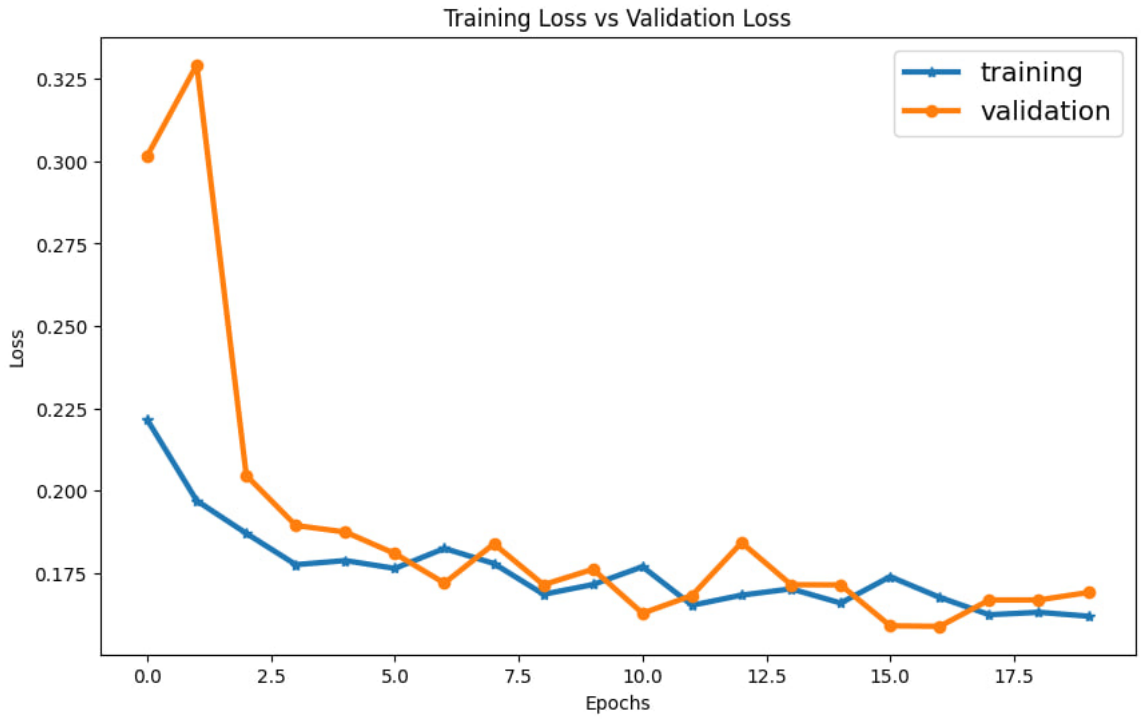

Рисунок 3.21 – Графік залежності втрат від епохи навчання моделі DenseNet121

Графік ROC Curves per Label на рисунку 3.22 відображає якість класифікації для кожного класу (патології) окремо. Більшість класів мають AUC значення вище 0.75, з деякими особливо високими результатами – наприклад, Cardiomegaly, Edema, Effusion та Pneumothorax досягають AUC ≈ 0.85–0.86. Це свідчить про високу здатність моделі правильно класифікувати позитивні й негативні випадки по цих мітках. Менш точні класи – Infiltration і Pneumonia (AUC ≈ 0.70) – ймовірно, страждають від меншої кількості даних або нечітких візуальних ознак.

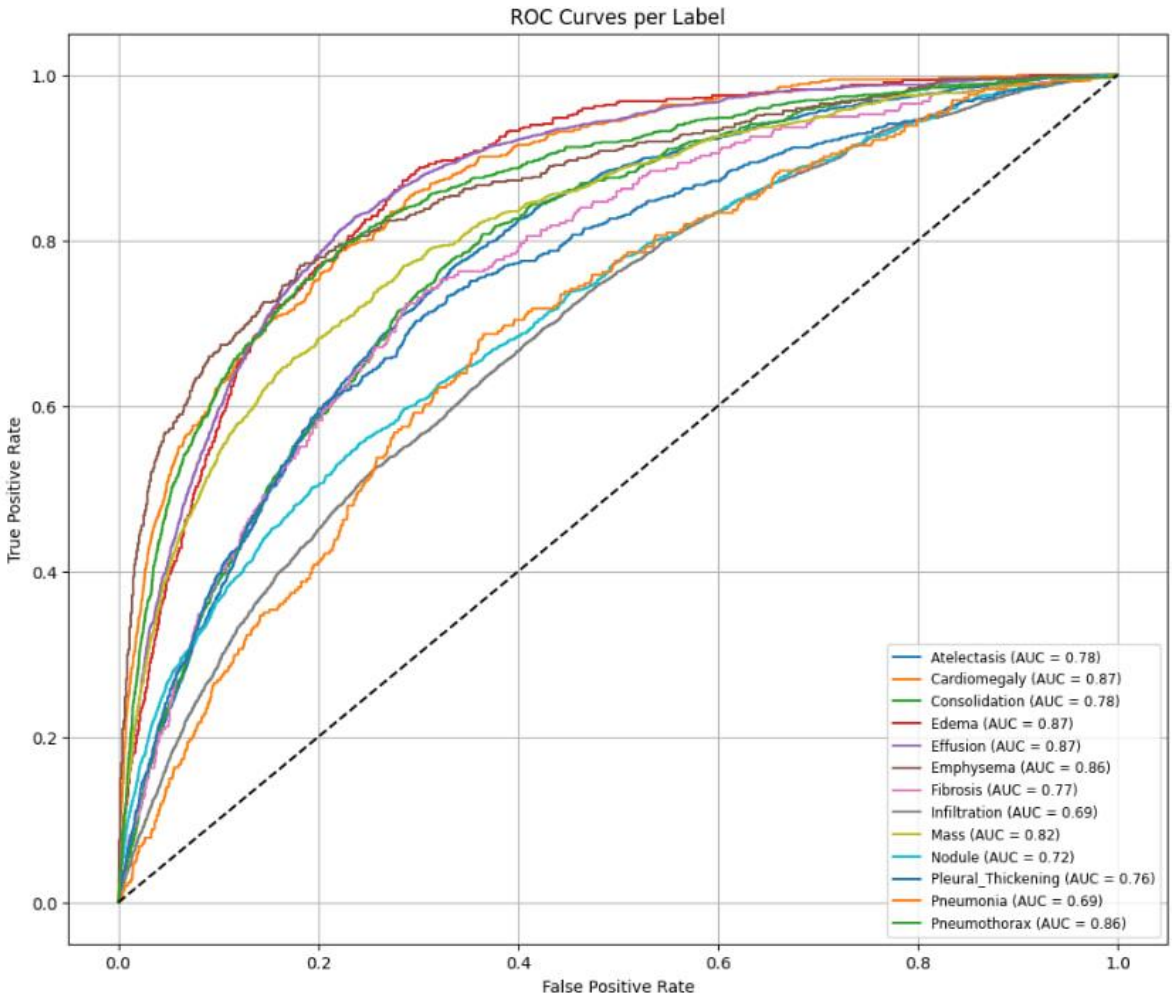

Рисунок 3.22 – Порівняння ROC-кривих для різних класів моделі DenseNet121

Загалом, модель показує добрі результати для багатокласової багатозначної класифікації медичних зображень, з хорошим балансом між тренувальними і валідаційними показниками.

3.2.3 Навчання EfficientNetB4

Модель EfficientNetB4 так само створюється за прикладом до моделей ResNet50 та DenseNet121.

```
base_model = EfficientNetB4(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D ()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(all_labels), activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

Рисунок 3.23 – Створення моделі EfficientNetB4

```
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=[
        BinaryAccuracy(name='binary_accuracy'),
        AUC(name='auc', multi_label=True)
    ]
)
model.summary()
```

(GlobalAveragePooling2D)			
dropout (Dropout)	(None, 1792)	0	global_average_pooling2d[…
dense (Dense)	(None, 13)	23,309	dropout[0][0]

Total params: 17,697,132 (67.51 MB)

Trainable params: 17,571,925 (67.03 MB)

Non-trainable params: 125,207 (489.09 KB)

Рисунок 3.24 – Налаштування параметрів компіляції моделі EfficientNetB4

```
callbacks = [
    EarlyStopping(monitor='val_auc', patience=5, mode='max', verbose=1),
    ModelCheckpoint('EfficientNetB4.keras', monitor='val_auc',
                    save_best_only=True, mode='max', verbose=1)
]


history = model.fit(
    train_generator,
    steps_per_epoch=200,
    validation_data=test_generator,
    validation_steps=50,
    epochs=20,
    verbose=1,
    callbacks=callbacks
)
```

Рисунок 3.25 – Запуск навчання моделі EfficientNetB4

Після навчання метрики моделі на тренувальних даних були значно вищими, ніж на валідації, що може свідчити про перенавчання або проблеми з валідаційними даними. Втрати на валідації мали великі коливання (від 0.21 до 181.9), що також вказує на нестабільність навчання.

```
Epoch 1/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.4891 - binary_accuracy: 0.8341 - loss: 0.4155
Epoch 1: val_auc improved from -inf to 0.50320, saving model to EfficientNetB4.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 952s 4s/step - auc: 0.4894 - binary_accuracy: 0.8345 - loss: 0.4148 - val_auc: 0.5032 - val_binary_accuracy: 0.9494 - val_lo
ss: 0.2007
Epoch 2/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.6400 - binary_accuracy: 0.9490 - loss: 0.1782
Epoch 2: val_auc improved from 0.50320 to 0.60067, saving model to EfficientNetB4.keras
200/200 ━━━━━━━━━━━━━━━━━━━━ 838s 4s/step - auc: 0.6401 - binary_accuracy: 0.9490 - loss: 0.1782 - val_auc: 0.6007 - val_binary_accuracy: 0.9470 - val_lo
ss: 0.1974
Epoch 3/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.6461 - binary_accuracy: 0.9465 - loss: 0.1823
Epoch 3: val_auc did not improve from 0.60067
200/200 ━━━━━━━━━━━━━━━━━━━━ 874s 4s/step - auc: 0.6461 - binary_accuracy: 0.9465 - loss: 0.1823 - val_auc: 0.5185 - val_binary_accuracy: 0.9456 - val_lo
ss: 0.1959
Epoch 4/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.6831 - binary_accuracy: 0.9443 - loss: 0.1855
Epoch 4: val_auc did not improve from 0.60067
200/200 ━━━━━━━━━━━━━━━━━━━━ 870s 4s/step - auc: 0.6831 - binary_accuracy: 0.9443 - loss: 0.1855 - val_auc: 0.5436 - val_binary_accuracy: 0.9462 - val_lo
ss: 0.2097
Epoch 5/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.6823 - binary_accuracy: 0.9449 - loss: 0.1807
Epoch 5: val_auc did not improve from 0.60067
200/200 ━━━━━━━━━━━━━━━━━━━━ 867s 4s/step - auc: 0.6824 - binary_accuracy: 0.9449 - loss: 0.1807 - val_auc: 0.5656 - val_binary_accuracy: 0.9510 - val_lo
ss: 0.1873
Epoch 6/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.7052 - binary_accuracy: 0.9430 - loss: 0.1858
Epoch 6: val_auc did not improve from 0.60067
200/200 ━━━━━━━━━━━━━━━━━━━━ 879s 4s/step - auc: 0.7053 - binary_accuracy: 0.9430 - loss: 0.1858 - val_auc: 0.5849 - val_binary_accuracy: 0.9462 - val_lo
ss: 0.1992
Epoch 7/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - auc: 0.7361 - binary_accuracy: 0.9444 - loss: 0.1788
Epoch 7: val_auc did not improve from 0.60067
200/200 ━━━━━━━━━━━━━━━━━━━━ 890s 4s/step - auc: 0.7361 - binary_accuracy: 0.9444 - loss: 0.1788 - val_auc: 0.5957 - val_binary_accuracy: 0.9444 - val_lo
ss: 0.1933
Epoch 7: early stopping
```

Рисунок 3.26 – Процес навчання моделі EfficientNetB4

Графік AUC на рисунку 3.26 показує проблеми з навчанням моделі - значення AUC на валідації залишаються на рівні 0.5-0.55 протягом усіх епох, що відповідає випадковому вгадуванню, тоді як тренувальний AUC зростає до 0.75. Така велика різниця між тренувальними та валідаційними показниками свідчить про серйозне перенавчання моделі.

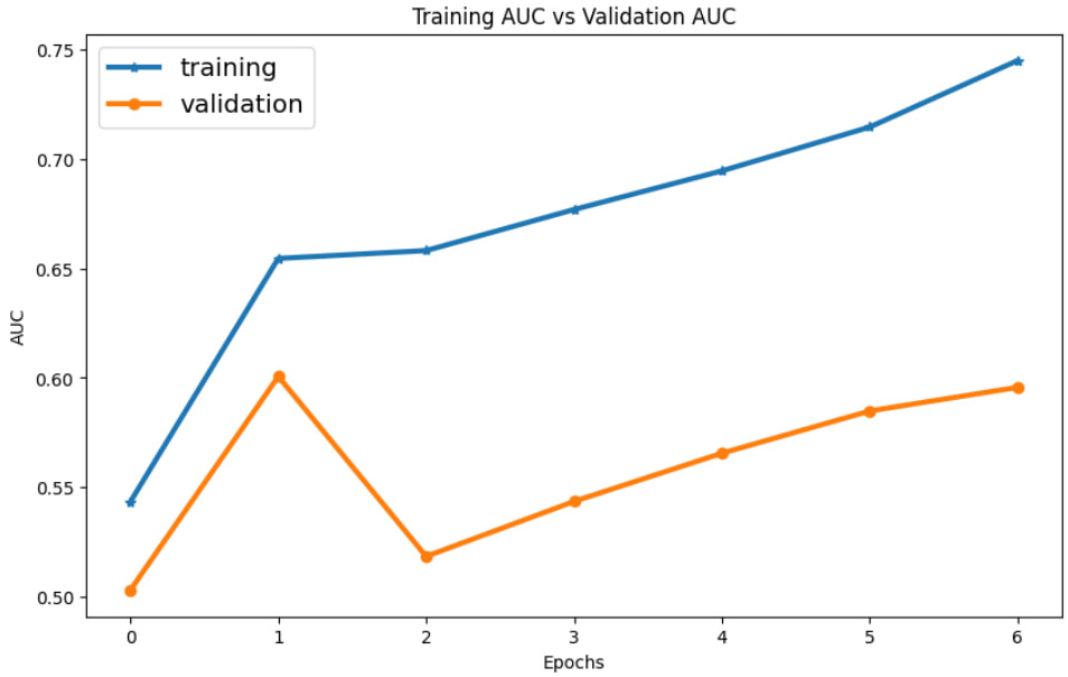

Рисунок 3.27 – Графік залежності AUC від епохи навчання моделі EfficientNetB4

Крива втрат на рисунку 3.28 підтверджує цю проблему - втрати на тренуванні швидко знижуються, але валідаційні втрати катастрофічно зростають до 200, що є явною ознакою того, що модель запам'ятовує тренувальні дані замість навчання корисним закономірностям.

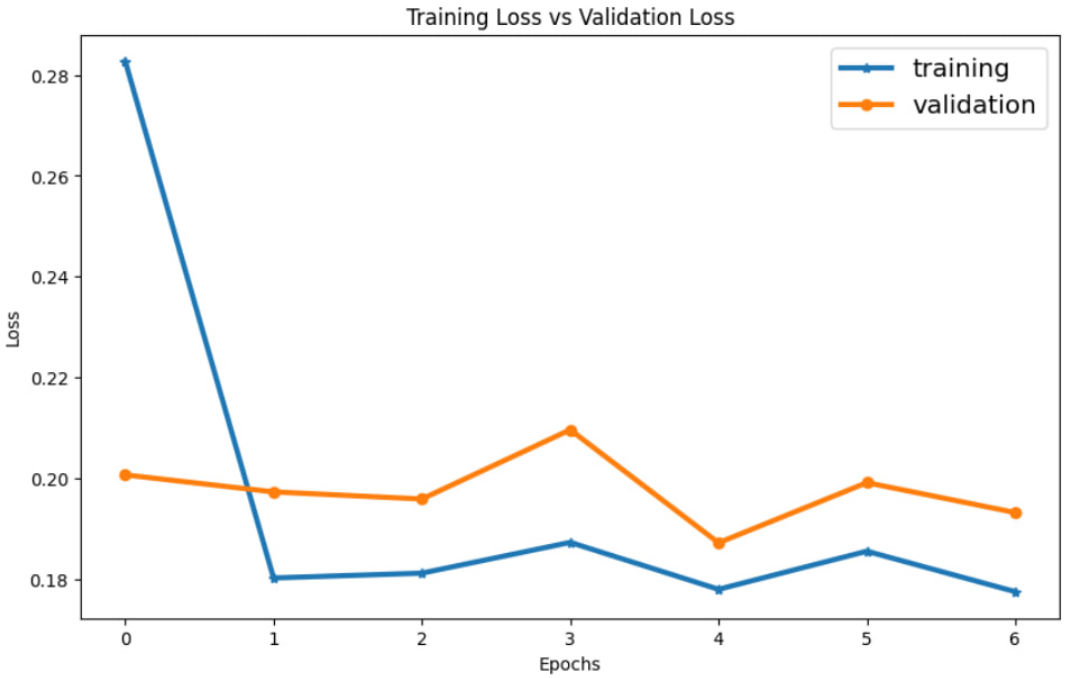

Рисунок 3.28 – Графік залежності втрат від епохи навчання моделі EfficientNetB4

ROC-криві на рисунку 3.29 для окремих класів демонструють повну неспроможність моделі класифікувати будь-яку з патологій - значення AUC для всіх 13 класів коливаються між 0.49 і 0.51, що практично не відрізняється від випадкового вгадування. Особливо тривожно, що навіть для таких типових патологій як кардіомегалія або пневмоторакс модель показує AUC=0.5, що робить її абсолютно непридатною для клінічного використання.

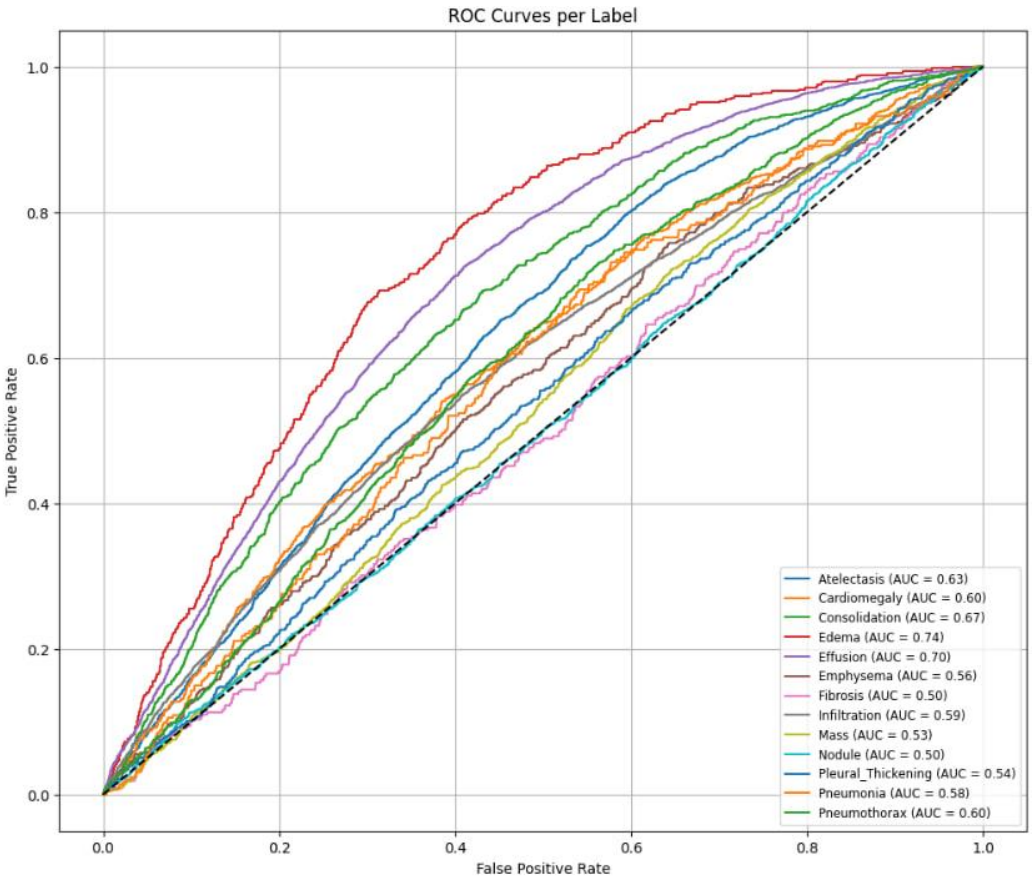

Рисунок 3.29 – Порівняння ROC-кривих для різних класів моделі EfficientNetB4

У випадку моделі EfficientNetB4 спостерігається менш успішне навчання у порівнянні з ResNet50. Максимальне значення AUC, яке вдалося досягти на валідаційній вибірці, становило 0.71 і було зафіксоване на 7-й епосі. Після цього покращення метрики не спостерігалося, через що було активовано механізм ранньої зупинки навчання – модель припинила тренування автоматично після 5 епох без покращення.

Це свідчить про те, що модель або не змогла ефективно адаптуватися до специфіки медичних рентген-знімків, або виявилася менш стабільною до параметрів навчання чи самої підготовки даних. Можливо, вона потребує більш тонкої настройки гіперпараметрів (наприклад, меншого learning rate або більш агресивної регуляризації), додаткового збалансування класів чи навіть донавчання на більших вибірках, перш ніж зможе перевершити результати ResNet50.

Хоча EfficientNetB3 є потужною архітектурою, у цьому конкретному випадку вона не змогла досягти достатнього рівня узагальнення, і її

продуктивність виявилася нижчою за базову модель.

3.3 Розробка програмного застосунку

Для перевірки моделей на тестових даних було написано застосунок на мові Python з використанням вбудованої бібліотеки Tkinter. Дана бібліотека дозволяє створювати віконний застосунок, що буде корисно для даної задачі, де необхідно створити простий для розуміння і легкий інтерфейс для взаємодії користувача з моделями нейромереж. Код застосунку наведено у додатку Б.

Інтерфейс застосунку на рисунку 3.30 дозволяє користувачу обрати одну з трьох моделей (ResNet50, DenseNet121 або EfficientNetB4), завантажити зображення рентгенівського знімку, та, при натиску на кнопку “Predict” побачити передбачення конкретної моделі у вигляді графіка та текстовому вигляді.

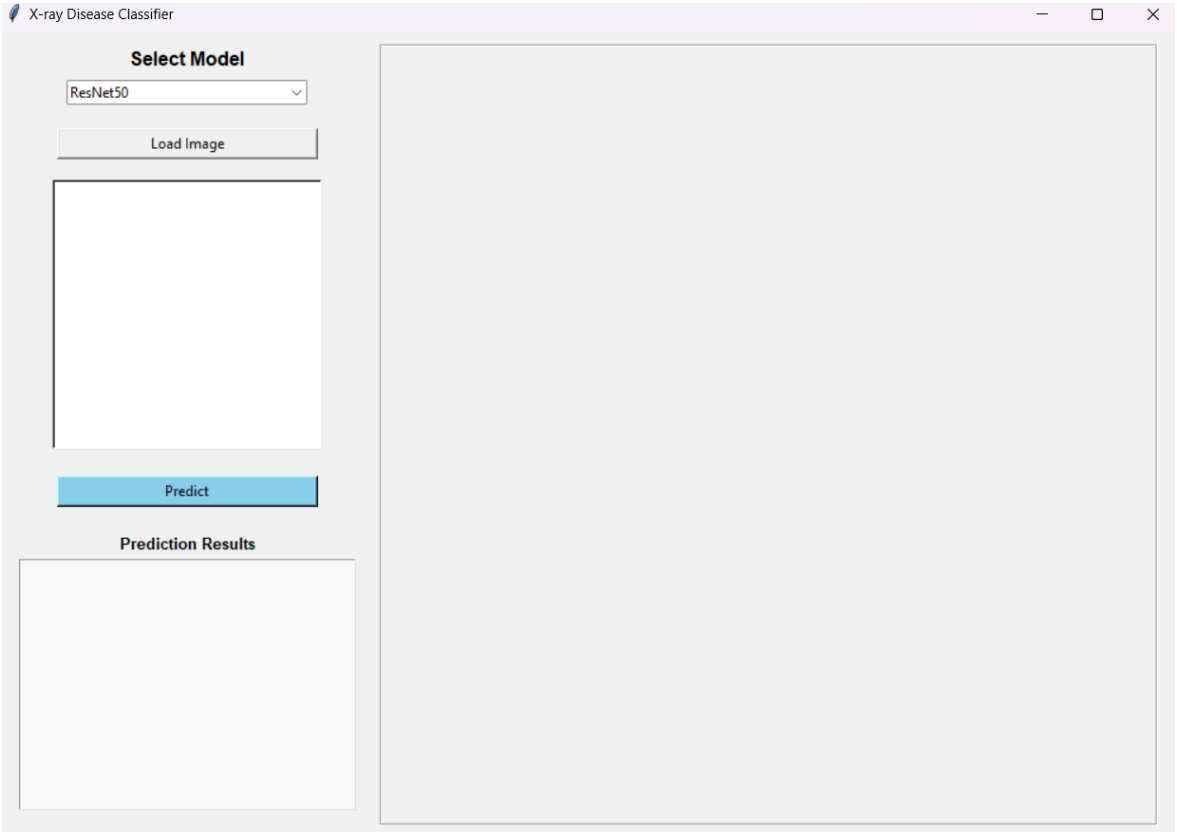

Рисунок 3.30 – Інтерфейс застосунку

У цьому прикладі (рис 3.31) модель, скоріше за все, передбачає, що зображення рентгенівського знімку подане на вхід моделі не має ознак

захворювань. Це видно як з графіку, де найвищі значення досягають позначки 0.09, а деякі і зовсім залишаються на позначці 0 (тобто модель впевнена у відсутності ознак цього захворювання) так і з текстового вікна, де показники усіх міток мають дуже низькі значення.

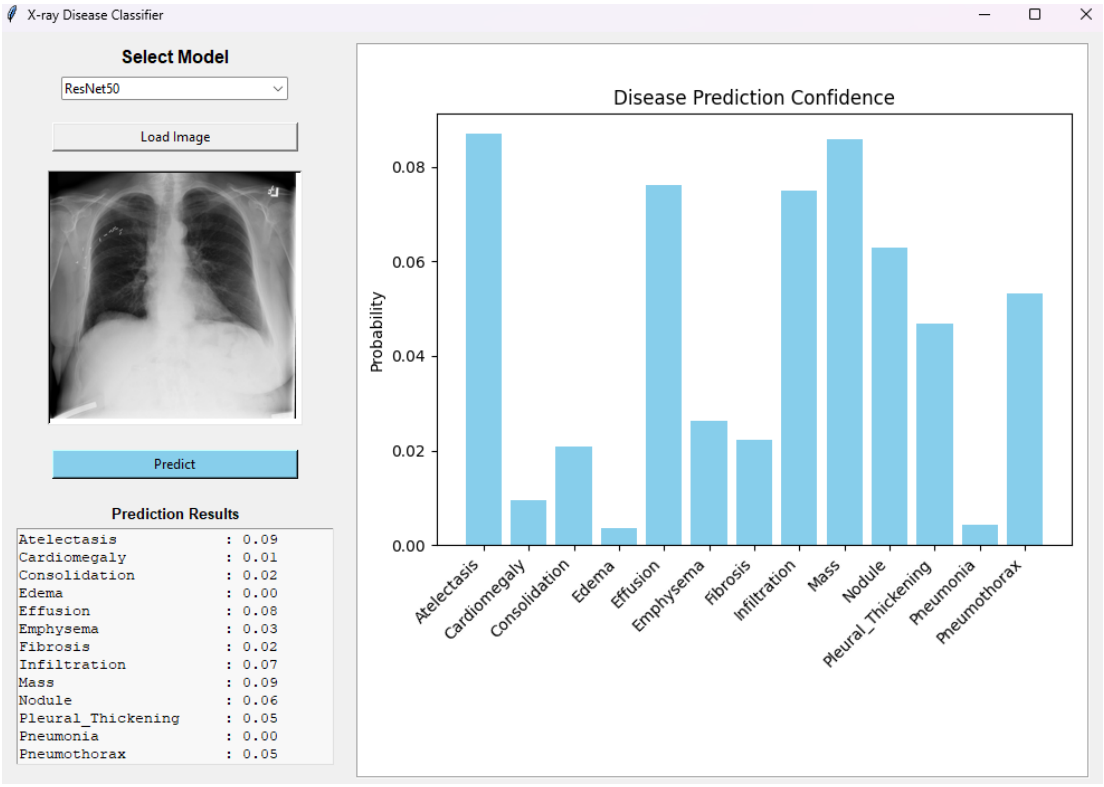

Рисунок 3.31 – Інтерфейс застосунку

Така подача результатів дозволяє користувачу і швидко визначити передбачення завдяки візуальному представленню так і подивитись на більш точні значення надані моделлю при передбаченні того чи іншого захворювання.

3.4 Перевірка моделей на тестових даних

Для прикладу роботи додатку та перевірки точності передбачення моделей на тестових даних, буде подано зображення рентгенівського знімку з очікуваним діагнозом “Cardiomegaly”.

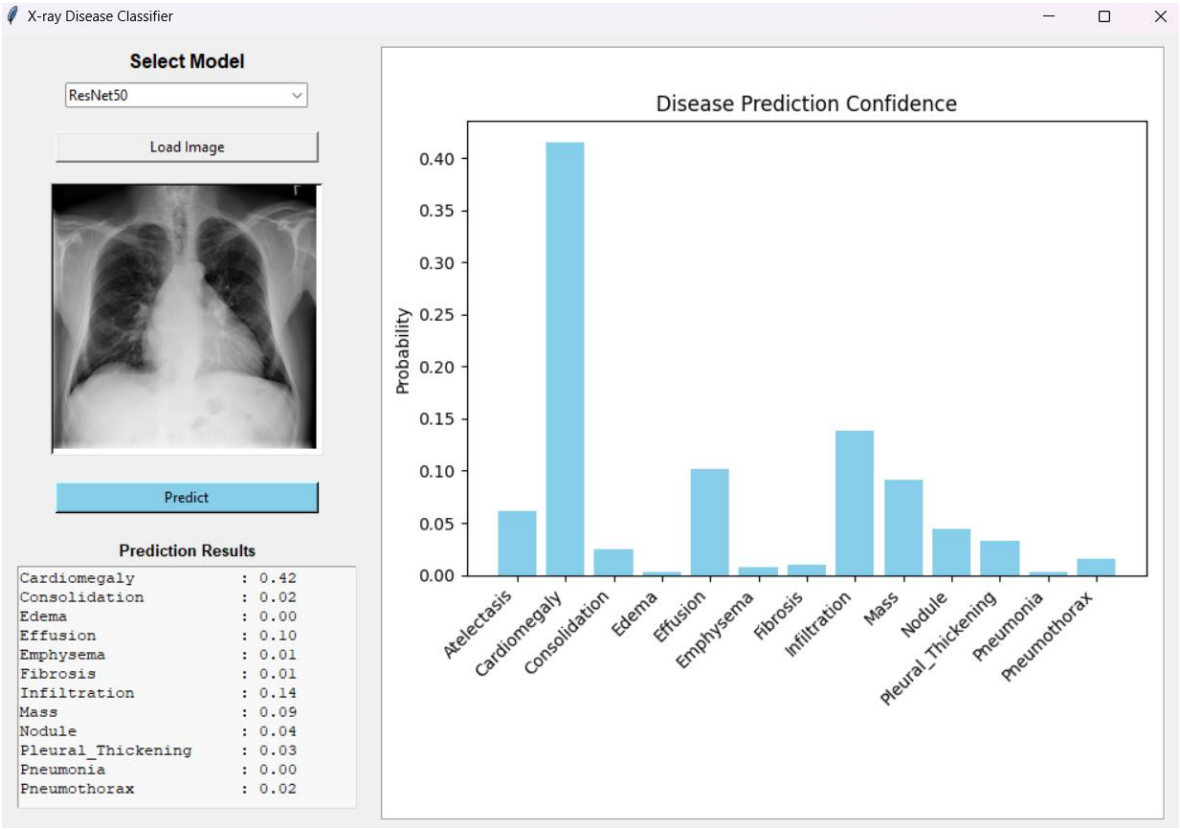

Рисунок 3.32 – Передбачення №1 моделі ResNet50

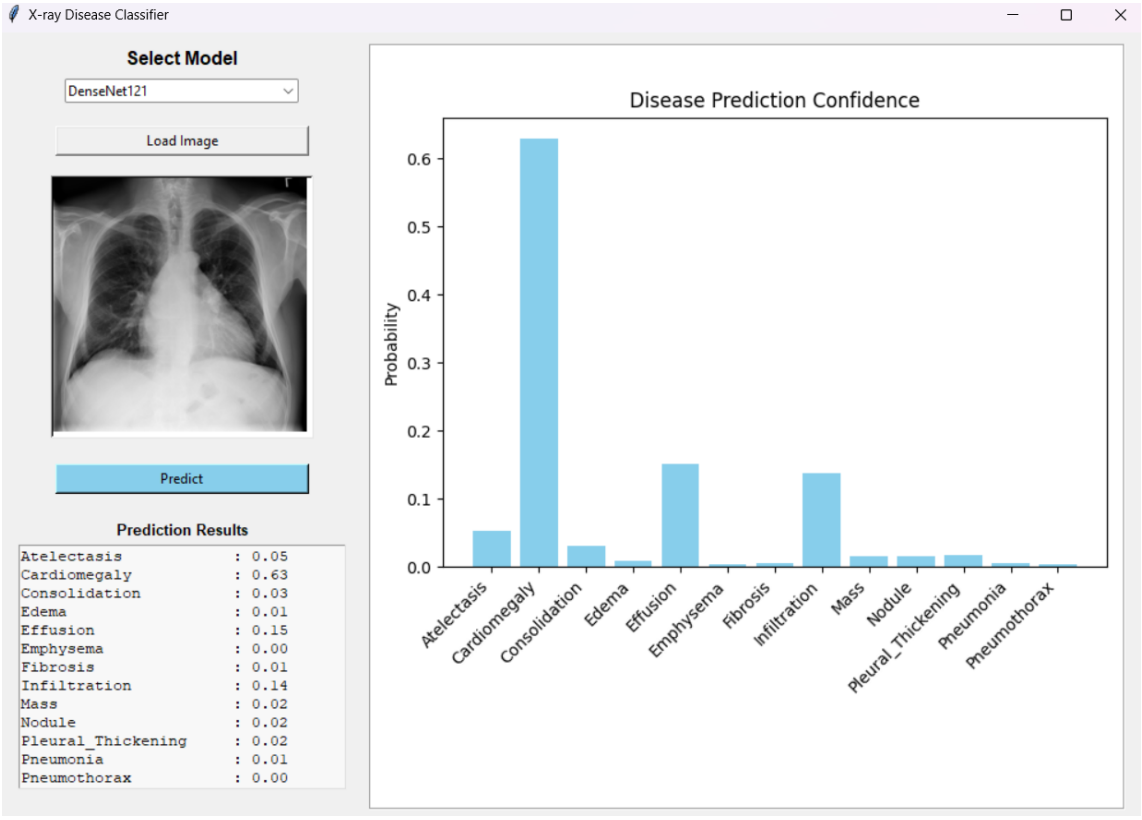

Рисунок 3.33 – Передбачення №1 моделі DenseNet121

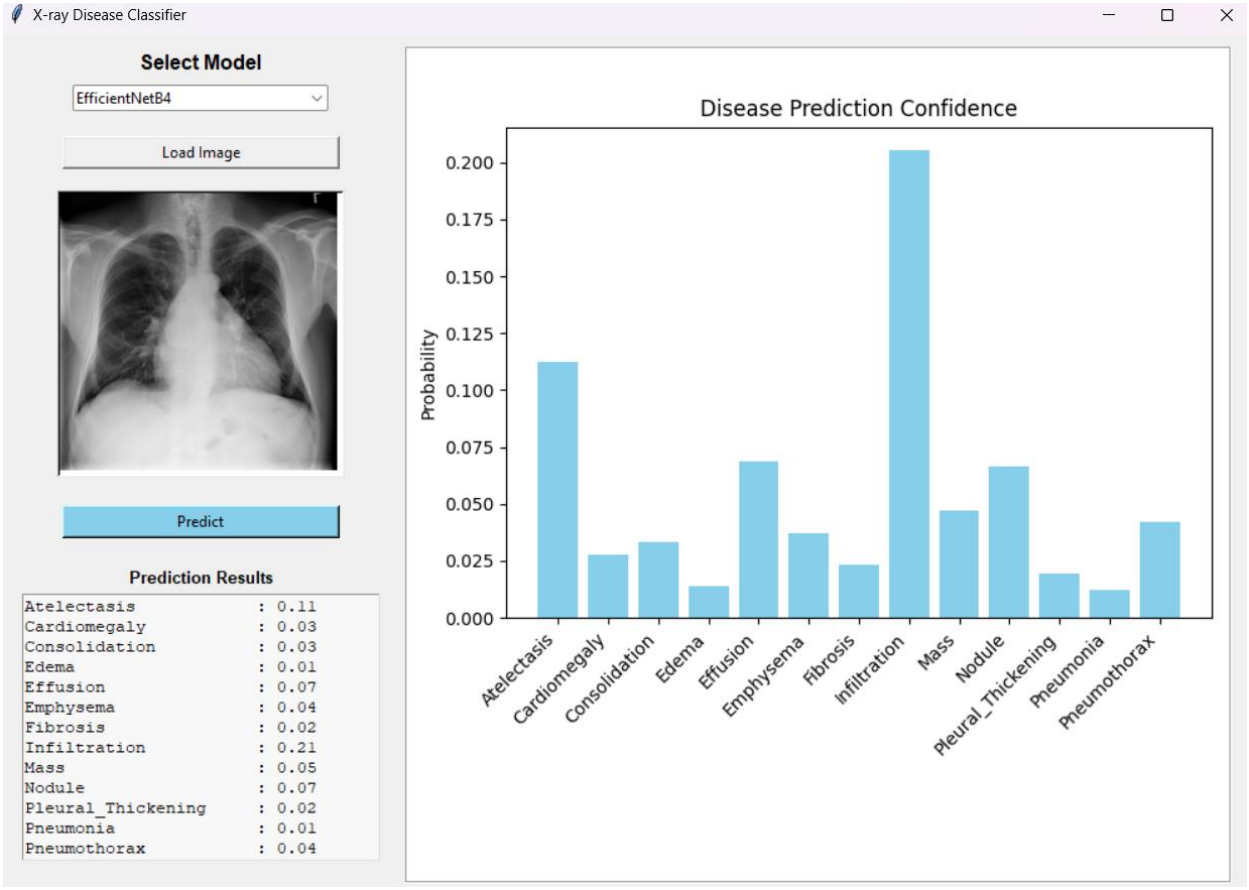

Рисунок 3.34 – Передбачення №1 моделі EfficientNetB4

Найвищу ймовірність для “Cardiomegaly” показала модель DenseNet121 – 0.63 (рис 3.33), що свідчить про значну впевненість у наявності цього стану. Це найкращий результат серед усіх моделей, і він може вказувати на правильний діагноз.

Модель ResNet50 також показала добру ймовірність “Cardiomegaly” – 0.42 (рис 3.32), що нижче, ніж у DenseNet121, але все ще є другим за величиною показником у своїй таблиці результатів. Тобто модель виявила ознаки патології, але з меншою впевненістю.

Модель EfficientNetB4 продемонструвала найнижчу ймовірність “Cardiomegaly” – лише 0.03 (рис 3.34), що суттєво відрізняється від результатів інших моделей. Найвищий показник у цій моделі – “Infiltration” (0.24), тоді як більшість інших значень також дуже низькі.

Так як набір даних також включає у себе багатоміткову класифікацію, буде доречно також перевірити точність передбачень моделей у випадках коли

діагностовано одразу декілька хвороб. Тому у наступному прикладі на вхід моделям буде надано зображення рентгенівського знімку з очікуваним результатом діагнозу “Emphysema” та “Pneumothorax”.

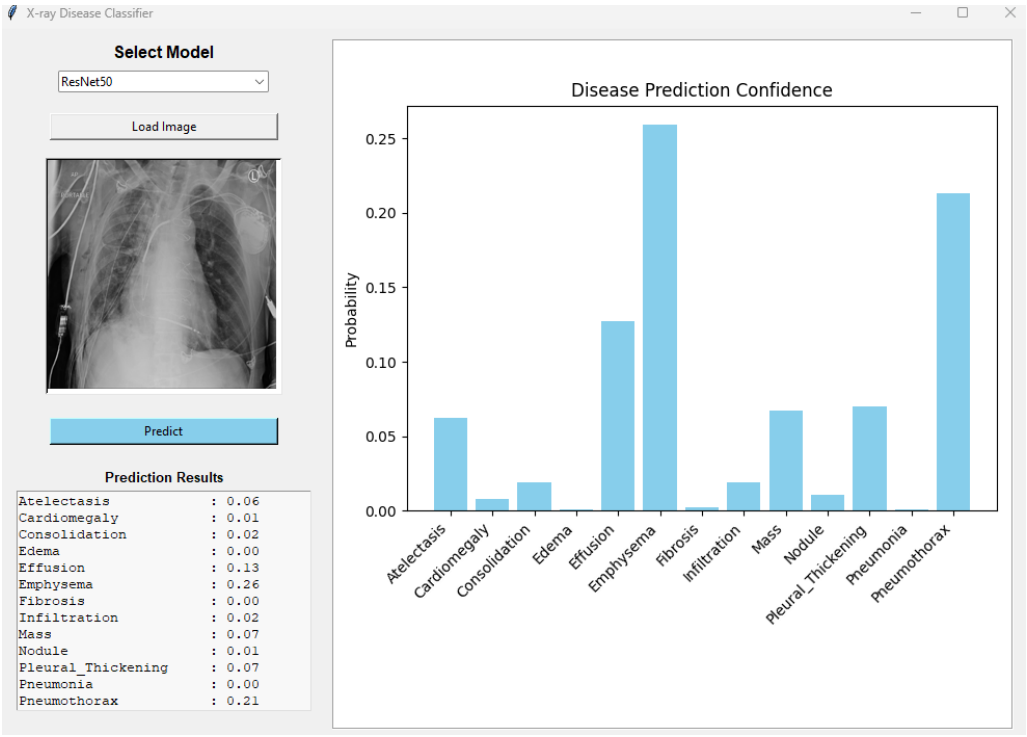

Рисунок 3.35 – Передбачення №2 моделі ResNet50

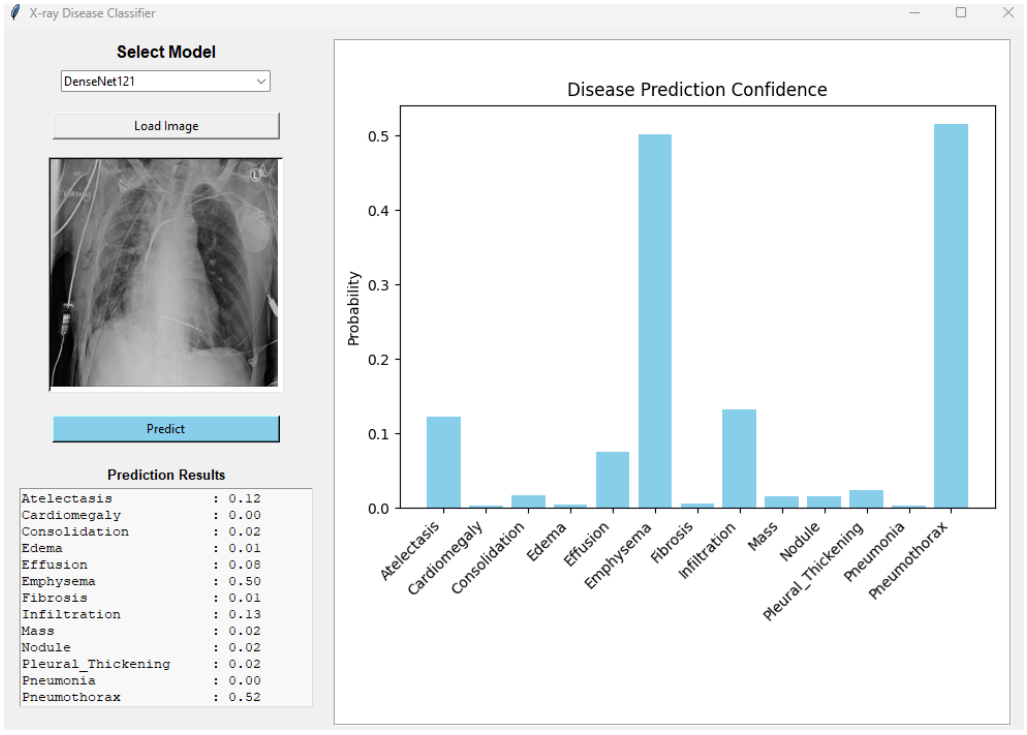

Рисунок 3.36 – Передбачення №2 моделі DenseNet121

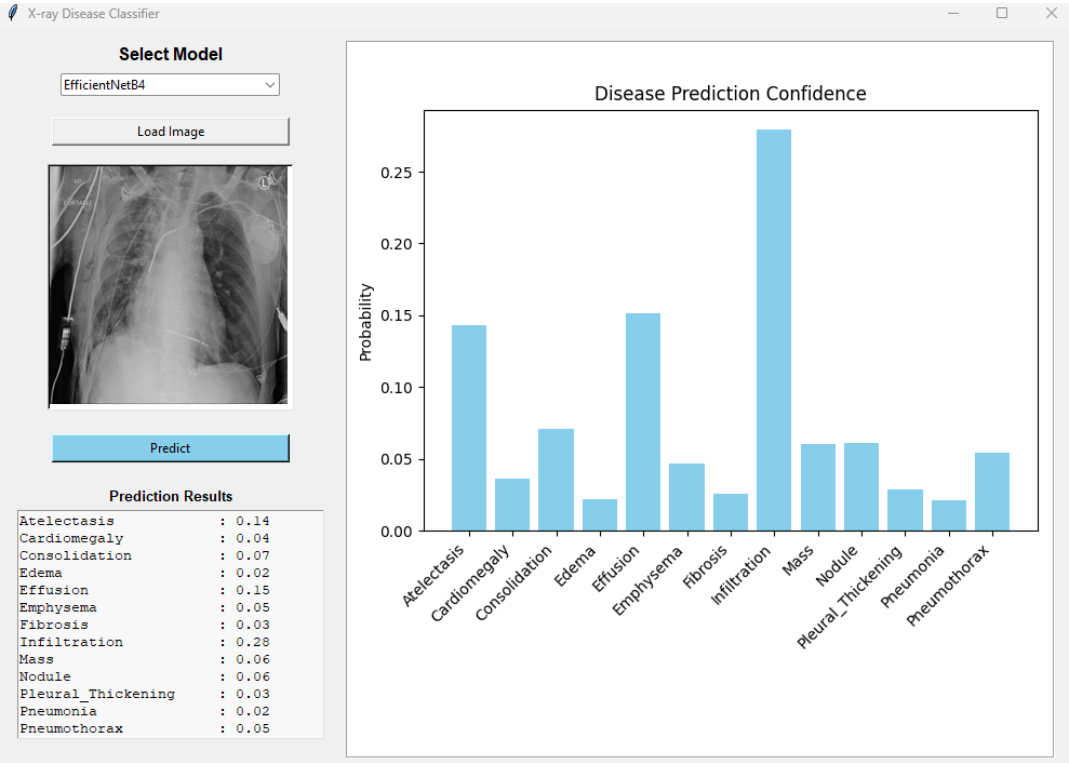

Рисунок 3.37 – Передбачення №2 моделі EfficientNetB4

На цьому прикладі модель ResNet50 показала менш впевнений результат ніж минулого разу, та все ж визначено 2 головних діагнози, які співпадають з очікуваними. Тут модель надала показники 0.26 для мітки “Emphysema” та 0.21 для “Pneumothorax” (рис 3.35). Це не є досить високими показниками, та все ж вони домінують серед інших.

Модель DenseNet121 знову показала найкращий результат у порівнянні з іншими моделями. У цьому прикладі модель досить точно визначили очікувані мітки “Emphysema” та “Pneumothorax” з ймовірностями 0.50 та 0.52 відповідно (рис 3.36). Модель також виключила інші мітки що є хорошим показником для моделі при задачі багатоміткової класифікації.

Так само як і в минулому прикладі, модель EfficientNetB4 показує найгірший результат і зовсім не виділяє очікувані мітки (рис 3.37). Натомість найвищий показник знову “Infiltration” (0.28), що є хибним для цього випадку.

Тепер можна також перевірити як моделі реагуватимуть на зразки які виключають усі захворювання, чи будуть усі значення низькими чи все таки

моделі старатимуться виділити якусь одну чи декілька міток. Тому у наступному прикладі на вхід моделям подається зображення рентгенівського знімку на якому не діагностовано жодної з наданих у мітках хвороб.

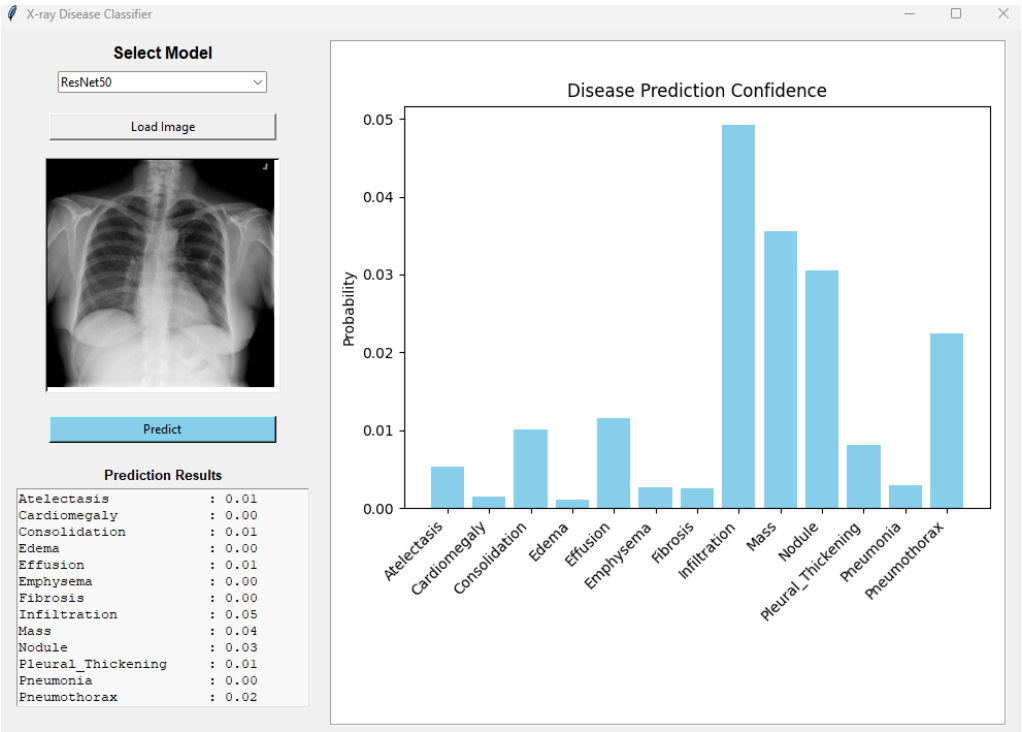

Рисунок 3.38 – Передбачення №3 моделі ResNet50

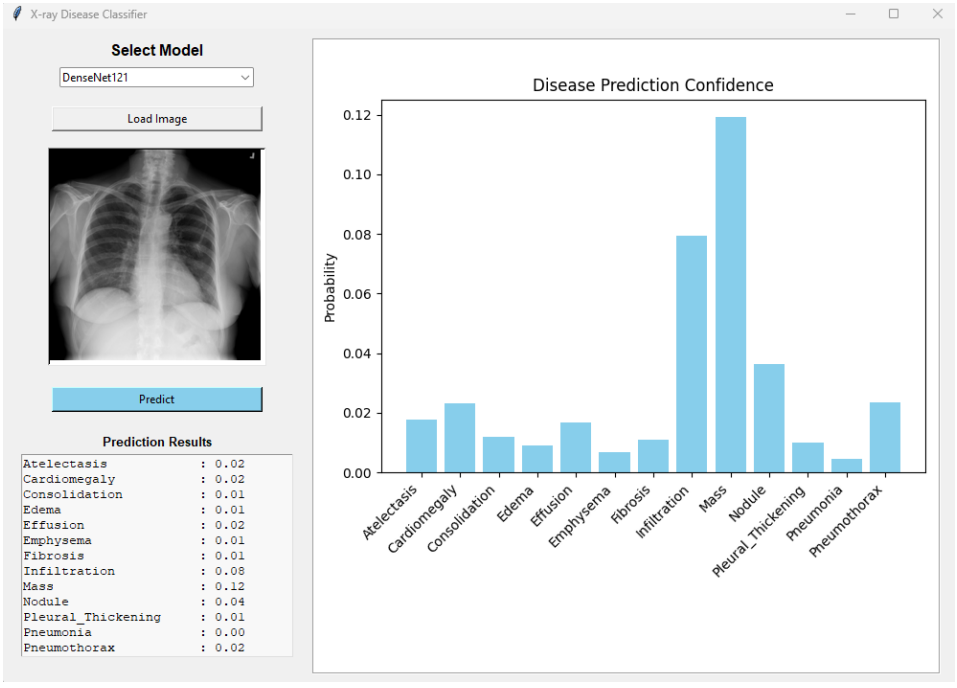

Рисунок 3.39 – Передбачення №3 моделі DenseNet121

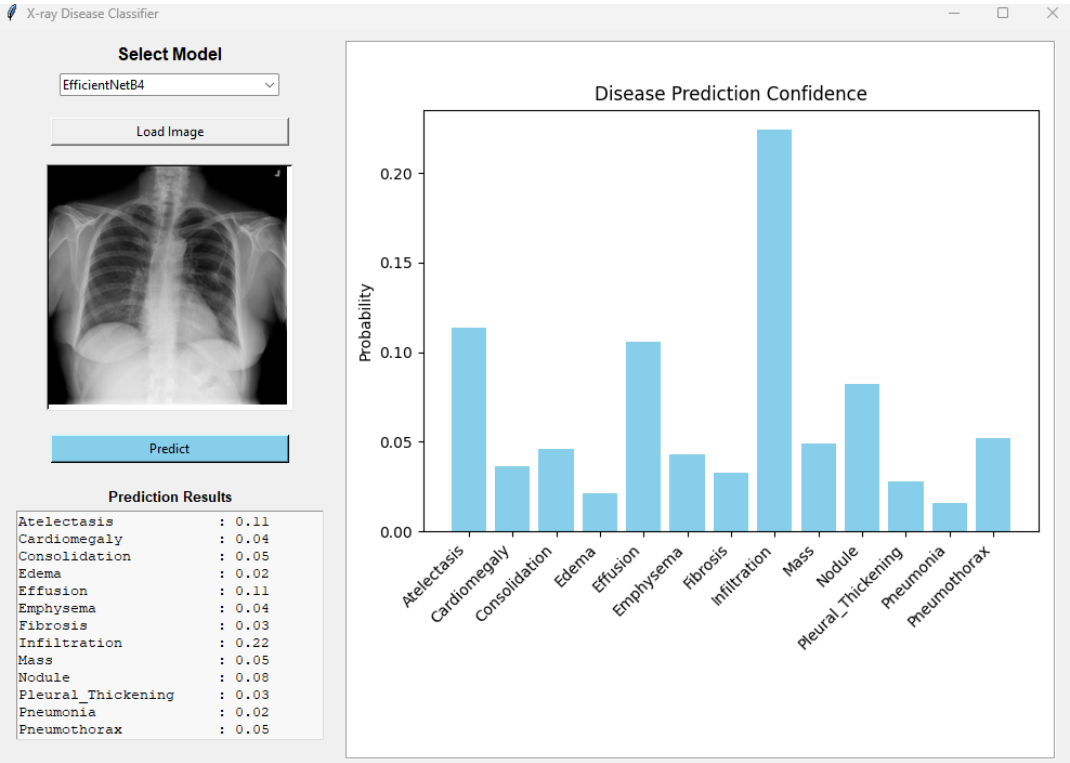

Рисунок 3.40 – Передбачення №3 моделі EfficientNetB4

У даному випадку модель ResNet50 з досить великою впевненістю виключила усі захворювання (рис 3.38). Тут найвищий показник мала мітка “Infiltration” (0.05), що є досить низьким значенням. Отже можна з впевненістю сказати, що у цьому випадку дана модель добре впоралась із завданням.

Натомість, модель DenseNet121 показала трохи вищі показники ніж попередня модель (рис 3.39). Найвищі показники мають мітки “Mass”(0.12) та “Infiltration”(0.8), що усе ще є досить низькими значеннями, та все ж вищими ніж у ResNet50. Тобто дана модель DenseNet121 має меншу впевненість у тому, що надане зображення рентгенівського знімку не має діагностованих захворювань.

Модель EfficientNetB4 показала результати такі самі як і у минулих прикладах (рис 3.40). Знову визначено найвищим показником “Infiltration” (0.22), що не є правильним у даному випадку.

3.5 Порівняння моделей нейромерж

Для порівняння якості навчання моделей глибокого навчання в задачі багатоміткової класифікації захворювань дихальних шляхів були використані

кілька основних метрик: AUC, Binary Accuracy та Loss. Кожна з них дає змогу оцінити модель з різних сторін і дозволяє виявити її сильні та слабкі сторони при роботі з медичними зображеннями.

Метрика AUC (Area Under the ROC Curve) є однією з найбільш поширених у задачах класифікації [38]. Вона відображає здатність моделі розрізняти позитивні та негативні приклади для кожного класу. AUC вимірює площу під ROC-кривою (графіком залежності істинно позитивної частоти від хибнопозитивної), де значення, близькі до 1, свідчать про високу точність моделі. У випадку багатоміткової класифікації AUC обчислюється окремо для кожного класу, а потім усереднюється. Для кожного класу спочатку будуються показники TPR (True Positive Rate) або чутливість та FPR (False Positive Rate):

$$TPR = \frac{TP}{TP+FN}, \tag{3.2}$$

де: TP (True Positives) — кількість випадків, коли модель правильно передбачила наявність класу.

FN (False Negatives) — кількість випадків, коли модель не виявила клас, хоча він був.

$$FPR = \frac{FP}{FP+TN}, \tag{3.3}$$

де: *FP* (False Positives) — кількість випадків, коли модель помилково передбачила наявність класу.

TN (True Negatives) — кількість випадків, коли модель правильно передбачила відсутність класу.

Потім обчислюється площа під ROC-кривою (AUC), яка є інтегралом залежності TPR(FPR). Для багатоміткової класифікації AUC усереднюється:

$$AUC_{avg} = \frac{1}{C}\sum_{i=1}^{C} AUC_i, \tag{3.4}$$

де: *AUC_i* — площа під ROC-кривою для класу i.

C — загальна кількість класів.

Binary Accuracy – це метрика, яка вимірює частку правильно класифікованих прикладів у задачі бінарної класифікації [38]. Для кожної ознаки (захворювання) передбачене значення порівнюється з фактичним, і якщо вони

збігаються (після округлення передбаченого значення), приклад вважається правильно класифікованим. Binary accuracy визначається як частка правильно класифікованих прикладів:

$$Binary\ Accuracy = \frac{1}{N}\sum_{i=1}^{N} 1[round(y_i^{pred}) = y_i^{true}], \quad (3.5)$$

де: *N* — загальна кількість прикладів у вибірці.

y_i^{pred} — передбачене значення моделі для прикладу i, зазвичай імовірність належності до класу.

y_i^{true} — фактична мітка прикладу i, яка дорівнює 0 або 1.

round(·) — округлення до 0 або 1 (поріг звичайно 0.5).

1[·] — індикаторна функція: повертає 1, якщо умова всередині дужок виконується, і 0 — якщо ні.

Loss-функція використовується для оцінки різниці між передбаченими й фактичними значеннями міток під час навчання [38]. У багатомітковій класифікації binary crossentropy дозволяє обробляти кожен клас незалежно, розраховуючи втрати по кожному з них і підсумовуючи результат. Чим менше значення loss, тим краще модель узгоджується з тренувальними даними. Втрати слугують основним індикатором процесу оптимізації моделі та її збіжності. Формула для функції втрат у багатомітковій бінарній класифікації (для кожного класу окремо, далі усереднюється):

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}(y_{ij}^{true} \cdot \log(y_{ij}^{pred}) + (1 - y_{ij}^{true}) \cdot \log(1 - y_{ij}^{pred})), \quad (3.6)$$

де: *N* — загальна кількість прикладів у вибірці.

C — кількість класів.

$y_{ij}^{pred} \in (0,1)$ — передбачена ймовірність моделі, що приклад i належить до класу j.

$y_{ij}^{true} \in \{0,1\}$ — справжня мітка класу j для прикладу i: 1, якщо клас присутній, інакше 0.

Ця функція ефективна для багатоміткових задач, бо розглядає кожен клас незалежно, дозволяючи моделі вчитися одночасно розпізнавати кілька станів.

Таблиця 3.1 – Порівняння моделей нейромереж для задачі класифікації

	AUC	Binary Accuracy	Loss
ResNet50	0.7867	0.9434	0.1758
DenseNet121	0.8059	0.9491	0.1557
EfficientNetB4	0.6007	0.9470	0.1974

Аналізуючи таблицю 3.1, можна зробити висновок, що найкращу загальну продуктивність показала модель DenseNet121, яка досягла найвищого значення AUC (0.8059) та найнижчого значення втрат (Loss = 0.1557). Модель ResNet50 трохи поступається, з нижчим AUC (0.7867) та вищим Loss (0.1758), проте також демонструє високу точність. Натомість EfficientNetB4 показала значно гірший результат за AUC (0.6007), що свідчить про слабку здатність моделі розрізняти класи, попри високу Binary Accuracy (0.9470). Це може вказувати на те, що модель переважно правильно передбачає відсутність захворювань, але не справляється з їх виявленням.

Цей аналіз також підтверджується тестуванням моделей на зображеннях рентгенівських знімків з різними діагнозами. Модель DenseNet121 як за показниками з таблиці так і за перевіркою на тестових даних показала найращий результат серед обраних архітектур, а EfficientNetB4 з тих же показників показує найгірший результат і не демонструє ознак навчання для задачі класифікації.

Висновки до розділу 3

У цьому розділі було реалізовано повний цикл побудови системи класифікації захворювань дихальних шляхів на основі рентгенівських знімків. На етапі підготовки даних здійснено попереднє опрацювання набору ChestX-ray14, зокрема очищення, кодування міток у формат багатоміткової класифікації та розділення на тренувальну й тестову вибірки. Далі було створено та навчено кілька моделей згорткових нейронних мереж, зокрема ResNet50, DenseNet121 та EfficientNetB4.

Для використання та тестування точності передбачення моделей було створено застосунок, який надає можливість користувачу отримати передбачення конкретної моделі щодо завантаженого зображення.

Моделі було протестовано на тестових даних, і для кожної обчислено метрики якості, зокрема AUC, Binary Accuracy та функцію втрат. Проведене порівняння показало, що найвищу якість класифікації демонструє модель DenseNet121, яка перевершила інші як за точністю, так і за здатністю розрізняти класи (AUC). З іншого боку, модель EfficientNetB4 показала найгірші результати, що свідчить про складність її адаптації до даного завдання без додаткового налаштування. Таким чином, було визначено найефективнішу архітектуру для подальшого використання в розробці програмного застосунку.

ВИСНОВКИ

У результаті виконання цієї роботи було досліджено проблему класифікації захворювань дихальних шляхів за допомогою методів глибокого навчання. Було виконано всебічний аналіз сучасних методів машинного навчання, зокрема технологій глибокого навчання та згорткових нейронних мереж, які активно застосовуються в медичній діагностиці. Розглянуто переваги та недоліки різних архітектур, а також їх придатність до задач класифікації зображень.

Для виконання задачі класифікації було здійснено вибір та попередню підготовку навчального набору даних, що містить рентгенівські знімки грудної клітки з відповідними мітками захворювань. Дані були адаптовані до формату, придатного для навчання моделей.

Було проведено розробку та навчання моделей нейронних мереж, заснованих на архітектурах ResNet50, DenseNet121 та EfficientNetB4. Для кожної моделі було здійснено навчання із застосуванням оптимізатора Adam, функції втрат binary crossentropy та метрик AUC і Binary Accuracy. Процес навчання супроводжувався моніторингом показників та використанням callback-функцій для збереження найкращих результатів.

На етапі оцінки продуктивності моделей було проведено тестування та порівняння результатів. Модель DenseNet121 досягла найкращого значення AUC — 0.8059, демонструючи найвищу здатність розпізнавати захворювання. Модель ResNet50 показала дещо гірші, але все ще придатні результати. У свою чергу, EfficientNetB4 не змогла забезпечити якісне навчання в обраній задачі, продемонструвавши AUC лише 0.6007.

Загалом, можна зробити висновки, що моделі на базі DenseNet121 та ResNet50 мають потенціал до використання в медичній практиці для автоматизованої діагностики захворювань дихальної системи за рентгенівськими знімками. Система, створена в межах роботи, є доказом ефективності глибоких нейронних мереж у реальних медичних задачах і може бути використана як основа для подальшого вдосконалення діагностичних інструментів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Важливість діагностики та лікування захворювань легень: розуміння, симптоми та методи відновлення. URL: https://akwatur.com/blog/vazhlyvist-diagnostyky-ta-likuvannya-zahvoryuvan-legen-rozuminnya-symptomy-ta-metody-vidnovlennya/ (дата звернення: 22.04.2025).

2. Передові методики діагностики хвороб дихальних шляхів: чим допоможе високотехнологічне обладнання. URL: https://novamed.in.ua/simejnyj-likar/peredovi-metodyky-diahnostyky-khvorob-dykhalnykh-shliakhiv-chym-dopomozhe-vysokotekhnolohichne-obladnannia/ (дата звернення: 22.04.2025).

3. Оглядова рентгенограма грудної клітки. URL: https://empendium.com/ua/manual/chapter/B72.I.B.4.1. (дата звернення: 22.04.2025).

4. РЕНТГЕНОЛОГІЧНІ МЕТОДИ ДОСЛІДЖЕННЯ. URL: http://dspace.zsmu.edu.ua/bitstream/123456789/9538/1/%D0%9F%D0%BE%D1%81%D1%96%D0%B1%20%D0%A0%D0%B5%D0%BD%D1%82%D0%B3%D0%B5%D0%BD%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4.pdf (дата звернення: 22.04.2025).

5. ДОБРОВСЬКА, Людмила Миколаївна; ДОБРОВСЬКА, Ірина Арбіківна. Теорія та практика нейронних мереж. 2015.

6. ТИШ, Є. Нейромережеві технології в задачах медичної діагностики. Матеріали II науково-технічної конференції „Інформаційні моделі, системи та технології “, 2012, 39-39.

7. Нейромережа: що це, як працює, найкращі приклади. High Bar Journal. URL: https://journal.gen.tech/post/sho-take-neiromerezhi (дата звернення: 25.04.2025).

8. Бородіна, О. О., and Д. О. Клименко. Штучний інтелект як революція в медицині. Diss. Полтавський національний технічний університет імені Юрія Кондратюка, 2018.

9. Purwadhika | Neural Network Implementation for Image Classification using CNN. URL: https://purwadhika.com/blog/neural-network-implementation-for-

image-classification-using-cnn (date of access: 25.04.2025).

10. Ullah, Hakeem, et al. "Stability analysis of MHD Jeffery–Hamel flow using artificial neural network." International Journal of Thermofluids 24 (2024): 100834.

11. Learn FluCoMa. URL: https://learn.flucoma.org/learn/mlp-parameters/ (date of access: 25.04.2025).

12. Як працює згорткова нейронна мережа. URL: https://robotdreams.cc/uk/blog/209-kak-rabotaet-svertochnaya-neyronnaya-set (дата звернення: 25.04.2025).

13. CNN Architecture: 5 Layers Explained Simply. URL: https://www.upgrad.com/blog/basic-cnn-architecture/ (date of access: 25.04.2025).

14. Keidar, Daphna, et al. "COVID-19 classification of X-ray images using deep neural networks." European radiology 31.12 (2021): 9654-9663.

15. Lakhani, Paras. "Deep convolutional neural networks for endotracheal tube position and X-ray image classification: challenges and opportunities." Journal of digital imaging 30 (2017): 460-468.

16. Moujahid, Hicham, et al. "Convolutional neural network based classification of patients with pneumonia using X-ray lung images." Advances in Science, Technology and Engineering Systems Journal 5.5 (2020): 167-175.

17. Alshmrani, Goram Mufarah M., et al. "A deep learning architecture for multi-class lung diseases classification using chest X-ray (CXR) images." Alexandria Engineering Journal 64 (2023): 923-935.

18. Saraiva, Arata Andrade, et al. "Classification of Images of Childhood Pneumonia using Convolutional Neural Networks." Bioimaging. 2019.

19. Papers with Code - ChestX-ray14 Dataset. The latest in Machine Learning. URL: https://paperswithcode.com/dataset/chestx-ray14 (date of access: 25.04.2025).

20. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

21. Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for

convolutional neural networks." International conference on machine learning. PMLR, 2019.

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

23. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature. URL: https://www.nature.com/articles/nature14539/ (дата звернення: 26.04.2025).

24. Fully connected layer vs. convolutional layer: explained. Built In. URL: https://builtin.com/machine-learning/fully-connected-layer/ (date of access: 26.04.2025).

25. ResNet: The basics and 3 ResNet extensions. Datagen. URL: https://datagen.tech/guides/computer-vision/resnet/ (дата звернення: 29.04.2025).

26. Tsang S.-H. Review: DenseNet – dense convolutional network (image classification). Towards Data Science. URL: https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803/ (date of access: 29.04.2025).

27. Malingan N. EfficientNet - scaler topics. Scaler Topics. URL: https://www.scaler.com/topics/deep-learning/efficientNet/ (date of access: 30.04.2025).

28. Welcome to Python.org. Python.org. URL: https://www.python.org/ (date of access: 28.04.2025).

29. TensorFlow. TensorFlow. URL: https://www.tensorflow.org/ (date of access: 08.05.2025).

30. Project Jupyter. URL: https://jupyter.org/ (date of access: 28.04.2025).

31. What is jupyter notebook?. Domino Data Lab | Unleash Data Science at Scale. URL: https://domino.ai/data-science-dictionary/jupyter-notebook/ (date of access: 01.05.2025).

32. pandas. PyPI. URL: https://pypi.org/project/pandas/ (date of access: 01.05.2025).

33. What is numpy? – numpy v1.26 manual. NumPy. URL: https://numpy.org/doc/stable/user/whatisnumpy.html (date of access: 02.05.2025).

34. Getting started. scikit-learn. URL: https://scikit-

learn.org/stable/getting_started.html (date of access: 05.05.2025).

35. Matplotlib documentation – Matplotlib 3.8.3 documentation. Matplotlib – Visualization with Python. URL: https://matplotlib.org/stable/index.html (date of access: 10.05.2025).

36. Keras documentation: About Keras 3. Keras: Deep Learning for humans. URL: https://keras.io/about/ (date of access: 14.05.2025).

37. Deep learning for computer vision: a brief review / A. Voulodimos et al. Computational intelligence and neuroscience. 2018. T. 2018. C. 1–13. URL: https://doi.org/10.1155/2018/7068349 (date of access: 16.05.2025).

38. Pei-Xia, Sun, Lin Hui-Ting, and Luo Tao. "Learning discriminative CNN features and similarity metrics for image retrieval." 2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC). IEEE, 2016.

ДОДАТОК А
Код навчання нейронних мереж

```
import numpy as np
import pandas as pd
import os
from glob import glob
import matplotlib.pyplot as plt
from itertools import chain
import tensorflow as tf
import seaborn as sns
from tensorflow.keras import backend as K
from tensorflow.keras.applications import ResNet50, DenseNet121,
EfficientNetB4
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.config import threading
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.metrics import AUC, BinaryAccuracy
from tensorflow.keras.models import load_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc

all_xray_df = pd.read_csv('./Chest_X-rays/Data_Entry_2017.csv')

all_image_paths = {
    os.path.basename(p): p
    for p in glob(os.path.join('.', 'Chest_X-rays', 'images*', '*',
'*.png'))
}
print(f'Scans available: {len(all_image_paths)} | Metadata entries:
{all_xray_df.shape[0]}')

all_xray_df['path'] = all_xray_df['Image Index'].map(all_image_paths.get)
all_xray_df['Patient Age'] = all_xray_df['Patient Age'].astype(int)
all_xray_df.sample(3)

label_counts = all_xray_df['Finding Labels'].value_counts().iloc[:15]

fig, ax1 = plt.subplots(figsize=(12, 8))
ax1.bar(np.arange(len(label_counts)) + 0.5, label_counts.values)
```

```
ax1.set_xticks(np.arange(len(label_counts)) + 0.5)
xtick_labels = ax1.set_xticklabels(label_counts.index, rotation=90)
plt.tight_layout()
plt.show()

all_xray_df['Finding Labels'] = all_xray_df['Finding
Labels'].str.replace('No Finding', '')
all_labels = np.unique(
    list(chain.from_iterable(all_xray_df['Finding Labels'].str.split('|')))
)
all_labels = [label for label in all_labels if label.strip()]
print(f'Extracted Labels ({len(all_labels)}): {all_labels}')
for c_label in all_labels:
    if len(c_label) > 1:
        all_xray_df[c_label] = all_xray_df['Finding Labels'].map(
            lambda findings: 1.0 if c_label in findings else 0.0
        )
all_xray_df.sample(3)

MIN_CASES = 1000
all_labels = [label for label in all_labels if all_xray_df[label].sum() >
MIN_CASES]

print(f'Filtered Labels ({len(all_labels)}):',
      [(label, int(all_xray_df[label].sum())) for label in all_labels])

all_xray_df = all_xray_df[['Image Index'] + all_labels]

all_image_paths = {
    os.path.basename(x): x
    for x in glob(os.path.join('.', 'Chest_X-rays', 'images_*', 'images',
'*.png'))
}

all_xray_df['full_path'] = all_xray_df['Image
Index'].map(all_image_paths.get)
all_xray_df = all_xray_df.dropna(subset=['full_path'])

train_df, test_df = train_test_split(all_xray_df, test_size=0.2,
random_state=42)

def create_generator(dataframe, x_col, y_col, datagen, batch_size=16,
target_size=(224, 224)):
    return datagen.flow_from_dataframe(
```

```
        dataframe=dataframe,
        x_col=x_col,
        y_col=y_col,
        directory=None,
        class_mode='raw',
        batch_size=batch_size,
        target_size=target_size,
        shuffle=True
    )

train_datagen = ImageDataGenerator(
    rescale=1./255,

preprocessing_function=tf.keras.applications.efficientnet.preprocess_input,
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)


test_datagen = ImageDataGenerator(
    rescale=1./255
)
train_generator = create_generator(train_df, 'full_path', all_labels,
train_datagen, batch_size=16)
test_generator = create_generator(test_df, 'full_path', all_labels,
test_datagen, batch_size=16)

base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3)) # для ResNet50
# base_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=(224, 224, 3)) # для DenseNet121
# base_model = EfficientNetB4(weights='imagenet', include_top=False,
input_shape=(224, 224, 3)) # для EfficientNetB4
x = base_model.output
x = GlobalAveragePooling2D ()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(all_labels), activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=[
        BinaryAccuracy(name='binary_accuracy'),
        AUC(name='auc', multi_label=True)
    ]
)
model.summary()

callbacks = [
    EarlyStopping(monitor='val_auc', patience=5, mode='max', verbose=1),
    ModelCheckpoint('model_name.keras', monitor='val_auc',
                    save_best_only=True, mode='max', verbose=1)
]


history = model.fit(
    train_generator,
    steps_per_epoch=200,
    validation_data=test_generator,
    validation_steps=50,
    epochs=20,
    verbose=1,
    callbacks=callbacks
)

def visualize_training(history, lw = 3):
    plt.figure(figsize=(10,6))
    plt.plot(history.history['auc'], label = 'training', marker = '*',
linewidth = lw)
    plt.plot(history.history['val_auc'], label = 'validation', marker =
'o', linewidth = lw)
    plt.title('Training AUC vs Validation AUC')
    plt.xlabel('Epochs')
    plt.ylabel('AUC')
    plt.legend(fontsize = 'x-large')
    plt.show()

    plt.figure(figsize=(10,6))
    plt.plot(history.history['loss'], label = 'training', marker = '*',
linewidth = lw)
    plt.plot(history.history['val_loss'], label = 'validation', marker =
'o', linewidth = lw)
```

```
    plt.title('Training Loss vs Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(fontsize = 'x-large')
    plt.show()
visualize_training(history)

y_true = []
y_pred = []

for x_batch, y_batch in test_generator:
    y_true.append(y_batch)
    y_pred.append(model.predict(x_batch, verbose=0))
    if len(y_true) * test_generator.batch_size >= test_generator.n:
        break

y_true = np.concatenate(y_true)
y_pred = np.concatenate(y_pred)

plt.figure(figsize=(12, 10))

for i, label in enumerate(all_labels):
    fpr, tpr, _ = roc_curve(y_true[:, i], y_pred[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{label} (AUC = {roc_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--')  # базова лінія
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves per Label')
plt.legend(loc='lower right', fontsize='small')
plt.grid(True)
plt.show()
```

ДОДАТОК Б
Код програмного застосунку

```
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
from PIL import Image, ImageTk
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

MODEL_PATHS = {
    "ResNet50": "resnet50_v2.keras",
    "DenseNet121": "densenet121.keras",
    "EfficientNetB4": "efficientnetb4.keras"
}

CLASS_LABELS = [
    "Atelectasis", "Cardiomegaly", "Consolidation", "Edema",
    "Effusion", "Emphysema", "Fibrosis", "Infiltration",
    "Mass", "Nodule", "Pleural_Thickening", "Pneumonia", "Pneumothorax"
]

class ChestXRayApp:
    def __init__(self, root):
        self.root = root
        self.root.title("X-ray Disease Classifier")
        self.root.geometry("1000x600")
        self.root.minsize(900, 500)

        self.image_path = None
        self.tk_image = None
        self.graph_canvas = None

        # Main layout
        main_frame = tk.Frame(root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Left side: controls & image
        left_frame = tk.Frame(main_frame)
        left_frame.pack(side=tk.LEFT, fill=tk.Y, padx=10)

        # Right side: graph
```

```
        self.graph_frame = tk.Frame(main_frame, bd=2, relief=tk.GROOVE)
        self.graph_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True,
padx=10)

        # === Left Section ===
        tk.Label(left_frame, text="Select Model", font=("Helvetica", 12,
"bold")).pack(pady=(0, 5))

        self.models = list(MODEL_PATHS.keys())
        self.selected_model = tk.StringVar(value=self.models[0])
        self.model_menu = ttk.Combobox(left_frame,
textvariable=self.selected_model, values=self.models, state="readonly",
width=30)
        self.model_menu.pack(pady=(0, 15))

        self.load_button = tk.Button(left_frame, text="Load Image",
command=self.load_image, width=30)
        self.load_button.pack(pady=5)

        # Image display
        self.canvas = tk.Canvas(left_frame, width=224, height=224,
bg='white', bd=2, relief=tk.SUNKEN)
        self.canvas.pack(pady=10)

        self.predict_button = tk.Button(left_frame, text="Predict",
command=self.predict, width=30, bg="skyblue")
        self.predict_button.pack(pady=10)

        # Results text area
        tk.Label(left_frame, text="Prediction Results", font=("Helvetica",
10, "bold")).pack(pady=(10, 0))
        self.result_text = tk.Text(left_frame, height=13, width=35,
state='disabled', bg="#f8f8f8")
        self.result_text.pack(pady=(2, 10))

        # Load models
        self.model_objects = {name: load_model(path, compile=False) for
name, path in MODEL_PATHS.items()}

    def load_image(self):
        path = filedialog.askopenfilename(filetypes=[("Image files", "*.png
*.jpg *.jpeg")])
        if not path:
            return
```

```
        self.image_path = path
        img = Image.open(self.image_path).convert("RGB")
        img = img.resize((224, 224))
        self.tk_image = ImageTk.PhotoImage(img)
        self.canvas.create_image(0, 0, anchor=tk.NW, image=self.tk_image)

    def preprocess_image(self):
        img = image.load_img(self.image_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0
        return img_array


    def predict(self):
        if not self.image_path:
            print("No image loaded.")
            return

        model_name = self.selected_model.get()
        model = self.model_objects[model_name]
        input_tensor = self.preprocess_image()

        prediction = model.predict(input_tensor)[0]
        self.show_prediction(prediction)

    def show_prediction(self, prediction):
        # Update result text box
        self.result_text.configure(state='normal')
        self.result_text.delete('1.0', tk.END)
        for label, prob in zip(CLASS_LABELS, prediction):
            self.result_text.insert(tk.END, f"{label:22s} : {prob:.2f}\n")
        self.result_text.configure(state='disabled')

        # Update graph
        if self.graph_canvas:
            self.graph_canvas.get_tk_widget().destroy()

        fig, ax = plt.subplots(figsize=(7, 4))
        ax.bar(CLASS_LABELS, prediction, color='skyblue')
        ax.set_xticklabels(CLASS_LABELS, rotation=45, ha="right")
        ax.set_ylabel("Probability")
        ax.set_title("Disease Prediction Confidence")
        fig.tight_layout()
```

```
        self.graph_canvas = FigureCanvasTkAgg(fig, master=self.graph_frame)
        self.graph_canvas.draw()
        self.graph_canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

if __name__ == "__main__":
    root = tk.Tk()
    app = ChestXRayApp(root)
    root.mainloop()
```