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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Веб-додаток для аналізу рентгенівських знімків з використанням нейронної мережі”

Виконав: студент 4 курсу, групи ТР-15
Теличко Євгеній Віталійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент каф. ЦТЕ, Володимир РУДИК _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент професор каф. ТАЕ, д.т.н., Михайло АБДУЛІН _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент каф. ЦТЕ, Антон ПАСІЧНЮК _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

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

Київ - 2025

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)
спеціальність 122 "Комп'ютерні науки"
Освітньо-професійна програма "Цифрові технології в енергетиці"

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА
(підпис)
"___" _____2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Теличку Євгенію Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи "Веб-додаток для аналізу рентгенівських знімків з використанням нейронної мережі"

Науковий керівник Рудик Володимир Іванович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "02" __червня__ 2025 року № 1875-с

2. Термін подання студентом роботи 09.06.2025 р.

3. Вихідні дані до роботи мова програмування Python, фреймворк TensorFlow (Keras API), бібліотека обробки зображень OpenCV, веб-фреймворк Streamlit, інструменти візуалізації Matplotlib і Seaborn, бібліотеки NumPy, pandas, Pillow, scikit-learn, середовище розробки PyCharm, інтерактивне середовище Jupyter Notebook.

4. Перелік питань, які потрібно розробити 1) провести аналіз існуючих методів та рішень для аналізу рентгенівських знімків засобами нейронних мереж; 2) обрати набір даних, методи та програмні засоби для реалізації поставленої мети, а також обґрунтувати їх вибір; 3) виконати аналіз та попередню обробку обраного набору даних; 4) розробити кілька моделей нейронних мереж та виконати їх порівняльний аналіз для вибору найкращої архітектури; 5) створити програмний застосунок для роботи з моделлю нейронної мережі, яка продемонструє найкращі показники ефективності.

5. Орієнтований перелік ілюстративного матеріалу графіки розподілу набору даних, діаграма, що демонструє архітектуру нейронної мережі; інтерфейс користувача.

6. Дата видачі завдання 13.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	13.09.2024	Виконано
2.	Аналіз методів та засобів розв’язання задачі	14-20.04.25	Виконано
3.	Розробка архітектури та загальної структури системи	21.04.2025-11.05.2025	Виконано
4.	Розробка окремих підсистем	11.05.2025-12.05.2025	Виконано
5.	Програмна реалізація системи	12.05.2025-13.05.2025	Виконано
6.	Оформлення пояснювальної записки	14.05.2025-27.05.2025	Виконано
7.	Захист програмного забезпечення	14.05.2025	Виконано
8.	Передзахист	28.05.2025	Виконано
9.	Захист		Виконано

Студент _____ Євгеній ТЕЛИЧКО
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник _____ Володимир РУДИК
(підпис) (ім’я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 70 сторінках, містить 25 ілюстрацій, 4 таблиці, 1 додаток, 20 джерел у переліку посилань.

Мета роботи — створення програмного забезпечення для виявлення переломів на рентгенівських знімках.

Методи та засоби: згорткові нейронні мережі (CNN), мова програмування Python, бібліотеки TensorFlow, Keras, OpenCV, scikit-learn, фреймворк Streamlit.

Результат — веб-додаток для автоматичної класифікації знімків, який інтегрує попередньо навчену модель глибинного навчання та забезпечує зручну взаємодію з користувачем.

Ключові слова: ГЛИБИННЕ НАВЧАННЯ, РЕНТГЕН, ПЕРЕЛОМ, КЛАСИФІКАЦІЯ, CNN, PYTHON

ABSTRACT

The thesis consists of 70 pages, 25 illustrations, 4 tables, 1 appendix, and 20 references.

Purpose: to create software for detecting fractures in X-ray images.

Methods and tools: Convolutional Neural Networks (CNN), Python programming language, TensorFlow, Keras, OpenCV, scikit-learn, Streamlit framework.

The result is a web application for automatic image classification that integrates a pre-trained deep learning model and provides convenient user interaction.

Keywords: DEEP LEARNING, X-RAY, FRACTURE, CLASSIFICATION, CNN, PYTHON

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AUC (*Area Under Curve*) — площа під ROC-кривою; метрика, що відображає здатність моделі класифікувати об’єкти незалежно від вибраного порогу. Значення AUC ближче до 1 свідчить про високу якість класифікації.

BatchNormalization — техніка нормалізації вихідних значень шару нейронної мережі в межах одного батчу, що покращує стабільність і швидкість навчання.

CNN (*Convolutional Neural Network*) — згорткова нейронна мережа; архітектура глибинного навчання, оптимізована для роботи з зображеннями.

Dropout — метод регуляризації, при якому випадковим чином «вимикаються» деякі нейрони під час навчання з метою запобігання перенавчанню.

Flatten — операція перетворення багатовимірного тензора в одновимірний вектор, що передує щільним (Dense) шарам.

GPU (*Graphics Processing Unit*) — графічний процесор; використовується для паралельної обробки даних, особливо ефективний при навчанні нейронних мереж.

Input Shape — форма тензора, що подається на вхід нейронної мережі (висота, ширина, кількість каналів).

L2-регуляризація — метод боротьби з перенавчанням шляхом додавання штрафу за велику вагу параметрів у функцію втрат.

LeakyReLU — модифікація активаційної функції ReLU, яка дозволяє невелику похідну при негативному вхідному значенні, що зменшує ризик «мертвих нейронів».

MaxPooling — операція підвибірки, яка зменшує розмірність зображення, зберігаючи найбільше значення в кожному регіоні.

MD5 — алгоритм хешування, що дозволяє ідентифікувати дублікати зображень шляхом порівняння їхніх контрольних сум.

ROC-крива (*Receiver Operating Characteristic Curve*) — графік залежності чутливості (Recall) від 1 − специфічності для різних порогів класифікації.

Sigmoid — активаційна функція, що стискає значення до інтервалу (0, 1); використовується на останньому шарі моделі для бінарної класифікації.

SpatialDropout2D — різновид Dropout, що вимикає цілі канали ознак (feature maps), а не окремі нейрони, що особливо ефективно при роботі з зображеннями.

Streamlit — фреймворк Python для створення веб-додатків з інтерфейсом у браузері; використовується для запуску моделі та візуалізації результатів.

TensorFlow — бібліотека з відкритим кодом для реалізації та навчання моделей глибинного навчання.

Training / Validation / Test Set — відповідно: тренувальна, валідаційна та тестова вибірки, що використовуються для навчання, налаштування та перевірки мо

ВСТУП

Упродовж останніх десятиліть інформаційні технології дедалі активніше проникають у сферу медицини, забезпечуючи нові можливості для діагностики, лікування та моніторингу стану пацієнтів. Одним із найважливіших напрямів цифрової трансформації є автоматизований аналіз медичних зображень, який дозволяє зменшити вплив людського фактора, пришвидшити ухвалення клінічних рішень та покращити загальну якість медичного обслуговування. Особливої актуальності ця проблема набуває в контексті рентгенології — методу, що широко застосовується для виявлення патологій кісткової системи, зокрема переломів.

Незважаючи на простоту отримання рентгенівських знімків, їх інтерпретація залишається складним завданням, що вимагає високої кваліфікації лікаря-рентгенолога. У багатьох випадках, особливо в умовах дефіциту медичних кадрів або в екстрених ситуаціях, існує ризик пропущення критичних ознак ушкоджень. Тому виникає об’єктивна потреба у створенні інтелектуальних систем, які здатні автоматично аналізувати рентгенограми та виявляти ознаки переломів із прийнятною точністю та швидкістю.

Сучасні методи глибинного навчання, зокрема згорткові нейронні мережі (Convolutional Neural Networks, CNN), продемонстрували високу ефективність у задачах обробки зображень і вже знайшли своє застосування в медичних інформаційних системах. Завдяки здатності автоматично виділяти важливі ознаки на зображеннях без потреби в ручному втручанні, такі моделі можуть стати надійним інструментом у діагностичному процесі. Їх поєднання з веб-технологіями відкриває можливість створення доступних онлайн-інструментів, якими можуть користуватися лікарі, науковці або навіть пацієнти — у разі потреби швидкої самодіагностики чи другого погляду.

Метою цієї дипломної роботи є розробка програмного застосунку для виявлення переломів на рентгенівських знімках з використанням нейронних мереж. Такий застосунок має забезпечити користувачеві зручний інтерфейс для завантаження зображень, їх автоматичної обробки, класифікації та виведення

результатів, що дозволяє прискорити попередній етап діагностики та підвищити її достовірність.

Для досягнення поставленої мети було визначено такі основні завдання.

1. Провести аналіз існуючих методів та рішень для аналізу рентгенівських знімків засобами нейронних мереж.

2. Обрати набір даних, методи та програмні засоби для реалізації поставленої мети, а також обґрунтувати їх вибір.

3. Виконати аналіз та попередню обробку обраного набору даних.

4. Розробити кілька моделей нейронних мереж та виконати їх порівняльний аналіз для вибору найкращої архітектури.

5. Створити програмний застосунок для роботи з моделлю нейронної мережі, яка продемонструє найкращі показники ефективності.

Апробація результатів роботи на момент написання дипломної роботи не здійснювалась. Проте розроблене рішення має потенціал для практичного використання в навчальних, дослідницьких та медичних установах, де є потреба в автоматизованому аналізі рентгенівських знімків.

Структура дипломної роботи включає чотири розділи. У першому розділі розглянуто формулювання прикладної задачі, виконано огляд сучасних методів аналізу медичних зображень, а також проаналізовано програмні засоби, які можуть бути використані для її реалізації. Другий розділ присвячено опису алгоритмів і моделей нейронних мереж, які застосовувались у процесі розробки, з урахуванням особливостей їх архітектури, параметрів та методів навчання. У третьому розділі представлено програмну реалізацію системи: архітектуру, структуру компонентів, взаємодію модулів. У четвертому розділі наведено результати тестування, демонстрацію функціоналу системи, а також проведено експериментальне порівняння ефективності розроблених моделей з метою виявлення найоптимальнішого варіанта.

1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ МЕТОДІВ ОБРОБКИ МЕДИЧНИХ ЗНІМКІВ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ

Мета розділу — сформулювати прикладну задачу, що розв’язується в межах дипломної роботи, дослідити сучасні підходи до її вирішення та обґрунтувати вибір інструментальних засобів для реалізації програмної системи.

У підрозділі 1.1 представлено загальну характеристику задачі автоматичного виявлення переломів на рентгенівських знімках, описано її практичне значення, вхідні та вихідні дані, а також основні вимоги до результату. У підрозділі 1.2 здійснено огляд сучасних методів обробки медичних зображень із використанням нейронних мереж, розглянуто типові архітектури, сильні та слабкі сторони кожного підходу.

1.1 Постановка задачі

Однією з актуальних задач сучасної медичної інформатики є автоматизоване виявлення патологій на рентгенівських знімках. Своєчасна і точна діагностика переломів є критично важливою як у клінічній практиці, так і в екстреній медичній допомозі. Проте в реальних умовах, особливо за високого навантаження на лікарів-рентгенологів або за браку фахівців, можливі випадки діагностичних помилок, що можуть призводити до ускладнень. Тому виникає потреба у створенні інтелектуальної системи, здатної автоматично визначати наявність або відсутність перелому на рентгенівському зображенні.

У межах дипломної роботи розглядається задача бінарної класифікації рентгенівських знімків за допомогою моделей глибинного навчання. Кожне зображення, що надходить на вхід системи, має бути класифіковане як таке, що містить ознаки перелому, або як зображення без видимих ушкоджень. Вхідними

даними є цифрові знімки у форматах .png, .jpg або .jpeg, отримані з відкритих медичних наборів даних або клінічних архівів. Розмір вхідного зображення не є фіксованим — усі зображення піддаються попередній обробці, що включає нормалізацію та масштабування до стандартного формату, придатного для подання до нейронної мережі.

Метою обробки є отримання класифікаційного висновку у вигляді ймовірнісного розподілу між двома класами: “перелом” або “немає перелому”. На основі отриманих значень здійснюється інтерпретація результату. Система працює з одним зображенням за один сеанс обробки, що дозволяє сфокусувати увагу на точності класифікації та уникнути перевантаження користувача.

Таким чином, задача, що розв’язується в дипломній роботі, полягає у побудові системи, яка забезпечує повний цикл обробки рентгенівського знімка: від завантаження та попередньої обробки зображення до класифікації за допомогою нейронної мережі та відображення результату користувачеві у зручному форматі.

1.2 Сучасні методи виявлення переломів на рентгенівських знімках з використанням глибинного навчання

Сучасні дослідження [1], [2], [3] у сфері медичного аналізу зображень засвідчують стрімкий розвиток технологій штучного інтелекту, зокрема глибинного навчання, що дедалі ширше впроваджуються в системи підтримки клінічних рішень. Особливо значущими є успіхи в галузі автоматичного виявлення патологій на основі візуальних даних, де нейронні мережі, насамперед згорткові, демонструють високу ефективність у розпізнаванні складних просторових структур на зображеннях.

Одним із найактуальніших напрямів застосування таких методів є діагностика травм опорно-рухового апарату, зокрема — виявлення переломів кісток на рентгенівських знімках. Це завдання має високу практичну цінність і

потребує точних, надійних і швидких рішень, особливо в умовах обмежених людських ресурсів або великого навантаження на лікарів.

У сучасній науковій літературі представлений широкий спектр технічних підходів до цієї проблеми — від класичних згорткових архітектур, орієнтованих на бінарну класифікацію, до складніших моделей локалізації та сегментації, зокрема адаптованих алгоритмів об’єктного виявлення, гібридних багаторівневих рішень, а також систем зі зворотним зв’язком та підкріпленням.

З огляду на це, у цьому підрозділі проведено огляд чотирьох репрезентативних наукових публікацій, які висвітлюють поточний стан справ у галузі, демонструють досягнення, технічні підходи, метрики ефективності, а також наявні виклики та обмеження. Обрані роботи різняться за рівнем складності моделей, обсягом і типом даних, постановкою задачі та глибиною аналізу, що дозволяє сформувати більш цілісне уявлення про основні тенденції розвитку в цій сфері.

1.2.1 Модифікація алгоритму YOLOv8

У дослідженні [4] запропоновано модифіковану версію архітектури YOLOv8, орієнтовану на виявлення та локалізацію переломів передпліччя на рентгенівських знімках. Автори інтегрували легші блоки C2f-Ghost та C3-Ghost, що дозволило суттєво зменшити кількість параметрів моделі й підвищити швидкість обробки без критичних втрат у точності. Крім того, розроблено вдосконалену функцію втрат, яка враховує просторову якість виявлення та стабільність bounding boxes.

Модель навчалася на власноруч зібраному та анотованому датасеті з 1892 зображень і досягла точності 91.2 %, recall 89.7 % та середнього показника точності (mAP@0.5) 94.5 %. Це свідчить про високу придатність рішення до клінічного використання в системах підтримки прийняття рішень. Важливо, що запропонована система дозволяє візуально локалізувати його на зображенні.

Попри сильні сторони, підхід має обмеження. По-перше, він вимагає високоякісної анотації з координатами ушкоджень — що може бути дороговартісним у масштабі великих проєктів. По-друге, модель розрахована саме на об’єктне виявлення, тоді як у межах цієї дипломної роботи розв’язується задача класифікації знімка в цілому (тобто «є перелом» / «немає перелому»).

Узагальнюючи, дана робота репрезентує ефективне рішення для локалізаційних задач у реальному часі, однак її ідеї щодо спрощення архітектури та оптимізації функції втрат можуть бути адаптовані й для класифікаційного підходу.

1.2.2 Методи глибинного навчання

Стаття [5] є широким оглядом існуючих методів глибинного навчання для автоматичного виявлення переломів та інших скелетних ушкоджень. У ній систематизовано понад 50 досліджень, що охоплюють як класичні CNN-архітектури, так і сучасні підходи — ResNet, DenseNet, MobileNet, EfficientNet, трансформери, а також гібридні та багатоканальні моделі. Автори подають порівняння методів за ключовими метриками (точність, recall, AUC, F1), розглядають типи даних, структури моделей, підходи до аугментації, балансування вибірки та застосування transfer learning.

Цінність публікації полягає у її оглядовій глибині: аналізуються також методи пояснення результатів (XAI) — Grad-CAM, LIME, SHAP — що особливо важливо у медичному контексті. Автори звертають увагу на потребу інтерпретованості та перевірки моделей на незалежних клінічних даних, а також на часті методологічні похибки, що виникають при оцінюванні моделей лише на внутрішньому тестовому наборі. Окремо наголошується, що ефективність моделей повинна оцінюватися не лише за формальними метриками, а й за здатністю підтримувати клінічно релевантні рішення. Також у роботі підкреслено, що моделі,

які добре показують себе в лабораторних умовах, нерідко демонструють гіршу продуктивність у реальних сценаріях застосування.

Робота є корисним орієнтиром для розробників прикладних систем, оскільки надає класифікацію існуючих архітектур за сценаріями використання, дозволяє оцінити переваги та недоліки підходів.

1.2.3 Глибокі архітектури

У дослідженні [6] здійснено прикладну оцінку трьох популярних архітектур глибинного навчання: ResNet50, DenseNet121 та Inception-v3 — у контексті задачі виявлення переломів зап’ястя. У роботі використано датасет MURA, однак особливу увагу приділено перевірці моделей на незалежних вибірках з інших джерел, що моделює умови клінічного застосування.

Автори доводять, що моделі, які демонструють AUC > 0.95 на внутрішніх тестах, можуть значно втрачати точність на зовнішніх даних через різницю у форматі знімків, рівні шумів, типах пристроїв і способах розмітки. Особливо чутливою до таких змін виявилася Inception-v3, тоді як ResNet50 показала найкращу стабільність при помірній складності.

Ключова цінність цієї роботи — в акценті на перевірку узагальнюваності моделей та їх надійності в умовах змін навколишнього середовища. Саме цей аспект особливо важливий у медичному застосуванні, де моделі мають демонструвати стійку продуктивність за межами тренувального набору.

Результати дослідження підтверджують доцільність використання ResNet-подібних архітектур у розробці прототипів, орієнтованих на широкий спектр рентгенівських знімків різної якості — що безпосередньо корелює з метою дипломної роботи.

1.2.4 Класична згорткова нейронна мережа

У публікації [7] розглядається побудова базової архітектури згорткової нейронної мережі, орієнтованої на вирішення задачі бінарної класифікації рентгенівських зображень — зокрема, розпізнавання наявності або відсутності ознак перелому. Запропонована модель відзначається своєю структурною простотою та прямолінійністю: вона складається з трьох послідовних згорткових блоків, кожен із яких включає згортку, підвибірку (pooling), а також подальше згладжування (flattening) та підключення до повнозв'язаних шарів. У якості функції активації використовується ReLU, як функція втрат обрана categorical cross-entropy, а оптимізація здійснюється за допомогою алгоритму Adam.

Навчання моделі проводилось на власноруч сформованому наборі даних, що складався з 1400 зображень, і дало змогу досягти точності понад 87 % на тестовій вибірці. Автори наголошують, що, попри невеликий розмір датасету та відсутність складних архітектурних рішень, отримані результати є цілком задовільними. Особливо актуальним даний підхід може бути в умовах обмежених обчислювальних ресурсів або в освітніх цілях, де важлива не стільки абсолютна ефективність, скільки наочність і швидкість реалізації.

Водночас, критичний аналіз роботи засвідчує, що архітектура не включає низку базових для сучасного глибинного навчання компонентів, таких як нормалізація шарів (BatchNormalization), регуляризація (Dropout або L2), стратифіковане тестування, перевірка на узагальнюваність чи детальний розбір типових помилок класифікації. Через це ефективність моделі може бути переоціненою у разі застосування до ширших або складніших вибірок. Утім, попри зазначені обмеження, ця модель може служити стартовою точкою для побудови складніших і більш стабільних рішень, у тому числі в контексті дипломної роботи. Подальше нарощування архітектури, використання більш глибоких блоків, додавання сучасних прийомів регуляризації та моніторингу навчання може суттєво підвищити як точність, так і узагальнювальну здатність системи.

2 ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ТА СЕРЕДОВИЩЕ РОЗРОБКИ

У цьому розділі розглянуто інструментальні засоби, які можуть бути використані для створення програмних систем, заснованих на нейронних мережах. Увагу зосереджено як на виборі мови програмування, так і на відповідних бібліотеках для побудови моделей глибинного навчання та обробки вхідних даних.

Спочатку буде проведено огляд найпопулярніших мов програмування, які використовуються у сфері машинного та глибинного навчання, з подальшим обґрунтуванням вибору мови Python [8] як основної для реалізації цієї дипломної роботи. Далі буде детально проаналізовано бібліотеки, що забезпечують створення, навчання та валідацію згорткових нейронних мереж, із порівнянням основних альтернатив. Третій підрозділ буде присвячено програмним засобам для попередньої обробки, аналізу та підготовки даних, що є обов'язковим етапом перед подачею інформації до моделі.

2.1 Мова програмування

Розробка сучасного програмного забезпечення на основі методів глибинного навчання потребує ретельного вибору мови програмування, яка забезпечить не лише необхідну обчислювальну продуктивність, а й гнучкість, розвинену екосистему бібліотек, сумісність із сучасними фреймворками машинного навчання та зручність у підтримці коду. Особливої ваги цей вибір набуває при роботі з задачами обробки зображень, де ефективність реалізації безпосередньо впливає на швидкість навчання моделей та можливість інтеграції з візуальними інтерфейсами.

Серед найбільш популярних мов, які використовуються у сфері штучного інтелекту та машинного навчання, варто виділити Python, Julia та R. Кожна з них має свої сильні та слабкі сторони, і саме порівняння їхніх можливостей дозволяє

обґрунтовано зробити вибір на користь тієї мови, яка найкраще відповідає вимогам конкретного проєкту.

2.1.1 Мова програмування Julia

Мова програмування Julia була створена спеціально для задач наукових обчислень, де поєднуються вимоги до високої швидкості, що наближається до C/C++, і простоти синтаксису, характерної для динамічних мов. Julia демонструє вражаючу продуктивність завдяки компіляції з використанням LLVM і векторизованій обробці даних. Зокрема, вона добре підходить для розв’язання диференціальних рівнянь, статистичних обчислень, обробки великих обсягів даних у реальному часі.

У сфері глибинного навчання Julia має кілька бібліотек, серед яких Flux.jl і Knet.jl є найвідомішими. Ці інструменти дозволяють реалізовувати нейронні мережі з достатньо високим рівнем гнучкості. Проте спільнота Julia ще не досягла такого рівня розвитку, як у Python, а кількість прикладів, документації та інтегрованих фреймворків для зручної роботи з GPU, обробки зображень та побудови інтерфейсів є обмеженою. Більшість навчальних матеріалів, наукових статей і відкритих реалізацій глибинних моделей сьогодні базуються саме на Python, що ускладнює швидку адаптацію Julia в таких прикладних проектах, як класифікація рентгенівських знімків.

2.1.2 Мова програмування R

Мова R історично розвивалась як інструмент для статистичного аналізу, візуалізації та побудови математичних моделей. Вона має глибоку інтеграцію з академічними інструментами, підтримку багатьох статистичних методів, вбудовані

засоби побудови графіків (ggplot2, lattice) та високоякісну візуалізацію результатів аналізу. R активно використовується в соціології, економетриці, біостатистиці.

У контексті машинного навчання R також має відповідні бібліотеки: caret, h2o, mlr, а для глибинного навчання — обгортки на кшталт kerasR або підтримку TensorFlow через tensorflow package. Однак такі обгортки є переважно інтерфейсами до Python-фреймворків, і фактично сам код все одно виконується у середовищі Python. Крім того, робота з зображеннями в R є значно складнішою і менш продуктивною: кількість бібліотек для обробки зображень дуже обмежена, а глибока інтеграція з GPU відсутня.

Таким чином, попри ефективність R у статистичних дослідженнях, її застосування для розробки повноцінних систем глибинного навчання обмежене, особливо в задачах комп'ютерного зору, де критично важлива робота з тензорами, нормалізація піксельних значень, попередня обробка форматів зображень та подальша інтеграція з візуальними веб-інтерфейсами.

2.1.3 Мова програмування Python

Python — беззаперечний лідер у сфері машинного та глибинного навчання. Він поєднує простий синтаксис, розвинену інфраструктуру, багатство бібліотек, активну спільноту та підтримку з боку провідних дослідницьких центрів і технологічних компаній. Завдяки відкритому коду, регулярному оновленню інструментів і величезній кількості навчальних матеріалів Python став де-факто стандартом для реалізації моделей штучного інтелекту.

Для глибинного навчання в Python доступні потужні фреймворки — TensorFlow [9] , Keras [10] , PyTorch, які дозволяють реалізовувати як найпростіші, так і промислові моделі зі збереженням усіх ваг, архітектур, історії навчання. Обробку числових масивів забезпечують NumPy, SciPy, Xarray, а візуалізацію — Matplotlib [11] , Seaborn, Plotly. Для роботи з табличними структурами існує Pandas, а для обробки зображень — OpenCV, scikit-image. При цьому всі ці бібліотеки легко

поєднуються в одному проєкті, забезпечуючи гнучке і зручне середовище розробки.

Python також підтримує розробку веб-інтерфейсів та інтеграцію моделей у застосунки. Зокрема, Streamlit [12] дозволяє створити повноцінний інтерфейс для запуску моделі, виведення результату класифікації, завантаження зображень — усе це без використання JavaScript чи HTML.

У межах цієї дипломної роботи Python було обрано з кількох причин. По-перше, він дозволив реалізувати всі етапи розробки: від попередньої обробки зображень, масштабування і нормалізації, до побудови моделі CNN, її тренування, тестування та розгортання у веб-застосунку. По-друге, Python має готові інструменти для візуалізації результатів, побудови ROC-кривих, оцінки метрик ефективності та збереження моделей. По-третє, завдяки величезній спільноті, більшість технічних питань мають докладні відповіді в офіційній документації або на форумах.

Порівняльний аналіз трьох мов засвідчує, що саме Python надає найширші можливості для практичного застосування глибинного навчання, особливо в таких прикладних задачах, як виявлення переломів на рентгенівських знімках. Простота реалізації, підтримка всіх сучасних фреймворків, гнучкість архітектур, активне середовище розробників — усе це забезпечує високу швидкість розробки та надійність роботи моделей у продакшн-середовищах.

Таким чином, Python є не лише технічно доцільним вибором, але й найефективнішим з точки зору результатів, що були досягнуті в межах цієї дипломної роботи.

2.2 Бібліотеки для роботи з нейронними мережами

Однією з ключових причин активного використання мови Python у галузі глибинного навчання є її багата, структурована та постійно розвивана екосистема бібліотек, яка охоплює весь цикл створення, вивчення, аналізу та впровадження

моделей штучного інтелекту. Python дозволяє розробляти як дослідницькі прототипи, так і промислові системи, забезпечуючи гнучке налаштування моделей, візуалізацію результатів і розгортання у вигляді інтерфейсів або сервісів. У межах розробки програмної системи для виявлення переломів на рентгенівських знімках було обрано надійний, поширений і перевірений набір бібліотек: TensorFlow, Keras, NumPy, Matplotlib, Seaborn і scikit-learn. Вибір цих інструментів продиктований не лише їх технічними можливостями, але й підтримкою великої спільноти, наявністю детальної документації, активною історією оновлень та сумісністю з іншими елементами програмної архітектури.

2.2.1 Бібліотека TensorFlow

Центральним елементом програмної реалізації виступає фреймворк TensorFlow — один із наймасштабніших та найпоширеніших засобів для глибинного навчання, розроблений Google і вперше представлений у 2015 році. TensorFlow надає повноцінну підтримку роботи з тензорами, побудови обчислювальних графів, автоматичного диференціювання, навчання моделей на GPU, а також розгортання у вигляді окремих сервісів або мобільних додатків. У дипломному проєкті застосовано версію TensorFlow 2.19, яка реалізує імперативний стиль програмування, зручні засоби керування процесом навчання, механізми callback-функцій, регуляризації, динамічного збереження моделей та адаптивного керування швидкістю навчання. Завдяки цьому було забезпечено повну контрольованість та відтворюваність усіх експериментів, що критично важливо в умовах наукової роботи.

Особливої уваги заслуговує можливість безшовної інтеграції TensorFlow із системами промислового рівня. Наприклад, для сервісифікації моделі або інтеграції в мобільні платформи можуть використовуватись компоненти TensorFlow Serving, TensorFlow.js або TensorFlow Lite. У порівнянні з іншими фреймворками, такими як PyTorch чи MXNet, TensorFlow надає ширші можливості

для практичного розгортання моделей, хоча PyTorch часто вважається зручнішим для побудови дослідницьких прототипів завдяки гнучкому динамічному графу обчислень.

2.2.2 API Keras

Розробка самої архітектури моделі здійснювалася за допомогою API Keras, який є офіційною частиною TensorFlow. Це високорівневий інтерфейс, що дозволяє описувати структуру мережі як послідовність шарів у декларативному стилі, без необхідності занурення в низькорівневу реалізацію обчислень. Усі чотири архітектури, розроблені в рамках цієї роботи (Custom CNN v1—v4), були реалізовані за допомогою функціонального або послідовного API Keras. До складу використовуваних елементів увійшли згорткові шари, шари нормалізації, пулінгу, активації, регуляризації (Dropout, L2), а також глобальні шари згортання та повнозв'язні блоки. Keras надає потужні механізми для компіляції моделей, гнучкого вибору функцій втрат, метрик ефективності та оптимізаторів, що значно спростило процес експериментування з конфігураціями та параметрами навчання.

Важливою перевагою використання Keras є її простота: зменшення обсягу коду на порядок у порівнянні з низькорівневими реалізаціями, відсутність потреби в ручному керуванні градієнтами та оновленням ваг, а також зрозумілий механізм навчання за одну команду. На відміну від fastai, яка орієнтована на PyTorch, або від Chainer, що втратив актуальність, Keras лишається основним стандартом для швидкого прототипування та навчання моделей у середовищі TensorFlow.

2.2.3 Бібліотека NumPy

Для обробки вхідних зображень та формування структури вхідного тензора використовувалась бібліотека NumPy — фундаментальний інструмент у всьому

стеку Python для чисельних обчислень. Саме через NumPy відбувалось масштабування зображень до необхідного розміру, нормалізація значень пікселів, приведення типів даних, а також формування тензора відповідного розміру з розширенням осей для батчування та передачі в модель. Завдяки NumPy вдалося реалізувати обробку та подачу даних у пам’яті без збереження тимчасових файлів, що значно покращило швидкість тренувального циклу.

Альтернативи, такі як CuPy або JAX, у цьому контексті не використовувались через вищу складність інтеграції та відсутність необхідності в GPU-прискоренні на етапі препроцесингу. Таким чином, NumPy виступила як універсальна обчислювальна основа всієї системи.

2.2.4 Бібліотеки Matplotlib та Seaborn

У рамках аналізу результатів роботи моделі велика увага приділялась візуалізації. Для цього використовувались дві взаємодоповнюючі бібліотеки — Matplotlib та Seaborn. Matplotlib дозволяє будувати базові графіки функції втрат, точності, зміни AUC у динаміці та зображення передбачень. Seaborn, у свою чергу, забезпечує ширші можливості для побудови статистичних візуалізацій, які дозволяють виявити закономірності в даних і провести якісний аналіз розподілу передбачень.

Зокрема, у межах роботи побудовано матриці неточностей, графіки порогових значень, гістограми передбачень для обох класів, а також ROC-криві для оцінки дискримінаційної здатності моделі. Завдяки інтеграції цих бібліотек з Jupyter Notebook вдалося оперативно виводити результати навчання безпосередньо під час кожної ітерації моделювання, що сприяло гнучкому налаштуванню архітектур.

2.2.5 Бібліотека Scikit-learn

Оцінка ефективності моделі, а також розрахунок метрик класифікації проводились із застосуванням бібліотеки scikit-learn [13]. Її інструментарій дозволив отримати не лише стандартні показники, як-от точність, повнота, F1-міра та AUC, але й деталізовані метрики для кожного класу, побудову стратифікованих вибірок, крос-валідацію, формування звітів у табличному форматі.

Scikit-learn був зручним і для тестування моделей за межами глибоких мереж, що вказує на його універсальність та придатність для побудови гібридних систем. Попри наявність альтернативних інструментів, таких як XGBoost або LightGBM, які спеціалізуються на ансамблевих методах, саме scikit-learn виявився найбільш доречним у контексті контролю якості CNN-моделей.

2.3 Бібліотеки для обробки даних

Одним із ключових етапів розробки системи глибинного навчання є забезпечення належної якості та структури вхідних даних. У випадку задач класифікації медичних зображень, зокрема рентгенівських знімків, які можуть істотно відрізнятись за роздільністю, форматом, контрастністю та джерелом походження, роль попередньої обробки даних набуває критичного значення.

Саме якісна підготовка вибірки — нормалізація, стандартизація, приведення до єдиного розміру та структури — дозволяє нейронній мережі навчатися ефективно, без втрат продуктивності або перекосів у розподілах класів. У межах даного дипломного проєкту було застосовано три ключові бібліотеки для обробки даних: OpenCV, NumPy та Pandas. Вони забезпечили повний цикл підготовки, починаючи від завантаження зображень і завершуючи підготовкою табличних структур з анотаціями.

Особливу роль у цьому процесі відіграла бібліотека OpenCV (Open Source Computer Vision Library). Це один із найстаріших та найпотужніших інструментів

для обробки зображень, який постійно оновлюється й підтримується відкритою спільнотою.

OpenCV підтримує понад 2500 функцій для аналізу зображень та відео, що охоплюють зчитування, перетворення кольору, фільтрацію, виявлення контурів, геометричні трансформації, детекцію об’єктів, тощо. Завдяки високій продуктивності й сумісності з бібліотеками на кшталт NumPy та TensorFlow, вона є стандартом де-факто в системах комп’ютерного зору.

У межах реалізованої системи OpenCV застосовувалась для обробки кожного вхідного зображення незалежно від його початкового розміру, формату або співвідношення сторін. Усі знімки були автоматично зчитані, масштабовані до розміру 384×384 пікселі та перетворені у відтінки сірого, що дозволило зменшити кількість параметрів моделі та уникнути надлишковості кольорової інформації, яка не є суттєвою у задачі виявлення переломів.

Зображення також нормалізувалися до діапазону [0, 1], що забезпечувало стабільність навчання, зниження ризику вибуху або занепаду градієнтів та покращену швидкість збіжності.

Особливо цінною була здатність OpenCV обробляти зображення без необхідності створення тимчасових файлів. Усі дії — зчитування, зміна розміру, нормалізація — виконувались у пам’яті, що мінімізувало час обробки та потребу в дисковому введенні/виведенні. Це дозволило інтегрувати OpenCV як частину єдиного пайплайну обробки, що включав зчитування з диску, перетворення, підготовку до тензора та подачу в модель.

У поєднанні з NumPy OpenCV створює надзвичайно ефективну обчислювальну основу, де масиви зображень легко перетворюються на тензори без втрати продуктивності.

У порівнянні з іншими бібліотеками, такими як PIL або torchvision, OpenCV демонструє вищу швидкість обробки, більшу кількість функцій і ширшу підтримку форматів.

Наприклад, PIL часто потребує додаткових кроків для нормалізації зображень і має обмеження щодо високопродуктивної обробки, а torchvision тісно

інтегрована з PyTorch, що ускладнює її використання в середовищі TensorFlow. OpenCV натомість є універсальним інструментом, сумісним як із низькорівневими обчисленнями, так і з високорівневими фреймворками глибинного навчання.

Другим базовим елементом системи стала бібліотека NumPy, яка забезпечує представлення зображень у вигляді багатовимірних масивів та виконання всіх необхідних операцій над ними. З її допомогою реалізовано базову логіку обробки даних: перетворення типів, масштабування, додавання осей, батчування, формування тензорів розміру (batch_size, height, width, channels). NumPy також використовувалась у функціях формування міні-батчів, підрахунку розмірностей, агрегування зображень за класами тощо. Таким чином, вона виступила універсальним посередником між вхідним зображенням і моделлю.

Третьою складовою, без якої неможливий аналіз супровідних даних, є Pandas — бібліотека для роботи з табличними структурами даних, що активно використовується в аналізі, підготовці та перетворенні структурованої інформації.

У межах цієї дипломної роботи Pandas відігравала надзвичайно важливу роль на етапі зчитування метаінформації з анотацій (розміток класів), вивчення розподілу класів між наборами, аналізу статистичних характеристик, а також формування підвибірок даних для тренувального, валідаційного та тестового наборів.

Саме завдяки Pandas вдалося реалізувати стратифікований поділ даних, що є критичним для задач бінарної класифікації зі змінним балансом класів. Також було виконано контроль коректності анотацій, виявлення дубльованих записів та невідповідностей між іменами файлів і наявністю зображень на диску.

У рамках побудови графіків і таблиць Pandas дозволила не лише працювати з CSV-файлами, але й виконувати групування за класами, вираховувати кількість прикладів у кожній групі, зливати різні таблиці на основі спільних ключів. Це забезпечило точний контроль над процесом побудови вибірки, унеможливило перекіс у навчанні та надало повну прозорість у структурі даних.

Однією з особливих переваг Pandas є її глибока інтеграція з іншими бібліотеками Python, зокрема Matplotlib, Seaborn, NumPy та scikit-learn. Завдяки

цьому будь-яку таблицю можна легко візуалізувати, трансформувати або подати як вхід до алгоритму машинного навчання.

Порівняно з масштабованими рішеннями, такими як Dask або PySpark, Pandas має перевагу в простоті, швидкості розробки та зручності локального налагодження. Вона не потребує складного середовища чи обчислювального кластера, тому є ідеальною для дослідницьких або локальних задач.

У випадку даного дипломного проєкту, де обсяг даних був достатньо великим для моделі, але не виходив за межі стандартної оперативної пам’яті, саме Pandas забезпечила найкращий баланс між функціональністю, продуктивністю та простотою.

Таким чином, поєднання OpenCV, NumPy та Pandas дозволило створити повноцінний, ефективний та відтворюваний механізм обробки даних, що охоплює як візуальний компонент, так і структуровану метаінформацію.

Кожна з бібліотек доповнювала іншу, утворюючи єдиний технологічний ланцюг підготовки зображень і супровідних анотацій. Завдяки цьому стало можливим ефективне навчання згорткової нейронної мережі, стабільне тестування та коректна інтерпретація результатів. У поєднанні з обраними фреймворками глибинного навчання цей стек забезпечив повну функціональність на всіх етапах — від даних до передбачення.

У результаті аналізу та обґрунтування технологічних засобів, використаних у дипломному проєкті, було сформовано раціональний і взаємопов’язаний стек інструментів для розробки системи глибинного навчання.

Було розглянуто найпоширеніші мови програмування, серед яких Python виявився оптимальним вибором завдяки своїй універсальності, гнучкості, простоті синтаксису та багатій екосистемі бібліотек, спеціалізованих на задачах машинного та глибинного навчання.

У рамках порівняльного аналізу бібліотек, призначених для побудови нейронних мереж, обґрунтовано доцільність застосування TensorFlow і Keras як стабільної основи для моделювання архітектур, а також використання NumPy, Matplotlib, Seaborn і scikit-learn як супутніх інструментів для обробки, візуалізації

та аналізу результатів. Ці бібліотеки забезпечили необхідну функціональність для побудови, навчання, оцінювання та вдосконалення згорткових нейронних мереж.

Також було показано, що якісна попередня обробка даних є критичною умовою для досягнення високої точності класифікації. З цією метою в системі були застосовані бібліотеки OpenCV, NumPy та Pandas, які дозволили реалізувати повноцінний та ефективний механізм підготовки як візуального матеріалу, так і супровідних структурованих анотацій. Кожен із інструментів відіграв специфічну роль у загальній архітектурі, створюючи узгоджене середовище для розробки, тестування й оптимізації моделі.

Загалом, сформований стек технологій не лише дозволив реалізувати поставлену задачу, а й забезпечив гнучкість, розширюваність і масштабованість розробленого рішення. У наступному розділі буде детально розглянуто архітектуру нейронних мереж, особливості їх навчання, порівняння версій та аргументи на користь обраної моделі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ АНАЛІЗУ РЕНТГЕНІВСЬКИХ ЗНІМКІВ

У цьому розділі представлено практичну реалізацію розробленої системи для автоматичного виявлення переломів на рентгенівських знімках. Робота над програмним забезпеченням охоплювала декілька ключових етапів: підготовку та дослідження набору даних, побудову й навчання моделей глибинного навчання, а також створення інтерактивного веб-додатку для зручної взаємодії користувача з системою.

Перший підрозділ присвячено опису використаного датасету, його структури, джерела походження, особливостей форматування та попередньої обробки. У другому підрозділі розглянуто побудову декількох варіантів моделей згорткових нейронних мереж, процес їх навчання та порівняння результатів на основі основних метрик класифікації. Третій підрозділ описує архітектуру веб-додатку, реалізованого з використанням фреймворку Streamlit, логіку його роботи та функціональні можливості, які надаються користувачеві.

3.1 Набір даних

Якість та репрезентативність навчального набору даних відіграє ключову роль у процесі побудови систем, що базуються на методах глибинного навчання, особливо в контексті медичної діагностики. Класифікація рентгенівських знімків — це задача, яка потребує не лише великої кількості зображень, а й глибокого розуміння їхнього походження, структурної варіативності, а також якості міток. Враховуючи цю обставину, під час реалізації даної дипломної роботи було прийнято обґрунтоване рішення не обмежуватися одним джерелом даних, а об’єднати два незалежні датасети, що мають відкриту ліцензію, відповідну тематику та підтримують однакову схему класифікації:

Bone_Fracture_Binary_Classification [14] і BoneFractureDataset [15]. Об'єднання таких наборів дозволяє значно підвищити узагальнюючу здатність майбутньої моделі, оскільки вона навчається на різноманітніших прикладах з різних клінічних середовищ.

Перший з обраних датасетів, Bone_Fracture_Binary_Classification, містив 10 595 зображень, з яких 5400 були класифіковані як "без перелому", а 5195 — як "перелом" (рисунок. 3.1). Ці знімки мали однорідний формат (переважно .jpg та .png) і, згідно з документацією, представляли зображення різних відділів опорно-рухового апарату. Структура набору була сформована відповідно до задачі бінарної класифікації, де мітка класу була явно задана для кожного зображення.

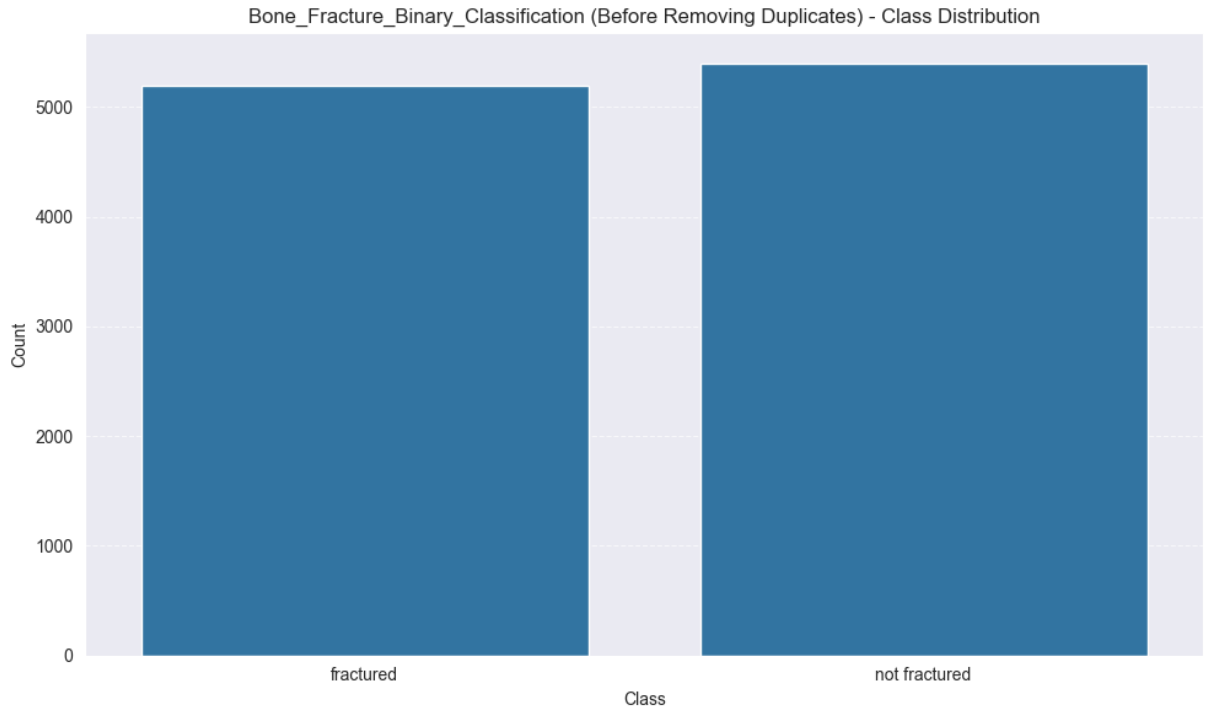

Рисунок 3.1 — Початковий розподіл класів у датасеті Bone_Fracture_Binary_Classification

На перший погляд набір виглядав збалансованим: кількість позитивних і негативних прикладів майже співпадала. Проте для більш надійної оцінки структури вибірки була здійснена перевірка її наявної унікальності. Оскільки дублікати зображень є однією з найбільш поширених проблем у медичних даних

— особливо коли збірка виконується з різних джерел або в результаті неправильного копіювання — було прийнято рішення здійснити детерміновану перевірку на дублікати за допомогою MD5-хешування [16]. Цей метод передбачає побудову геш-кодової сигнатури кожного зображення на основі його бітового представлення, що дозволяє виявити не лише явні дублікати (однакові назви), а й повністю ідентичні зображення з різними іменами.

У результаті аналізу було виявлено 6709 дублікатів, що становило понад 63 % від усього обсягу початкового датасету. Це означало, що більш ніж половина зображень дублювала інші приклади. Видалення цих повторів було критично необхідним для збереження надійності моделі, оскільки наявність дублікатів могла призвести до перенавчання, коли модель «запам’ятовує» приклади замість того, щоб навчитися узагальнювати закономірності. До і після видалення дублікатів співвідношення обсягів наведено нижче (рисунок 3.2).

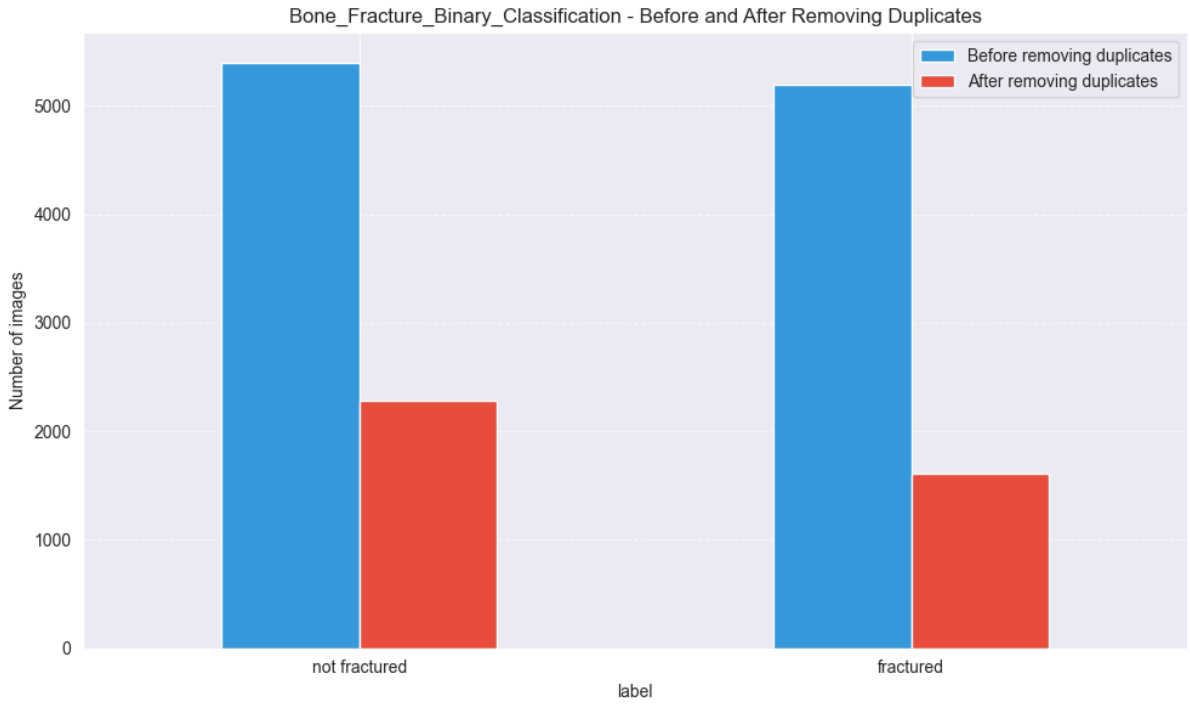

Рисунок. 3.2 — Порівняння розміру датасету Bone_Fracture_Binary_Classification до і після очищення

Після видалення дублікатів у наборі залишилося 3886 унікальних зображень, із яких 2282 належали до класу “без перелому”, а 1604 — до класу “перелом” Хоча обсяг суттєво зменшився, однак залишкова вибірка зберегла інформаційну цінність і класовий баланс, прийнятний для подальшого навчання моделі без спеціальних компенсаторних заходів. Оновлений розподіл класів представлено на (рисунок 3.3).

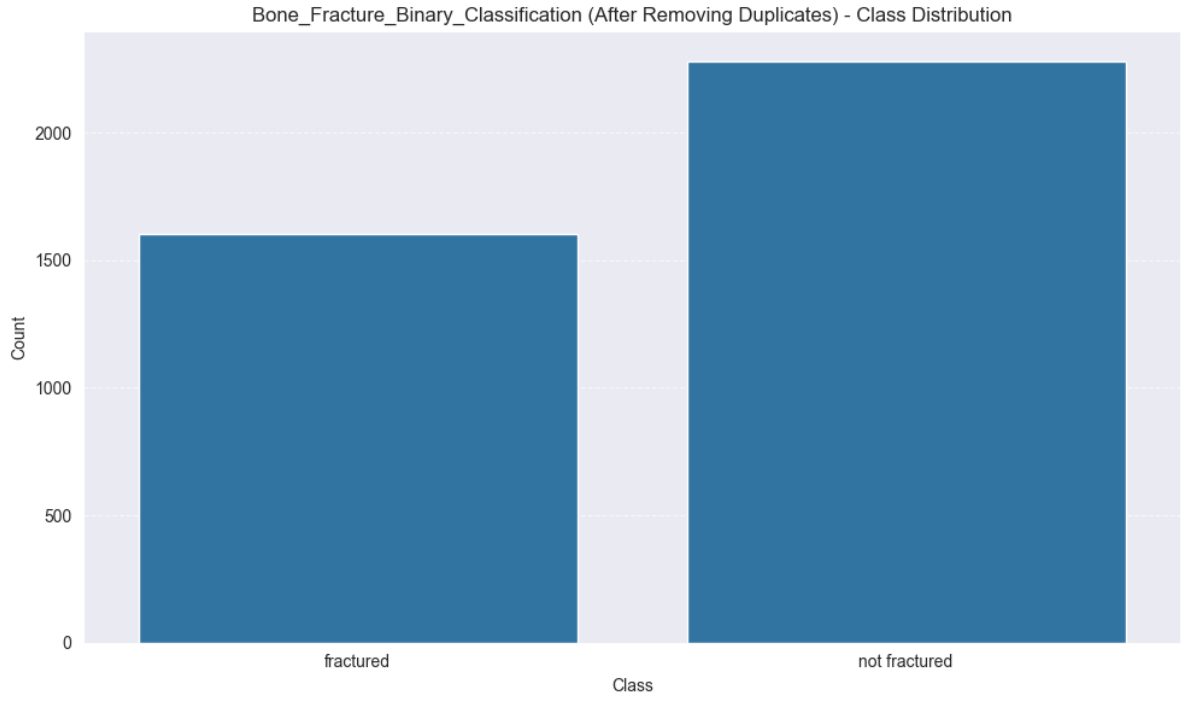

Рисунок 3.3 — Розподіл класів у першому датасеті після видалення дублікатів

Другим джерелом даних став датасет BoneFractureDataset, який на етапі завантаження містив 9463 зображення, розподілені майже порівну між класами: 4840 знімків були мічені як *з* переломами, а 4623 — як без переломів (рисунок 3.4). Зображення, включені до цього набору, також охоплюють різні анатомічні ділянки та містять певну варіативність у якості, форматі й типі знімків, що робить його цінним джерелом для побудови узагальнюючої класифікаційної системи. Окрім того, датасет супроводжується чіткою структурою каталогів та доступною документацією, що значно полегшило етапи завантаження, попередньої обробки та подальшої інтеграції до об’єднаної вибірки.

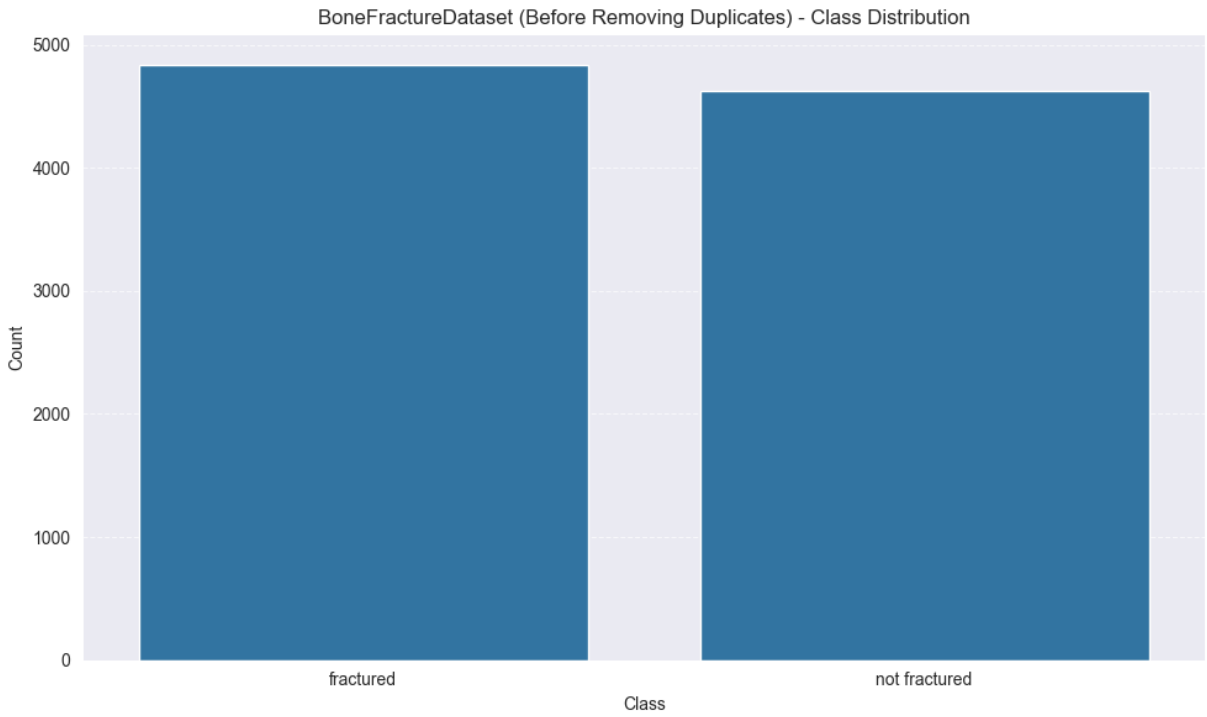

Рисунок 3.4 — Розподіл класів у BoneFractureDataset до очищення

Цей набір виявився якіснішим у плані структурної цілісності: лише 123 файли були визнані дублікативними за результатами перевірки на основі MD5-хешування. Це вказує на ретельнішу підготовку даних у процесі формування набору — імовірно, було застосовано контроль унікальності ще на етапі первинного збирання зображень або під час ручної верифікації перед публікацією. Така відносно мала кількість повторів, менш ніж 1.3 % від загального обсягу, вигідно вирізняє цей датасет у порівнянні з першим, де дублікати становили понад 60 %.

Після очищення залишилося 9339 унікальних зображень (рисунок 3.5), із яких 4733 належали до класу “перелом”, а 4606 — до класу “без перелому”. Важливо підкреслити, що навіть після видалення дублікатів загальне співвідношення між класами збереглося майже ідеально збалансованим.

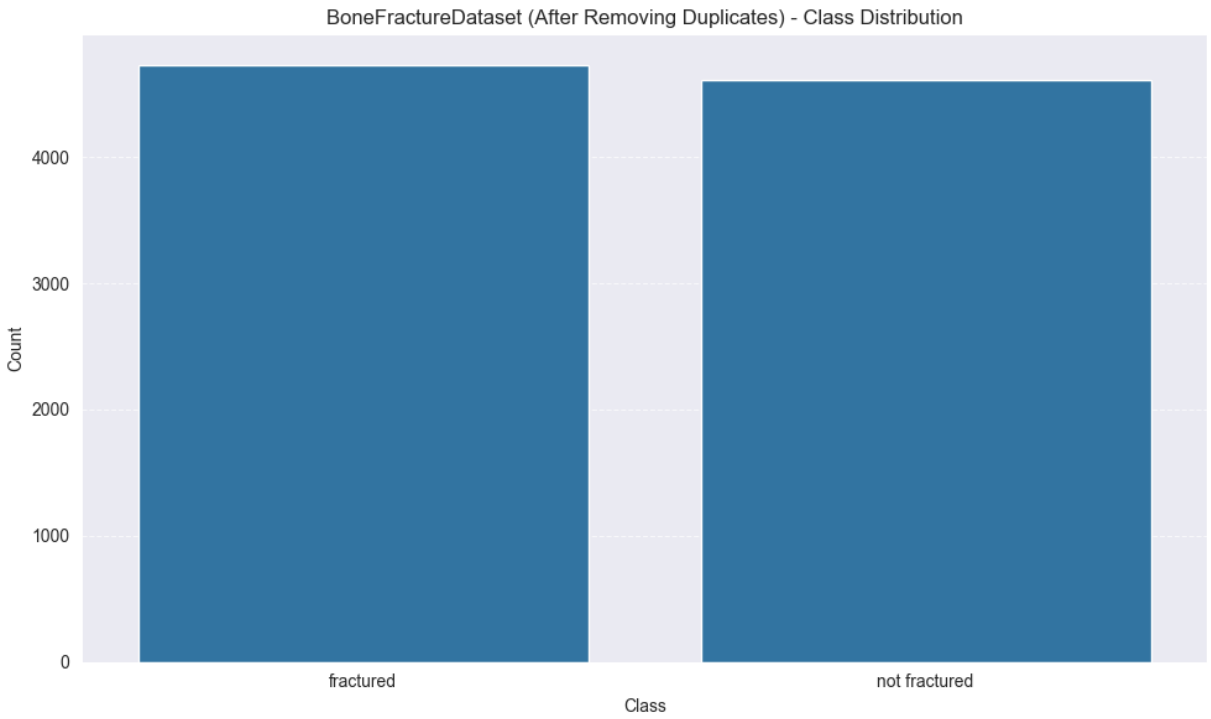

Рисунок 3.5 — Розподіл класів у BoneFractureDataset після видалення дублікатів

На наступному етапі два очищені датасети було об’єднано. Це дозволило не лише збільшити кількість прикладів, а й забезпечити більшу різноманітність стилів знімкування, анатомічних зон, маркувань, а також джерел походження. Така варіативність позитивно впливає на здатність моделі до узагальнення та підвищує її стійкість до шуму та зміни умов знімкування. Об’єднана вибірка після остаточної перевірки (у тому числі на міждатасетні дублікати) містила 9861 унікальне зображення, з яких 4674 відповідали класу “без перелому”, а 5187 — “перелом” (рисунок 3.6). Незначний дисбаланс між класами залишився допустимим для ефективного навчання моделі без додаткових заходів балансування. Об'єднаний набір став основною основою для побудови й тестування моделей, а також гарантував кращу адаптованість системи до реальних, неідеалізованих медичних умов. Його остаточна структура та обсяг надали достатньо статистичної потужності для стабільного навчання згорткової нейронної мережі.

Рисунок 3.6 — Підсумковий розподіл класів у фінальному об’єднаному наборі даних

Рентгенівські знімки у фінальному датасеті були різні за розміром, контрастністю та інформативністю. Частина з них містила допоміжні маркування, як-от літери “L” та “R”, службові поля, іноді — артефакти від цифрової обробки або компресії. Це дозволяє моделі навчатися на зображеннях, які наближені до реальних клінічних умов і включають типові для лікарської практики варіанти оформлення.

Після остаточного очищення було виконано розділення даних на підвибірки (рисунок 3.7). Використано класичну схему пропорційного поділу: 70% зображень відійшло до тренувального набору, 15% — до валідаційного, і ще 15% — до тестового. Всі вибірки формувалися із дотриманням принципу стратифікації, тобто з урахуванням збереження початкового співвідношення класів. Такий підхід дозволяє мінімізувати ризик викривлення результатів під час оцінювання, забезпечуючи справедливу перевірку якості узагальнення моделі.

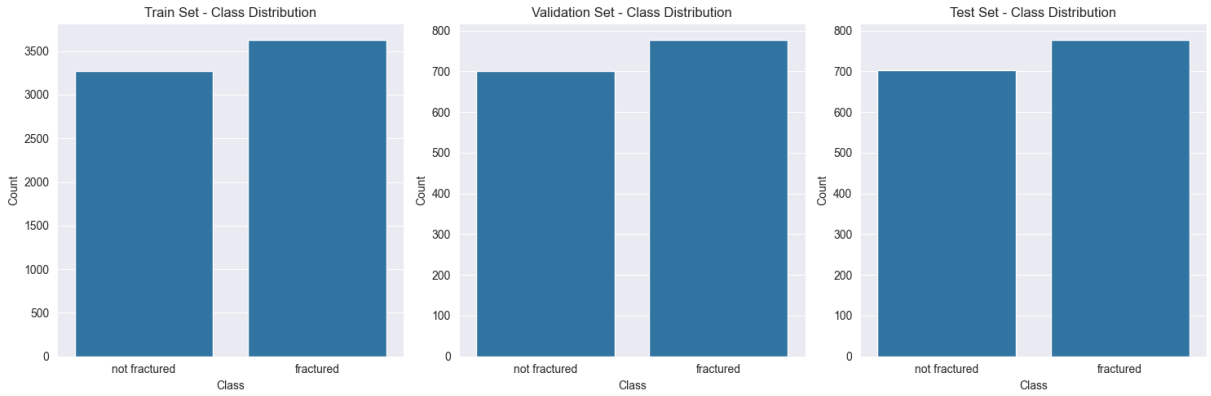

Рисунок 3.7 — Розподіл набору даних на вибірки

У підсумку було сформовано чистий, структурований та збалансований датасет, який не лише забезпечує належні умови для навчання згорткової нейронної мережі, але й імітує різноманітність та неоднорідність реальних клінічних даних. Якісна підготовка даних стала фундаментом для наступних етапів розробки системи — побудови, навчання, валідації моделі та створення її користувацького застосування.

3.2 Моделі та їх навчання

У цьому підрозділі розглянуто архітектури згорткових нейронних мереж, розроблені в межах даної дипломної роботи, а також процес їх навчання, результати оцінювання та аналіз ефективності. Кожна модель представлена окремо, з описом її внутрішньої структури, динаміки метрик під час навчання, ключових показників на тестовій вибірці, а також графічною інтерпретацією результатів, зокрема у вигляді ROC-кривої та матриці неточностей. Такий підхід дозволяє не лише порівняти моделі між собою, а й виявити сильні та слабкі сторони кожної з них в контексті поставленої задачі класифікації рентгенівських знімків.

3.2.1 Модель Custom CNN v1

Перший варіант моделі, умовно позначений як Custom CNN v1, був створений із метою побудови простої, проте функціонально спроможної архітектури згорткової нейронної мережі для бінарної класифікації рентгенівських знімків. Основним завданням цієї версії було сформувати базову архітектурну структуру, від якої надалі можна буде відштовхуватись у процесі поступової оптимізації та поглиблення.

Архітектура Custom CNN v1 (рисунок 3.8) включає три згорткові блоки з фільтрами у 32, 64 та 128 каналів відповідно. Кожен блок має наступну структуру: згортка (Conv2D) з активацією ReLU, шар пакетної нормалізації (BatchNormalization [17]), підвибірка (MaxPooling2D) та шар Dropout із коефіцієнтом 0.25. На завершення — Flatten, повнозв'язний шар із 128 нейронів, ще один BatchNormalization, Dropout (0.5), і вихідний шар із sigmoid-активацією. Це типова, але стійка архітектура для задач бінарної класифікації з середнім обсягом зображень. Її перевага полягає в простоті, швидкому навчанню та легкості в інтерпретації.

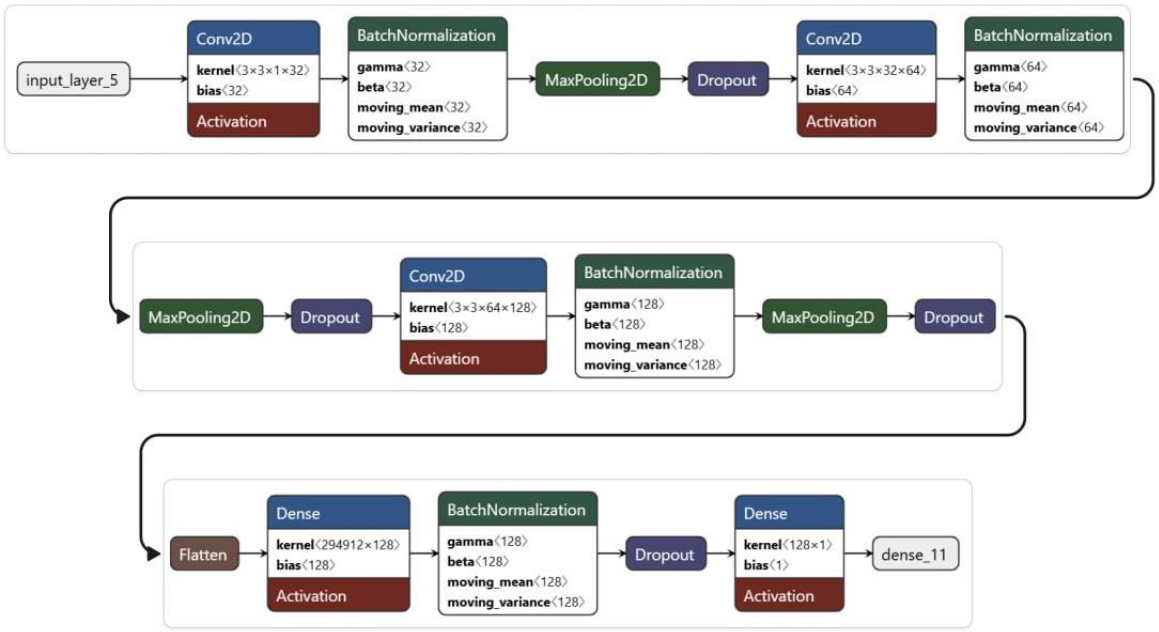

Рисунок 3.8 — Архітектура Custom CNN v1

Модель компілювалась із використанням оптимізатора Adam [18] (learning rate = 0.001), функції втрат binary_crossentropy, а метриками виступали точність (accuracy) та площа під ROC-кривою (AUC). Усі експерименти з навчання проводилися протягом 30 епох, із моніторингом поведінки валідаційної метрики AUC — це дозволило не лише відслідковувати динаміку класифікації, але й краще вловлювати моменти перенавчання.

На графіках точності, втрат та AUC (рисунок 3.9) помітно, що модель досить швидко виходить на плато: вже на 10 — 12 епосі показники стабілізуються, втрата мінімізується, а AUC перевищує 0.99. Така поведінка характерна для моделей, які «розуміють» патерн класифікації, однак можуть страждати від обмеженості узагальнення в умовах нових даних.

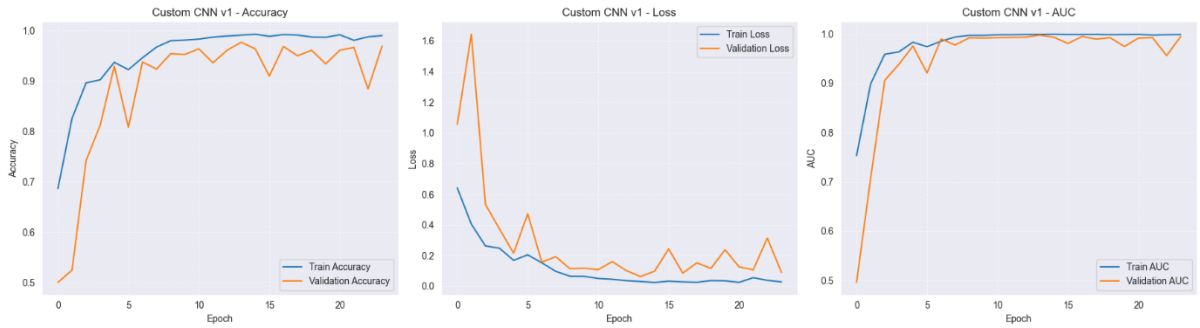

Рисунок 3.9 — Динаміка accuracy, loss та AUC для Custom CNN v1

Підсумкові метрики на тестовій вибірці можна побачити у таблиці 3.1

Таблиця 3.1 — Метрики на тестовій вибірці

Accuracy	Precision	Recall	F1 Score	AUC
0.9764	0.9769	0.9781	0.9775	0.9941

Ці показники є виключно високими як для базової моделі, особливо AUC = 0.9941 (рисунок 3.10), що свідчить про майже ідеальну здатність відрізняти класи незалежно від вибраного порогу класифікації. Водночас надзвичайно високі

значення таких метрик повинні викликати обережність, адже існує ймовірність, що модель пристосувалась до специфічних рис датасету (наприклад, до артефактів, шрифтів, неочищених позначок), а не до самих ознак перелому.

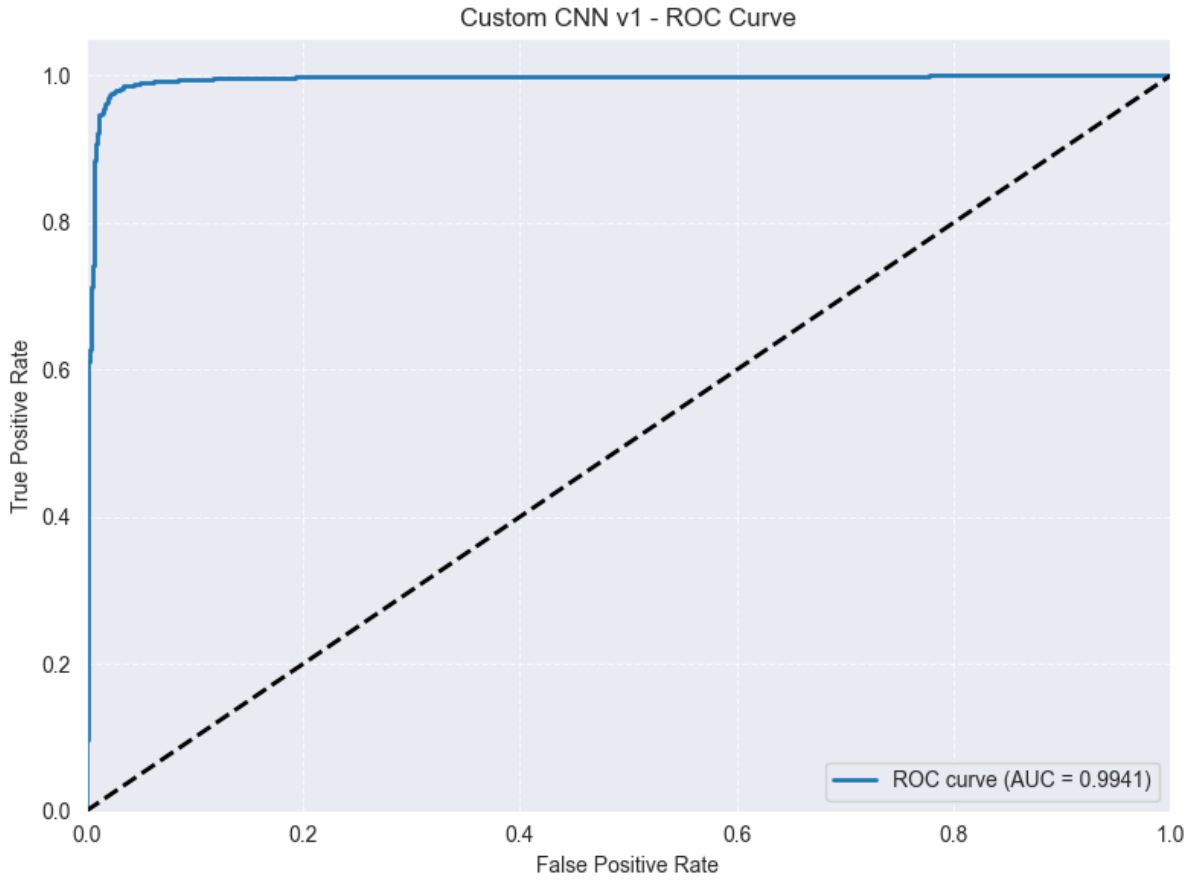

Рисунок 3.10 — ROC-крива моделі Custom CNN v1 (AUC = 0.9941)

Для більшої переконливості аналіз доповнено матрицею неточностей (рисунок 3.11), яка демонструє, що модель допустила всього 35 помилкових класифікацій: 17 хибнонегативних і 18 хибнопозитивних. Із 1480 зображень у тестовій вибірці це свідчить про високу узгодженість передбачень навіть у крайових випадках. Така збалансованість між чутливістю та специфічністю підтверджує, що модель не схиляється до надмірної переоцінки одного з класів, що часто трапляється при наявності класового дисбалансу.

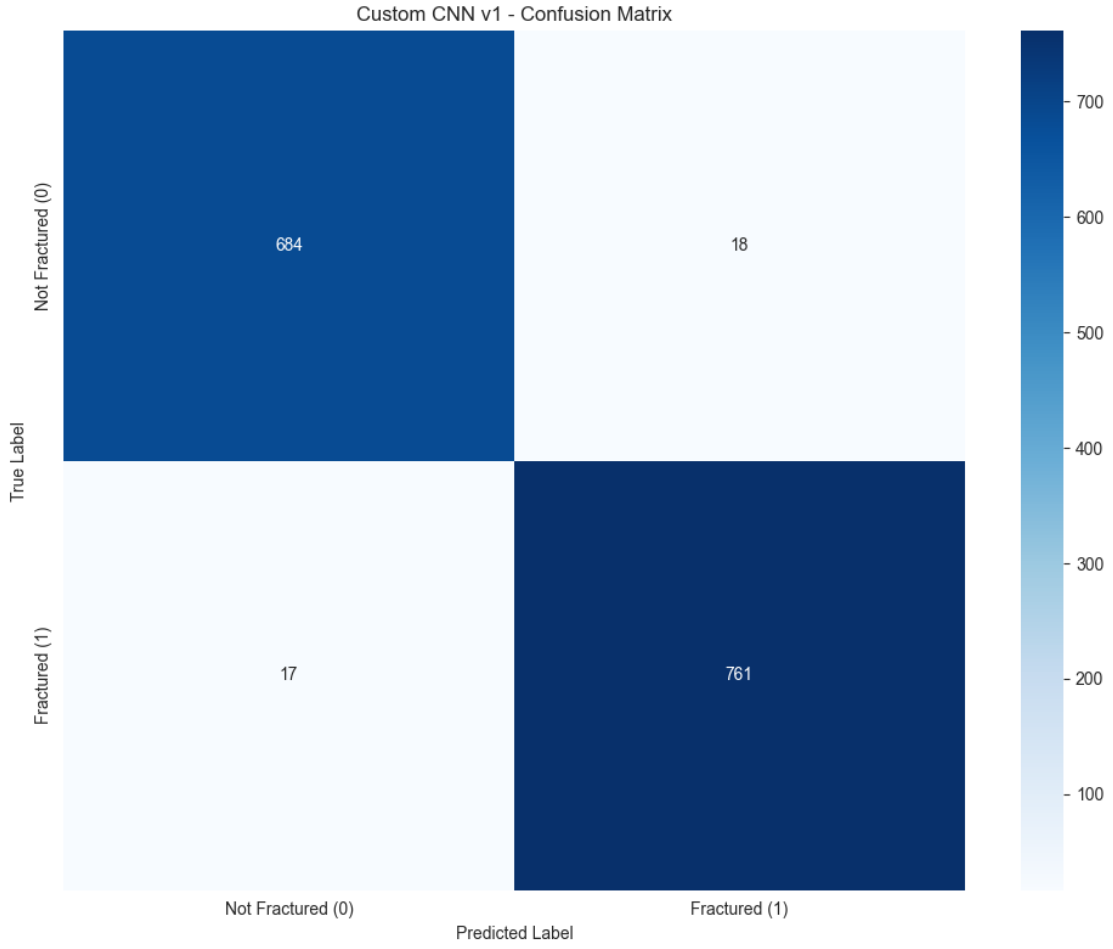

Рисунок 3.11 — Confusion matrix для моделі Custom CNN v1

Узагальнюючи, Custom CNN v1 стала міцною відправною точкою у процесі розробки. Її архітектура забезпечила високу точність, збалансовані метрики та стабільну динаміку навчання. Водночас аналіз результатів показав, що, попри свою простоту й ефективність, ця модель має певні обмеження — зокрема у гнучкості до складніших патернів та потенційному перенавчанні. Саме ці міркування стали підґрунтям для розробки наступної версії — Custom CNN v2_5, яка мала на меті підвищити стійкість, покращити узагальнення та краще контролювати складність мережі без втрати точності.

3.2.1 Модель Custom CNN v2_5

Другим варіантом архітектури, розробленим у межах дипломної роботи, стала модель Custom CNN v2_5, яка являє собою вдосконалення базової структури з використанням глибшої конфігурації, більш агресивної регуляризації та альтернативних активаційних функцій. Основною ідеєю при побудові цієї версії було зберегти базову інтуїтивну архітектуру згорткової мережі, водночас підвищивши її стійкість до перенавчання, здатність до узагальнення та виразність.

На відміну від Custom CNN v1, ця модель включає чотири згорткові блоки (а не три), що дало змогу поглибити фільтраційний рівень і забезпечити детальніше вилучення ознак (рисунок 3.11). Кожен блок містить згортку з ядром 3×3 та регуляризатором L2 [19] (l2=0.001), за якою слідують BatchNormalization, нелінійність LeakyReLU [20] (на відміну від ReLU у v1), підвибірка MaxPooling2D і Dropout від 0.2 до 0.25. Після згорткової частини модель переходить до Flatten, повнозв'язного шару на 128 нейронів із збереженням L2-регуляризації, нормалізації, LeakyReLU та Dropout=0.4. Вихідний шар залишається з sigmoid-активацією.

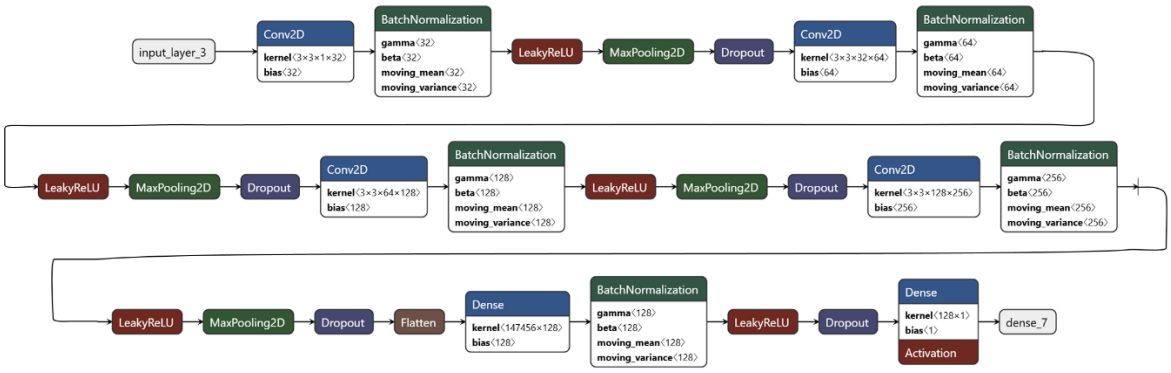

Рисунок 3.11 — Архітектура Custom CNN v2_5

Важливою зміною стало зменшення швидкості навчання до 0.0001, що дало змогу моделі навчатись повільніше, але стабільніше, уникаючи стрибків втрат на валідаційному наборі. Усі інші гіперпараметри, зокрема кількість епох (30),

функція втрат (binary_crossentropy) та метрики (accuracy, AUC), залишилися незмінними для коректного порівняння.

Динаміка навчання моделі (рисунок 3.12) демонструє стабільне зниження функції втрат і поступове зростання AUC. Помітна початкова нестабільність на валідаційному наборі свідчить про складність конфігурації, однак вже після 5 — 6 епох крива стабілізується, а загальна AUC досягає плато на рівні ~0.99, що вказує на хороше узагальнення.

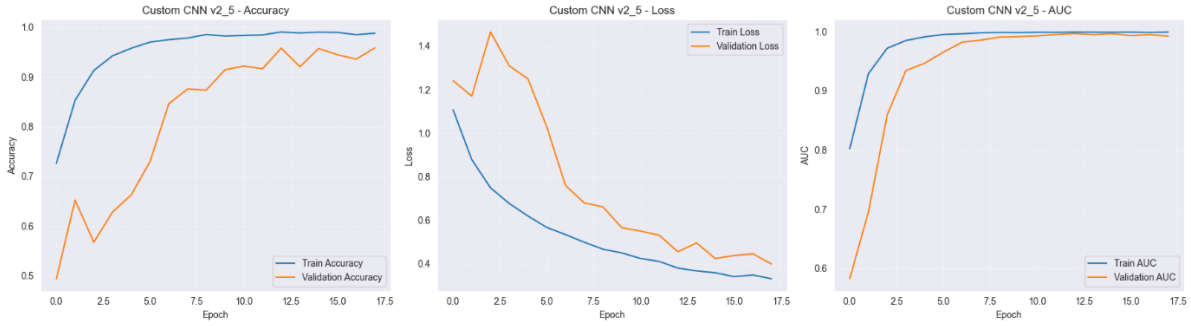

Рисунок 3.12 — Динаміка accuracy, loss і AUC при навчанні моделі Custom CNN v2_5

Підсумкові метрики на тестовій вибірці можна побачити у таблиці 3.2

Таблиця 3.2 — Метрики на тестовій вибірці

Accuracy	Precision	Recall	F1 Score	AUC
0.9520	0.9862	0.9216	0.9528	0.9918

Ці результати потребують уважної інтерпретації. З одного боку, AUC і точність залишаються надзвичайно високими, однак порівняно з v1 спостерігається менше хибнопозитивних передбачень (лише 10), що видно на матриці неточностей (рисунок 3.13). З іншого боку, кількість хибнонегативних класифікацій зросла — 61 приклад помилково віднесено до класу “без перелому”, що знижує показник Recall.

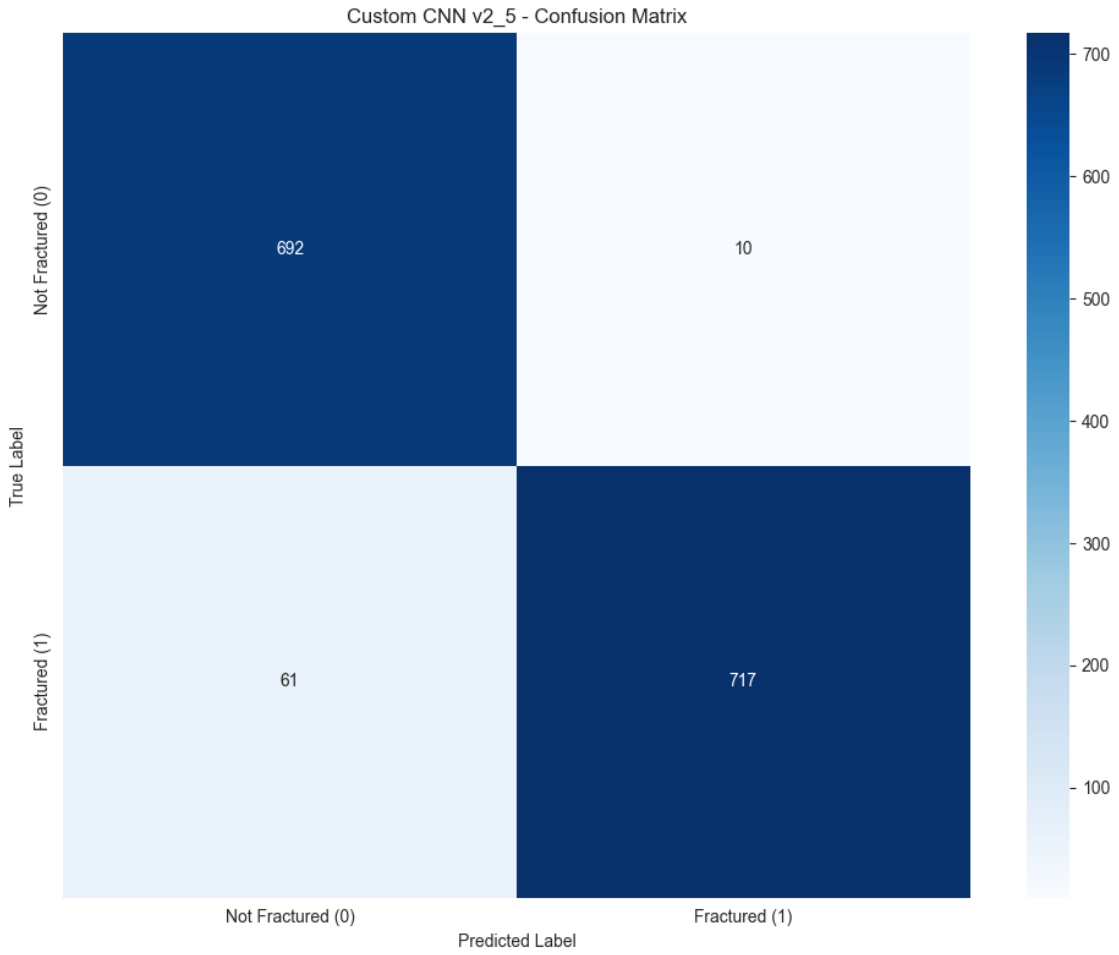

Рисунок 3.13 — Confusion matrix для Custom CNN v2_5

Це свідчить про більш обережну поведінку моделі: вона менш схильна помилково “діагностувати” перелом, що зменшує кількість хибних тривог, але водночас підвищує ймовірність пропуску справжнього випадку. Залежно від клінічного контексту це може бути як перевагою (менше помилкових діагнозів), так і ризиком (втрата істинного випадку).

Порівняння з Custom CNN v1 дає цікаві спостереження. Обидві моделі показали надзвичайно високі метрики AUC (>0.99), однак поведінка метрик різна:

v1 показала більш агресивну чутливість: Recall = 0.9781, що зменшує ймовірність пропуску перелому, але Precision = 0.9769;

v2_5 натомість підвищила Precision (0.9862), зменшивши кількість хибнопозитивних випадків, але за рахунок Recall, який опустився до 0.9216.

У практичному сенсі Custom CNN v2_5 є більш стабільною та обережною моделлю, яка краще працює з “нормальними” знімками, а також краще справляється з шумами, нетиповими прикладами чи знімками з нечіткими ознаками. Це пояснюється застосуванням регуляризації (L2), Dropout і менш різкої активації LeakyReLU, яка зменшує “залипання” нейронів на нульовій активації.

Узагальнюючи, Custom CNN v2_5 виявилась покращеним варіантом порівняно з v1 з точки зору стабільності, контрольованого узагальнення та гнучкості архітектури. Незважаючи на деяке зменшення Recall, вона значно зменшила ймовірність хибних тривог, а загальна AUC залишилась на рівні, який можна вважати майже ідеальним. Ці характеристики роблять її перспективною для практичного застосування або подальшого донавчання на спеціалізованих медичних підвибірках.

3.2.3 Модель Custom CNN v3

Третя модель, позначена як Custom CNN v3, була спробою реалізувати глибшу та архітектурно складнішу структуру, що включає механізми залишкових зв’язків (skip-connections), більшу кількість згорткових блоків та глобальне середнє згортання. Основна гіпотеза полягала в тому, що збільшення глибини моделі та використання елементів, властивих сучасним ResNet-подібним архітектурам, дозволить моделі краще вловлювати складні просторові залежності в рентгенівських знімках.

Модель (рисунок 3.14) починається з двох послідовних згорткових шарів (32 фільтри), після яких зберігається проміжний результат як skip-з’єднання (skip1). Далі йдуть шари підвибірки, регуляризації (SpatialDropout) і наступний згортковий блок (64 фільтри), знову із збереженням skip2. Обидва з’єднання пізніше інтегруються через операцію додавання (add) з обхідним шляхом, трансформованим через згортку 1×1. У третій та четвертий блоки модель заглиблюється до 128 і 256 фільтрів відповідно, після чого використовується

GlobalAveragePooling2D, замість Flatten. Це зменшує ризик перенавчання й адаптує архітектуру до змінного розміру вхідних зображень. Ще однією новацією є експоненційне зменшення швидкості навчання, реалізоване через ExponentialDecay.

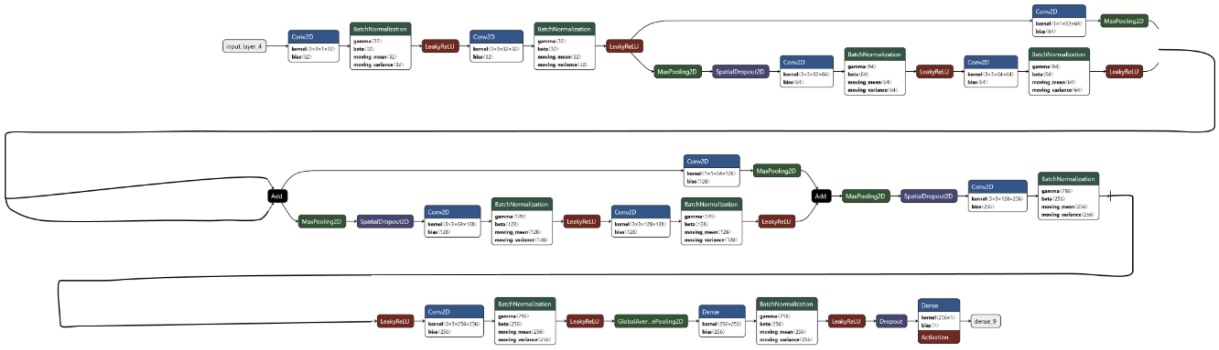

Рисунок 3.14 — Архітектура Custom CNN v3

Попри теоретичну привабливість, модель виявилась нестабільною на практиці. На графіках динаміки навчання (рисунок 3.15) видно, що хоча train accuracy і AUC поступово зростають, поведінка на валідаційній вибірці є нерівномірною та хаотичною. Метрики втрат і AUC коливаються, не демонструючи чіткої тенденції до покращення, а іноді — навіть деградацію в середині етапу навчання.

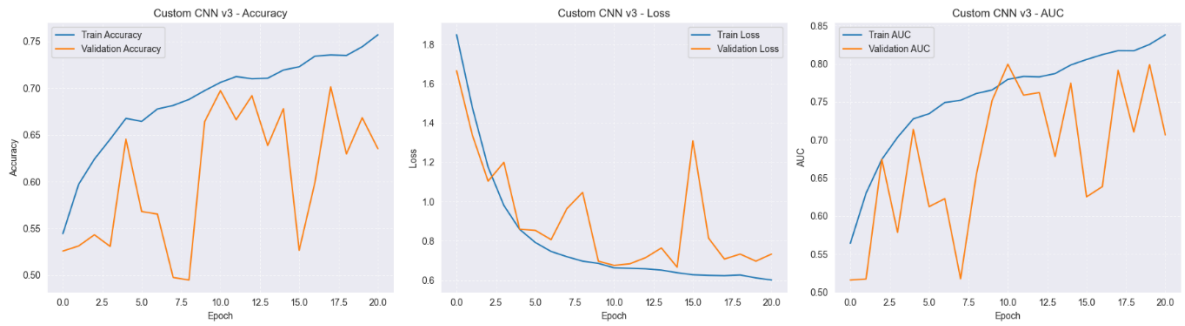

Рисунок 3.15 — Динаміка accuracy, loss та AUC для Custom CNN v3

Тестування моделі дало наступні результати (таблиця 3.3):

Таблиця 3.3 — Результати моделі Custom CNN v3

Accuracy	Precision	Recall	F1 Score	AUC
0.6980	0.6590	0.8817	0.7543	0.8118

AUC (рисунок 3.16) суттєво нижча, ніж у попередніх моделей: на 18 % гірше за Custom CNN v1 і на 18 % гірше за v2_5. Це означає, що загальна здатність моделі розрізняти між класами була обмежена, попри високу recall (чутливість). Водночас precision становить лише 0.659, що означає велику кількість хибнопозитивних випадків.

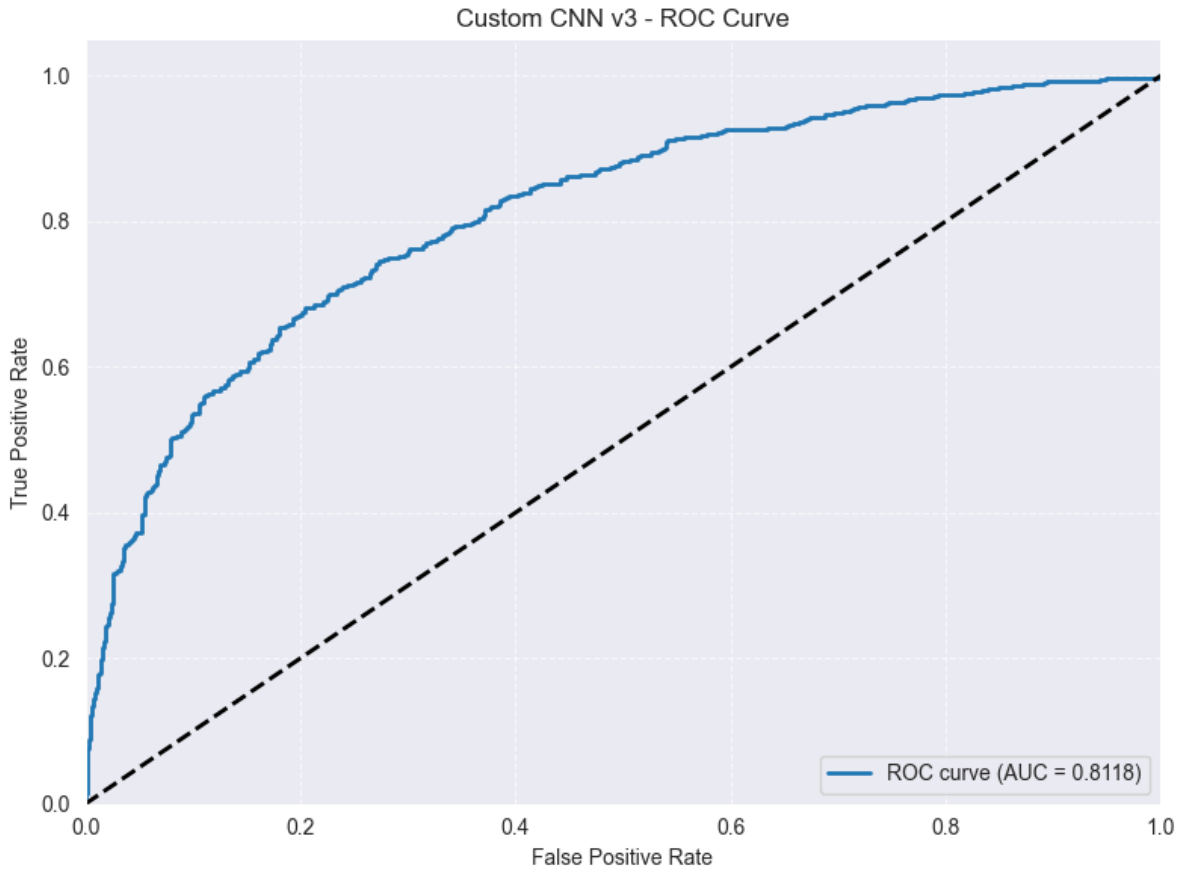

Рисунок 3.16 — ROC-крива моделі Custom CNN v3 (AUC = 0.8118)

Матриця неточностей (рисунок 3.17) також підтверджує це: модель помилилась у 355 з 702 прикладах класу «без перелому», класифікувавши їх хибно як «перелом». Це потенційно може призвести до великої кількості хибних тривог у реальному використанні, що в медичному контексті є критичною проблемою — така модель схильна «перестраховуватись».

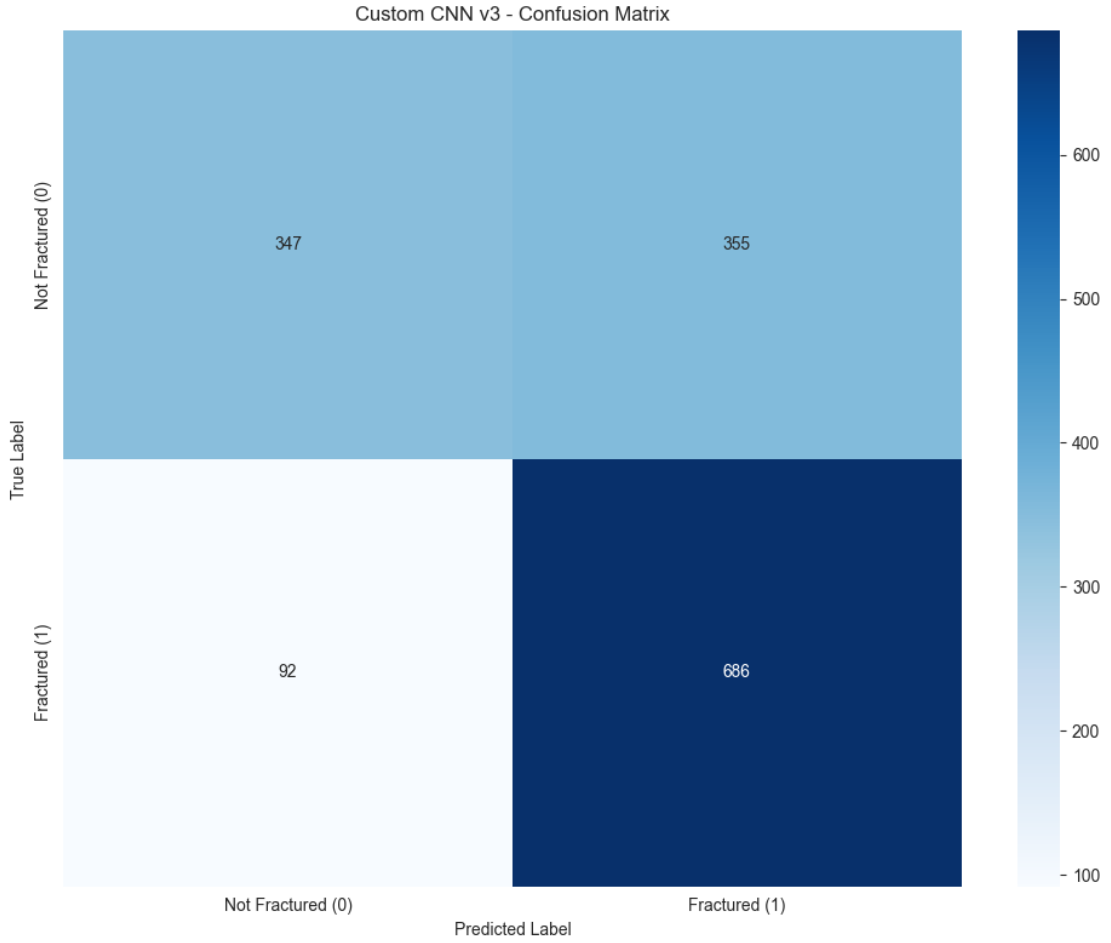

Рисунок 3.17 — Матриця неточностей для Custom CNN v3

Попри складність архітектури, Custom CNN v3 виявилась переобтяженою, з надмірною кількістю параметрів і недостатньою стабільністю. Причинами цього можуть бути:

— перенасичення шарами з’єднань, що ускладнило зворотне поширення градієнта;

— висока глибина без відповідної кількості даних — глибокі мережі краще працюють на великих обсягах даних, яких могло бути недостатньо;

— нестабільність регуляризації та дропаута (особливо SpatialDropout), які при недостатньому налаштуванні могли знищити критичну інформацію на ранніх рівнях.

Таким чином, Custom CNN v3 засвідчила обмеженість підходу «чим глибше — тим краще», якщо не забезпечено відповідне середовище, якість даних і ретельну настройку кожного компонента. Ця модель стала важливим експериментом, який показав межі стабільності архітектури та вказав на потребу у кращому балансі між складністю та контрольованістю навчального процесу.

3.2.4 Модель Custom CNN v4

Заключна модель у рамках цього дослідження — Custom CNN v4 — була розроблена як спроба збалансувати архітектурну глибину Custom CNN v3 із практичною стабільністю та точністю Custom CNN v2_5. Її мета полягала у створенні компромісного рішення, яке могло б поєднувати сильні сторони попередніх версій, одночасно зменшуючи нестабільність та перенавчання, притаманні глибоким мережам.

Custom CNN v4 (рисунок 3.18) складається з чотирьох згорткових блоків, кожен з яких включає Conv2D, BatchNormalization, LeakyReLU, MaxPooling2D та SpatialDropout2D. Регуляризація здійснюється через L2 з менш агресивним коефіцієнтом (1e−4), що дозволяє контролювати складність без надмірного «штучного спрощення» моделі. На відміну від v3, тут не використовуються залишкові зв’язки, а також спрощено конфігурацію глобального згортання та фінальної щільної частини мережі.

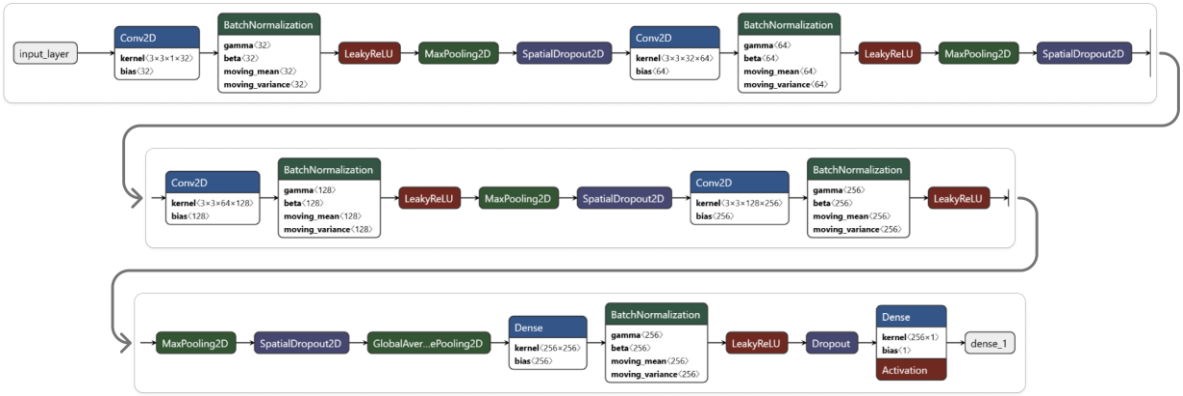

Рисунок 3.18 — Архітектура Custom CNN v4

Навчання моделі здійснювалося протягом **30 епох**, з використанням оптимізатора Adam та експоненційного розкладу швидкості навчання (learning rate decay). Як і в інших моделях, відстежувалась функція втрат, точність і AUC на тренувальній та валідаційній вибірках (рисунок 3.19).

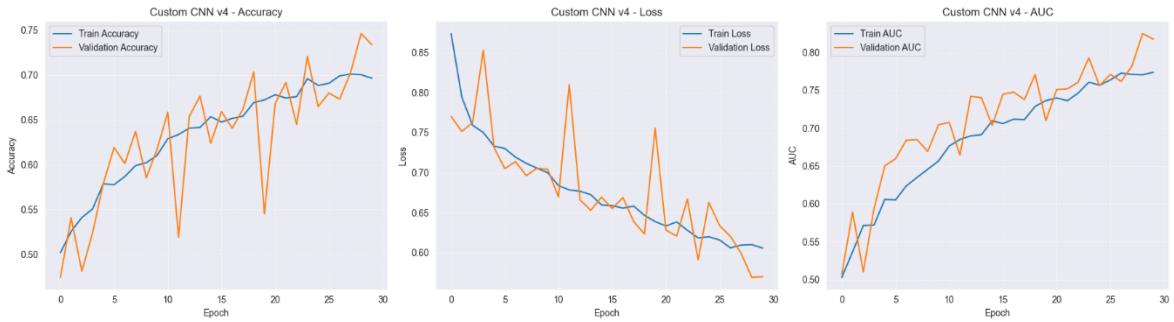

Рисунок 3.19 — Динаміка точності, втрат і AUC під час навчання Custom CNN v4

Після завершення навчання модель продемонструвала наступні результати (таблиця 4.4) на тестовій вибірці

Таблиця 4.4 — Результати Custom CNN v4

Accuracy	Precision	Recall	F1 Score	AUC
0.7581	0.8443	0.6620	0.7421	0.8367

У порівнянні з попередніми моделями, ці результати можна вважати посередніми. З одного боку, модель суттєво покращила precision у порівнянні з v3 (0.8443 проти 0.6590), що означає зниження кількості хибнопозитивних класифікацій. З іншого боку, Recall у моделі виявився найнижчим серед усіх протестованих архітектур, що свідчить про високий ризик пропуску фактичних випадків переломів, тобто хибнонегативні передбачення.

Цей баланс також видно з матриці неточностей (рисунок 3.20): з 778 справжніх випадків переломів 263 були помилково класифіковані як “без перелому”. Такий рівень хибнонегативних результатів не є прийнятним у більшості клінічних сценаріїв, оскільки може призвести до відсутності подальшої медичної діагностики.

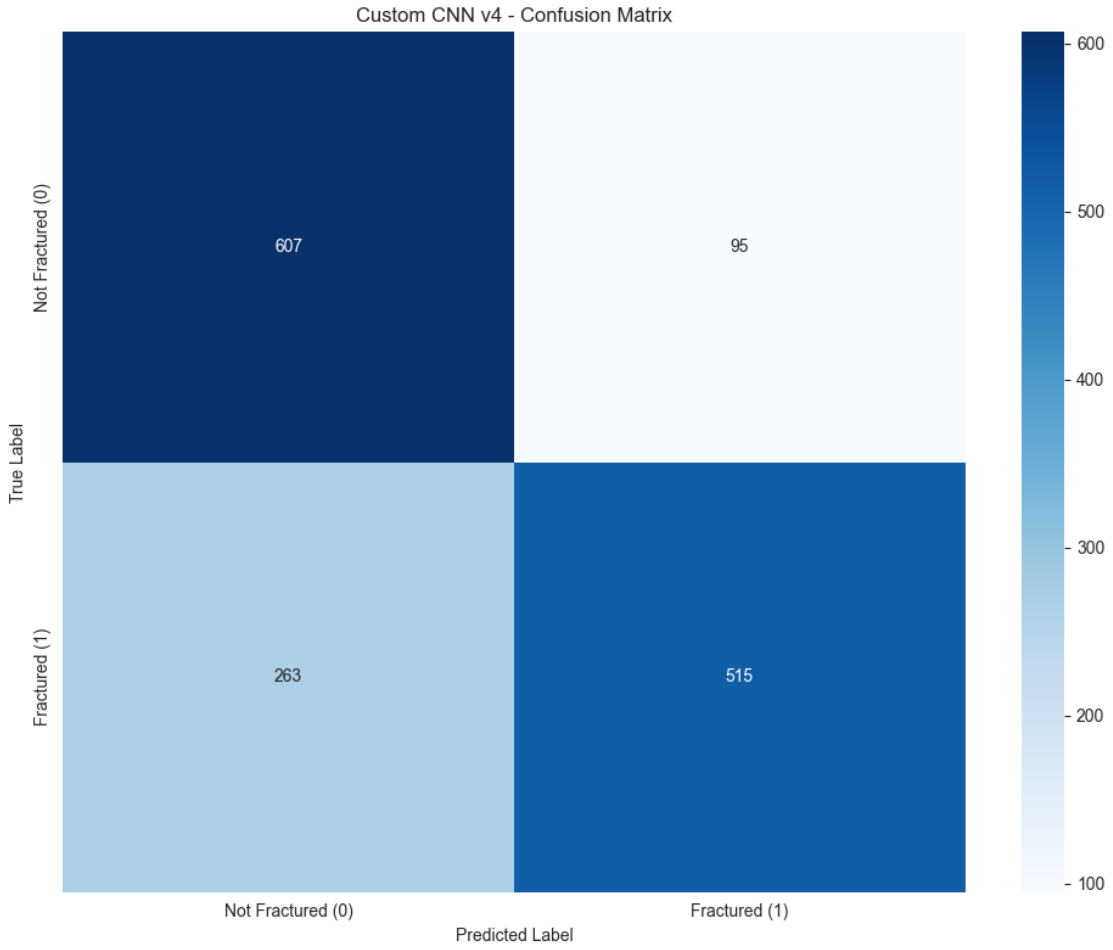

Рисунок 3.20 — Матриця неточностей Custom CNN v4

Візуальний аналіз кривих навчання свідчить про кращу стабільність, ніж у v3: валідаційні метрики не коливаються настільки хаотично, AUC поступово зростає. Проте, на відміну від v2_5 чи v1, у моделі відсутній чіткий момент виходу на плато: AUC продовжує коливатись, не досягаючи сталого рівня понад 0.90.

Модель Custom CNN v4 стала спробою знайти компроміс між складністю та стабільністю, проте її результати виявились середніми: кращими за нестабільну v3, але гіршими за точнішу v1 і найефективнішу v2_5. Її головною перевагою є краща контрольованість регуляризації та зменшення хибних позитивів. Проте низький рівень Recall і обмежений AUC свідчать, що ця версія не досягла бажаного балансу.

У клінічному контексті така модель може використовуватись лише як попередній фільтр, орієнтований на безпечне виключення явно нормальних випадків, проте не як основний діагностичний інструмент. Вона може також служити основою для подальших модифікацій — зокрема з додаванням attention-механізмів або самонавчання.

3.2.5 Загальні висновки щодо ефективності моделей

У межах цього підрозділу було здійснено експериментальне дослідження чотирьох згорткових нейронних мереж власної розробки — Custom CNN v1, v2_5, v3 та v4. Кожна з моделей мала унікальну архітектуру, різний ступінь глибини, інші стратегії регуляризації та принципи побудови (наявність або відсутність залишкових зв’язків, глобальне згортання, тип активації, форма Dropout). Аналіз метрик дозволив об’єктивно порівняти ефективність моделей та визначити оптимальну конфігурацію для задачі виявлення переломів на рентгенівських знімках.

Згідно з графіками (рисунок 3.21), модель Custom CNN v2_5 продемонструвала найвищі результати практично за всіма ключовими метриками. Зокрема, вона досягла найвищого значення Precision (0.9862), що вказує на низьку частоту хибнопозитивних спрацьовувань, а також продемонструвала AUC = 0.9918

— один із найвищих серед усіх моделей. F1 Score = 0.9528 підтверджує, що модель зберігає добрий баланс між чутливістю та специфічністю.

Модель Custom CNN v1, попри свою просту архітектуру, показала надзвичайно високі показники точності (Accuracy = 0.9764) та Recall (0.9781), однак у більш детальному аналізі виявилось, що вона має тенденцію до надмірної чутливості й може бути схильною до перенавчання. Вона є якісною стартовою точкою, однак поступається v2_5 за стабільністю та ефективністю контролю за хибними передбаченнями.

Моделі Custom CNN v3 та v4 мали нижчі показники на всіх метриках, попри те, що v3 мала складну ResNet-подібну структуру. Зокрема, Custom CNN v3 виявилася нестабільною, із AUC = 0.8118 та значним розкидом результатів під час навчання, що свідчить про низьку узагальнюваність і чутливість до параметрів. Модель v4, натомість, мала дещо вищу стабільність, але так і не досягла високих результатів: AUC = 0.8367, Recall = 0.6620 — найнижчий серед усіх моделей.

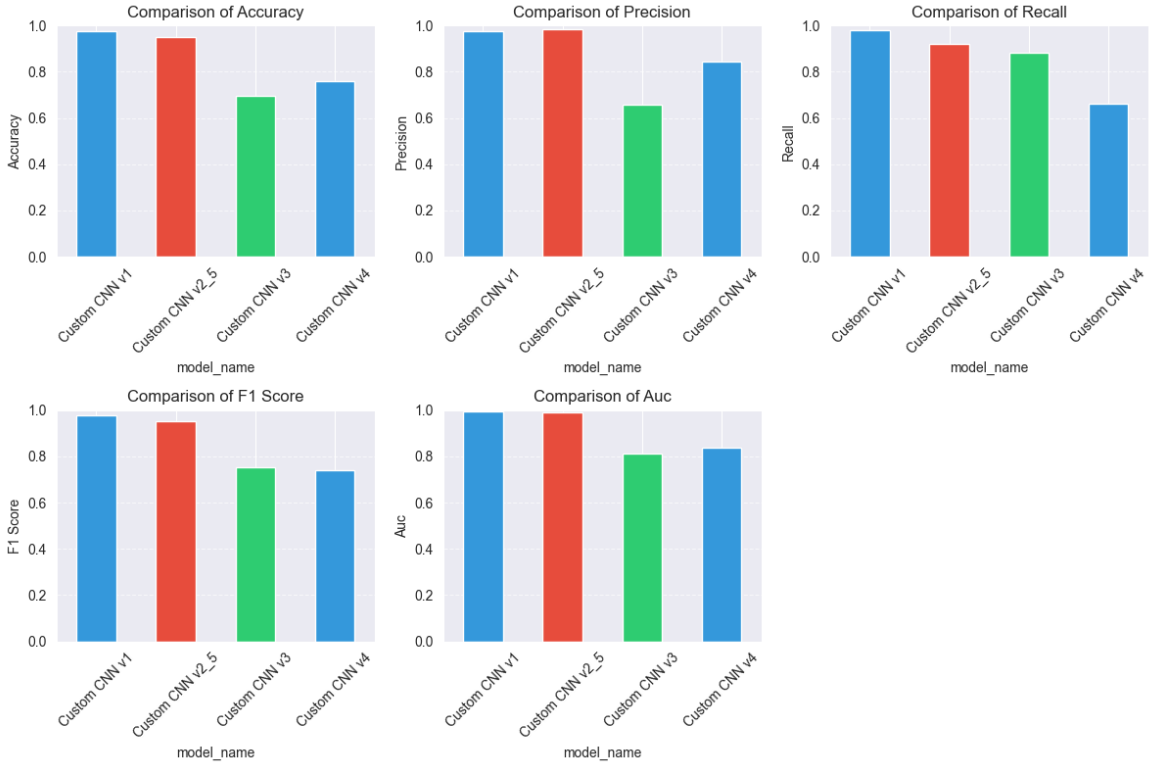

Рисунок 3.21 — Порівняння моделей за метриками точності, повноти, F1-міри, Precision та AUC

Таким чином, з урахуванням багатофакторного аналізу — включаючи оцінку основних метрик класифікації (AUC, точність, повнота, F1-міра), стабільність навчання на всіх етапах, поведінку моделі на валідаційних і тестових вибірках, а також рівень архітектурної складності — було сформульовано висновок, що саме модель Custom CNN v2_5 є найбільш збалансованим і надійним варіантом серед усіх протестованих конфігурацій. Вона не лише забезпечила найвищі показники за низкою ключових характеристик, але й продемонструвала здатність до стабільного узагальнення без втрати чутливості або точності в умовах складних вхідних даних.

Крім того, її архітектура виявилася технічно ефективною: за рахунок використання регуляризації (L2), помірного Dropout, активації LeakyReLU та розумного співвідношення глибини і кількості параметрів вдалося досягти оптимального балансу між обчислювальними витратами та якістю результатів. Саме ці особливості дозволили безперешкодно інтегрувати Custom CNN v2_5 у веб-додаток як основну модель для класифікації зображень.

На практиці це означає, що в межах запропонованого користувацького інтерфейсу саме ця модель відповідає за аналіз рентгенівських знімків, забезпечуючи надійні передбачення з високою достовірністю. Її ефективність, підтверджена як у кількісному, так і в якісному вимірах, свідчить про готовність до застосування в реальних медичних або освітніх умовах, де швидкість обробки, стабільність роботи та точність рішень мають критичне значення.

4 ІНСТАЛЯЦІЯ ТА ФУНКЦІОНАЛ ВЕБ-ДОДАТКУ

У цьому розділі наведено опис процесу інсталяції та розгортання програмного забезпечення для виявлення переломів на рентгенівських знімках, розробленого в межах дипломної роботи. Також представлено загальні вимоги до обчислювальної техніки, необхідної для коректної роботи застосунку, та демонстрацію його функціональних можливостей у тестовому середовищі. Інструкції орієнтовані на кінцевого користувача або технічного спеціаліста, який виконує розгортання системи на локальному ПК або сервері.

4.1 Інсталяція програмного забезпечення та системні вимоги

Розроблене програмне забезпечення реалізоване у вигляді локального веб-додатку, що забезпечує інтуїтивно зрозумілий інтерфейс для виявлення переломів на рентгенівських знімках із використанням нейронної мережі. Основу клієнтської частини становить фреймворк Streamlit, який дозволяє запускати застосунок без необхідності складного розгортання або окремого серверного середовища. Для запуску достатньо мати встановлене середовище Python та актуальні версії бібліотек глибинного навчання.

Система є кросплатформенною та може бути запущена під операційними системами Windows, macOS або Linux. Рекомендованою конфігурацією є комп’ютер із сучасним процесором, не менше ніж 8 ГБ оперативної пам’яті, бажано з підтримкою багатопоточності. Наявність графічного прискорювача (GPU) не є обов’язковою, проте може суттєво пришвидшити обробку зображень та інференс моделі. Для повноцінної роботи застосунку також необхідна стабільна версія браузера (Google Chrome, Mozilla Firefox або аналоги), оскільки інтерфейс запускається локально через HTTP-протокол.

Процес інсталяції програмного забезпечення передбачає завантаження проєкту з репозиторію, налаштування ізольованого середовища Python

(рекомендовано для уникнення конфліктів залежностей), встановлення необхідних бібліотек згідно з файлом requirements.txt та запуск головного файлу застосунку app.py. Усі дії виконуються через інтерфейс командного рядка.

Після запуску застосунок доступний у локальній мережі за адресою, яку автоматично згенерує Streamlit, як правило — http://localhost:8501. Інтерфейс дозволяє користувачеві завантажити зображення рентгенівського знімка у форматі .jpg, .jpeg або .png, обрати попередньо навчену модель, ініціювати процес класифікації та отримати результат у вигляді ймовірності наявності перелому. Результати виводяться з візуальним супроводом: прогрес-баром, кольоровим повідомленням та додатковим текстовим поясненням. Уся обробка даних виконується локально, що гарантує конфіденційність медичної інформації.

Таким чином, застосунок є автономним, не потребує доступу до зовнішніх API або баз даних і може бути легко розгорнутий на будь-якому комп’ютері, що відповідає мінімальним вимогам. Це дозволяє інтегрувати систему як допоміжний інструмент у діагностичні кабінети, навчальні лабораторії або дослідницькі середовища.

4.2 Робота користувача з програмною системою

У цьому підрозділі наведено приклад типового сценарію роботи з реалізованою системою, що дозволяє здійснювати автоматичний аналіз рентгенівських знімків з метою виявлення можливих переломів. Інтерфейс було розроблено з урахуванням принципів зручності, доступності та мінімізації технічних бар’єрів для кінцевого користувача — лікаря, лаборанта або дослідника.

Після запуску веб-застосунку користувач потрапляє на головний екран (рисунок 4.1), де одразу відображається назва системи, коротка інструкція щодо її використання, а також меню навігації. Цей екран є початковою точкою взаємодії з користувачем і виконує інформативну функцію.

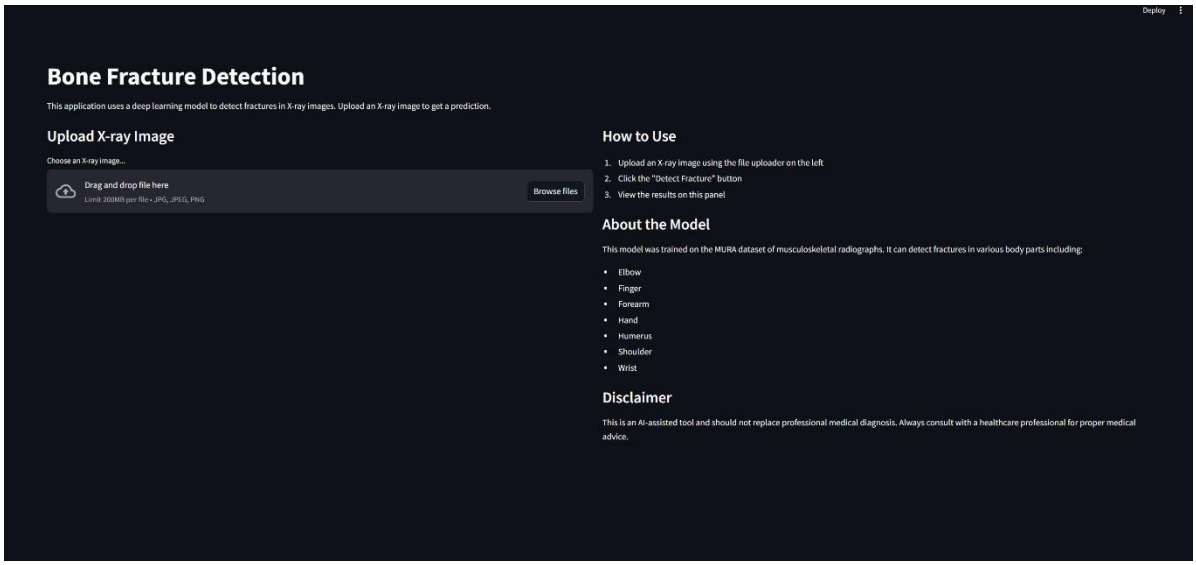

Рисунок 4.1 — Головний екран додатка

Ключовий функціональний блок розташований нижче — це секція для завантаження зображення (рисунок 4.2), яка дозволяє обрати файл рентгенівського знімка з локального комп’ютера. Система підтримує найбільш поширені графічні формати — .jpg, .jpeg, .png. Одразу після завантаження зображення відображається у вікні перегляду, що дозволяє переконатися у коректності вибраного файлу.

Рисунок 4.2 — Секція для завантаження зображення

Після завантаження користувач переходить до секції запуску аналізу (рисунок 4.3), яка за замовчуванням містить оптимальну конфігурацію — використання моделі Custom CNN v2_5, яка, згідно з результатами попереднього розділу, продемонструвала найвищу ефективність класифікації. Додаткових дій із боку користувача не вимагається, що мінімізує ризик помилок і спрощує використання системи навіть для недосвідчених користувачів.

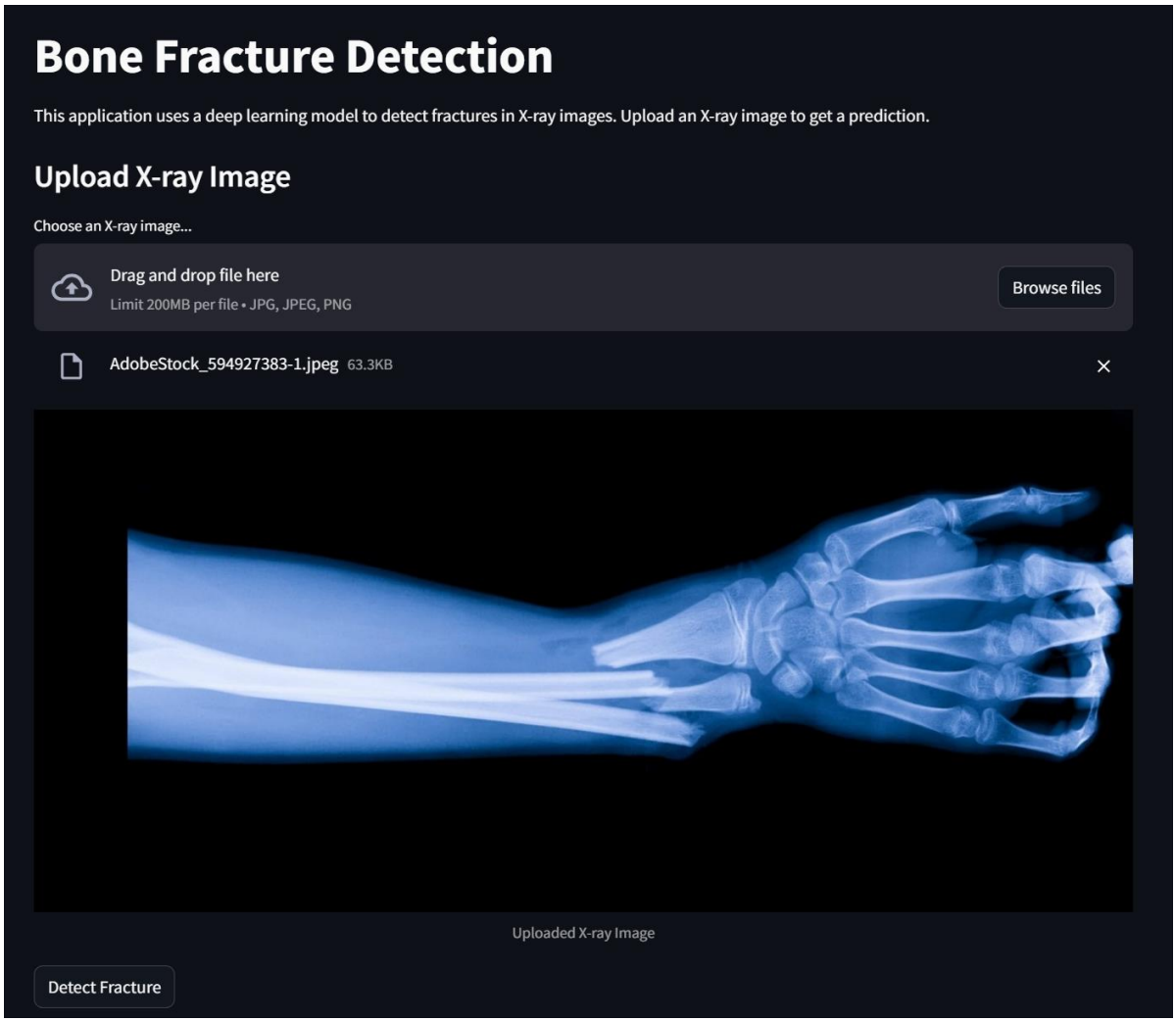

Рисунок 4.3 — Готова конфігурація для запуску аналіза

Натисненням кнопки «Detect Fracture» ініціюється процес класифікації зображення. У цей момент система виконує попередню обробку зображення (масштабування, нормалізація), подає його на вхід нейронної мережі, а після завершення обчислень виводить результат. Результат класифікації (рисунок 4.4) представлено у формі ймовірності — значення від 0 до 1, що інтерпретується як ступінь впевненості в наявності перелому. Крім того, система виводить текстове повідомлення (наприклад, «перелом виявлено» або «перелом не виявлено») та індикатор (кольоровий блок або прогрес-бар), що полегшує візуальне сприйняття.

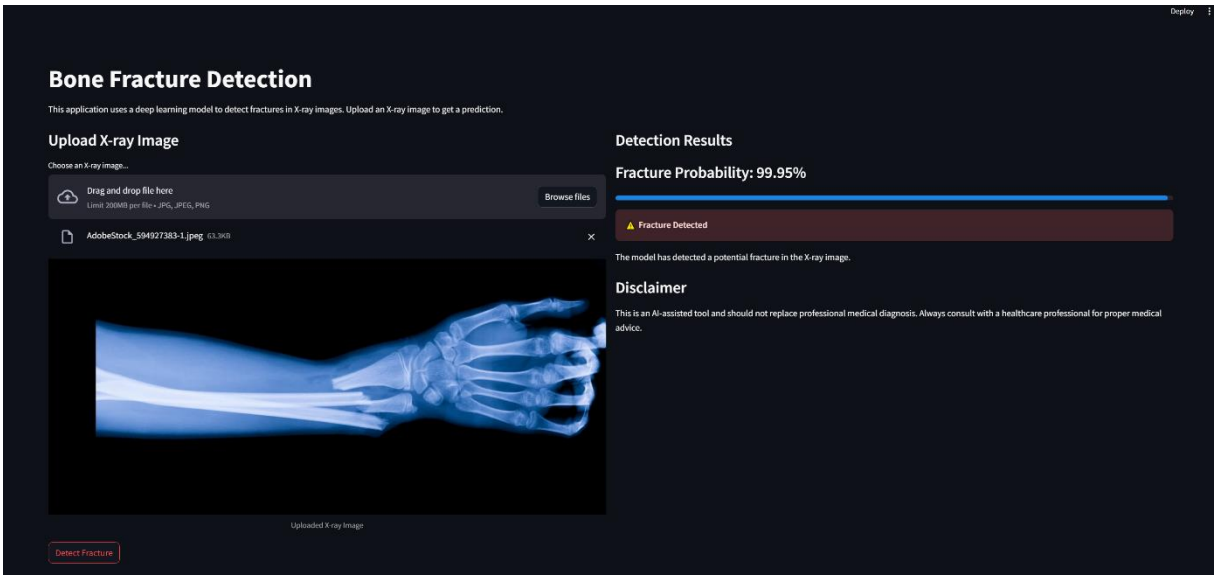

Рисунок 4.4 — Результат роботи програми

У разі повторного використання користувач може просто завантажити нове зображення без необхідності перезапуску системи. Така логіка реалізована через механізм повторного використання ресурсу моделі, що дозволяє уникнути надмірного навантаження на обчислювальну систему.

Розроблений інтерфейс є прикладом простого, але функціонального користувацького рішення, яке дозволяє швидко інтегрувати можливості глибинного навчання в медичну практику без залучення спеціалістів з інформаційних технологій.

ВИСНОВКИ

Під час виконання дипломної роботи було виконано огляд та аналіз згорткових нейронних мереж для виявлення перелому на рентгенівському знімку, а також створено веб застосунок для реалізації результату роботи нейронної мережі.

Проведено аналіз існуючих рішень у сфері автоматичного виявлення переломів за рентгенівськими знімками. Було вивчено актуальні дослідження, в яких застосовуються різні варіації CNN-архітектур, зокрема моделі локалізації та класифікації. Встановлено, що хоча сучасні моделі демонструють високу точність, вони потребують ретельного налаштування, обмежені у здатності до генералізації та чутливі до нерівномірності даних. Це обґрунтовує необхідність розробки власної архітектури, оптимізованої під конкретну задачу, з акцентом на простоту, стабільність і контрольовану складність.

У якості даних для тренування нейронних мереж, було обрано два набори даних з сайту Kaggle. Вони містять велику кількість знімків різних частин тіла, а також аугментовані знімки. У якості програмних засобів обрано мову Python, у якості основних бібліотек — TensorFlow, Pandas та Streamlit.

Аналіз даних показав, що набори даних містять дублікати, як в межах свого набору даних так і в межах двох. Було проведено виявлення та усунення дублікатів у кожному датасеті окремо, а також між ними. Очищення було реалізовано за допомогою хешування (MD5), що забезпечило точне виявлення однакових зображень навіть за умови незначних змін у структурі. Після цього всі зображення було уніфіковано: масштабовано до розміру 384×384 пікселів, переведено у відтінки сірого, нормалізовано до діапазону [0, 1], а також сформовано у форматі, сумісному з TensorFlow. Додатково було виконано стратифікований поділ вибірки на тренувальний, валідаційний і тестовий набори, що забезпечило збалансованість і відтворюваність експериментів.

У ході дослідження було розроблено та порівняно чотири архітектури згорткових нейронних мереж з поступовим ускладненням структури. Базова

модель (v1) показала хороші результати, але мала ознаки перенавчання. Архітектури v3 і v4, з більшою глибиною та залишковими зв’язками, виявились менш стабільними на валідаційних даних, що свідчить про переобчислення або перенавантаження моделі. Модель v2_5, у якій поєднано помірну глибину, L2-регуляризацію, Dropout та активацію LeakyReLU, виявилась найстабільнішою: вона досягла найвищого балансу між точністю, AUC, повнотою та узагальнюючою здатністю. Саме ця модель була обрана як основна для впровадження у програмну систему.

На основі отриманої моделі було створено повноцінний веб-застосунок, який дозволяє користувачеві у зручній формі взаємодіяти з нейронною мережею. Інтерфейс реалізовано у середовищі Streamlit, що забезпечує швидке розгортання та масштабування. Користувач може завантажити рентгенівський знімок, отримати результат класифікації у вигляді тексту, ймовірнісного прогнозу та візуального відображення. Такий застосунок може бути використаний як допоміжний інструмент для попереднього медичного аналізу або як прототип для подальшого вдосконалення систем комп’ютерної діагностики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Deep Residual Learning for Image Recognition / K. He та ін. IEEE. 2016. ISSN 1063-6919. URL: https://doi.org/10.1109/CVPR.2016.90.

2. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv. 2015. URL: https://doi.org/10.48550/arXiv.1409.1556.

3. Tan M., V. Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ArXiv. 2020. URL: https://doi.org/10.48550/arXiv.1905.11946.

4. Meza G., Ganta D., Gonzalez Torres S. Deep Learning Approach for Arm Fracture Detection Based on an Improved YOLOv8 Algorithm. Algorithms. 2024. Vol. 17, no. 11. P. 19. DOI: https://doi.org/10.3390/a17110471.

5. Skeletal Fracture Detection with Deep Learning: A Comprehensive Review / Z. Su et al. Diagnostics. 2023. Vol. 13, no. 20. P. 21. DOI: https://doi.org/10.3390/diagnostics13203245.

6. Critical evaluation of deep neural networks for wrist fracture detection / A. M. Raisuddin et al. Scientific Reports. 2021. Vol. 11, no. 1. P. 11. DOI: https://doi.org/10.1038/s41598-021-85570-2.

7. Bagaria R., Wadhwani S., Wadhwani A. K. Bone Fracture Detection in X-ray Images using Convolutional Neural Network. SCRS Conference Proceedings on Intelligent Systems, India, 2021. SCRS, 2022. P. 459-466. ISBN 978-93-91842-08-6. DOI: https://doi.org/10.52458/978-93-91842-08-6-43

8. Python 3.12.9 documentation. Python. docs. URL: https://docs.python.org/release/3.12.9.

9. Tensorflow. URL: https://www.tensorflow.org/.

10. Chollet F. Deep Learning with Python, Second Edition. 2021. 504 c. ISBN 9781617296864.

11. Matplotlib. Tutorials. URL: https://matplotlib.org/stable/tutorials/index.

12. Streamlit documentation. Streamlit. docs. URL: https://docs.streamlit.io/.

13. Scikit-learn. Stable. URL: https://scikit-learn.org/stable/.

14. Madushani R. Bone Fracture Multi-Region X-ray Data. Kaggle. URL: https://www.kaggle.com/datasets/bmadushanirodrigo/fracture-multi-region-x-ray-data/data (дата звернення: 17.05.2025).

15. Jalil O. Bone Fracture Dataset. Kaggle. URL: https://www.kaggle.com/datasets/osamajalilhassan/bone-fracture-dataset (дата звернення: 17.05.2025).

16. MD5 hash in Python. Geeksforgeeks. 09.05.2025. URL: https://www.geeksforgeeks.org/md5-hash-python/.

17. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ArXiv. 2015. URL: https://www.researchgate.net/publication/272194743_Batch_Normalization_Accelerating_Deep_Network_Training_by_Reducing_Internal_Covariate_Shift.

18. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. ArXiv. 2017. URL: https://doi.org/10.48550/arXiv.1412.6980.

19. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition / A. Géron. O'Reilly Media, Inc., 2019. ISBN 9781492032649.

20. Improving neural networks by preventing co-adaptation of feature detectors / G. E. Hinton та ін. ArXiv. 2012. URL: https://doi.org/10.48550/arXiv.1207.0580.

ДОДАТОК А

Реалізація архітектури та підготовки до тренування чотирьох моделей

Програмні засоби:

— мова програмування – **Python**;

— фреймворк глибинного навчання – **TensorFlow 2.x з Keras API**;

— реалізація згорткових нейронних мереж – **keras.Sequential / keras.Model**;

— регуляризація та нормалізація – **L2, BatchNormalization, Dropout**;

— оптимізація – **Adam, ExponentialDecay**;

— підготовка зображень – **OpenCV (cv2), NumPy, Pillow**;

— поділ на вибірки та розрахунок метрик – **scikit-learn**;

— візуалізація процесу навчання – **Matplotlib, Seaborn**.

```
def build_custom_cnn_v1(input_shape=(384, 384, 1)):
    """
    Build a simple CNN model for bone fracture detection.

    Args:
        input_shape (tuple): Shape of input images (height, width, channels)

    Returns:
        keras.Model: Compiled model ready for training
    """
    model = models.Sequential([
        Input(shape=input_shape),

        # First convolutional block
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Second convolutional block
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Third convolutional block
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Flatten and dense layers
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy', keras.metrics.AUC(name='auc_2')]
    )
```

```
    return model


def build_custom_cnn_v2_5(input_shape=(384, 384, 1)):
    """
    Hybrid CNN model with L2 regularization.
    """
    l2 = keras.regularizers.l2(0.001)

    model = keras.Sequential([
        Input(shape=input_shape),

        layers.Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2),
        layers.BatchNormalization(),
        layers.LeakyReLU(negative_slope=0.1),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.2),

        layers.Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2),
        layers.BatchNormalization(),
        layers.LeakyReLU(negative_slope=0.1),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.2),

        layers.Conv2D(128, (3, 3), padding='same', kernel_regularizer=l2),
        layers.BatchNormalization(),
        layers.LeakyReLU(negative_slope=0.1),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        layers.Conv2D(256, (3, 3), padding='same', kernel_regularizer=l2),
        layers.BatchNormalization(),
        layers.LeakyReLU(negative_slope=0.1),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        layers.Flatten(),

        layers.Dense(128, kernel_regularizer=l2),
        layers.BatchNormalization(),
        layers.LeakyReLU(negative_slope=0.1),
        layers.Dropout(0.4),

        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.0001),
        loss='binary_crossentropy',
        metrics=['accuracy', keras.metrics.AUC(name='auc_2')]
    )

    return model


def build_custom_cnn_v3(input_shape=(384, 384, 1)):
    """
    Build an advanced CNN model with residual connections.

    Args:
        input_shape (tuple): Shape of input images (height, width, channels)

    Returns:
        keras.Model: Compiled model ready for training
    """

    l2 = keras.regularizers.l2(0.001)  # L2 regularization
    # Input layer
    inputs = keras.Input(shape=input_shape)

    # First convolutional block
    x = layers.Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2)(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
    x = layers.Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(negative_slope=0.1)(x)
```

```
# Store the output before pooling for skip connection
skip1 = x

# Pooling and dropout
x = layers.MaxPooling2D((2, 2))(x)
x = layers.SpatialDropout2D(0.25)(x)

# Second convolutional block
x = layers.Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)
x = layers.Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)

# Process skip1 connection to match current dimensions
skip1 = layers.Conv2D(64, (1, 1), padding='same', kernel_regularizer=l2)(skip1)
skip1 = layers.MaxPooling2D((2, 2))(skip1)

# Add skip connection
x = layers.add([x, skip1])

# Store the output before pooling for skip connection
skip2 = x

# Pooling and dropout
x = layers.MaxPooling2D((2, 2))(x)
x = layers.SpatialDropout2D(0.25)(x)

# Third convolutional block
x = layers.Conv2D(128, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)
x = layers.Conv2D(128, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)

# Process skip2 connection to match current dimensions
skip2 = layers.Conv2D(128, (1, 1), padding='same', kernel_regularizer=l2)(skip2)
skip2 = layers.MaxPooling2D((2, 2))(skip2)

# Add skip connection
x = layers.add([x, skip2])

# Pooling and dropout
x = layers.MaxPooling2D((2, 2))(x)
x = layers.SpatialDropout2D(0.25)(x)

# Fourth convolutional block (new)
x = layers.Conv2D(256, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)
x = layers.Conv2D(256, (3, 3), padding='same', kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)

# Global pooling
x = layers.GlobalAveragePooling2D()(x)

# Fully connected layers
x = layers.Dense(256, kernel_regularizer=l2)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(negative_slope=0.1)(x)
x = layers.Dropout(0.5)(x)

# Output layer
outputs = layers.Dense(1, activation='sigmoid')(x)

# Create model
model = keras.Model(inputs, outputs)

# Learning rate scheduler
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.001,
    decay_steps=1000,
    decay_rate=0.9,
    staircase=True
)
```

```
    # Compile model
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
        loss='binary_crossentropy',
        metrics=['accuracy', keras.metrics.AUC(name='auc_2')]
    )

    return model


def build_custom_cnn_v4(input_shape=(384, 384, 1)):
    """
    Optimized CNN model (v4) combining strengths of v2.5 and v3 with regularization.
    """
    l2 = keras.regularizers.l2(1e-4)
    inputs = keras.Input(shape=input_shape)

    # Block 1
    x = layers.Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2)(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(0.1)(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.SpatialDropout2D(0.25)(x)

    # Block 2
    x = layers.Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(0.1)(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.SpatialDropout2D(0.25)(x)

    # Block 3
    x = layers.Conv2D(128, (3, 3), padding='same', kernel_regularizer=l2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(0.1)(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.SpatialDropout2D(0.25)(x)

    # Block 4
    x = layers.Conv2D(256, (3, 3), padding='same', kernel_regularizer=l2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(0.1)(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.SpatialDropout2D(0.25)(x)

    # Global pooling instead of Flatten
    x = layers.GlobalAveragePooling2D()(x)

    # Dense layers
    x = layers.Dense(256, kernel_regularizer=l2)(x)
    x = layers.BatchNormalization()(x)
    x = layers.LeakyReLU(0.1)(x)
    x = layers.Dropout(0.5)(x)

    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs, outputs)

    # Learning rate schedule
    lr_schedule = keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=0.001,
        decay_steps=1000,
        decay_rate=0.9,
        staircase=True
    )

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
        loss='binary_crossentropy',
        metrics=['accuracy', keras.metrics.AUC(name='auc_2')]
    )

    return model
```

```
def train_model(model, model_name, train_df, val_df, epochs=30, batch_size=32, target_size=(384, 384),
                with_class_weights=True):
    """
    Train a model and save the best version.

    Args:
        model (keras.Model): Model to train
        model_name (str): Name for saving the model
        train_df (pandas.DataFrame): Training data
        val_df (pandas.DataFrame): Validation data
        epochs (int): Number of training epochs
        batch_size (int): Batch size
        target_size (tuple): Image dimensions
        with_class_weights (bool): Whether to use class weights

    Returns:
        tuple: (trained_model, history)
    """
    # Create directories for checkpoints and logs
    checkpoint_dir = 'checkpoints'
    os.makedirs(checkpoint_dir, exist_ok=True)

    log_dir = os.path.join('logs', model_name + '_retrained')
    os.makedirs(log_dir, exist_ok=True)

    # Set up callbacks
    callbacks = [
        ModelCheckpoint(
            filepath=os.path.join(checkpoint_dir, f'{model_name}_retrained_best.h5'),
            monitor='val_auc_2',
            mode='max',
            save_best_only=True,
            verbose=1
        ),
        EarlyStopping(
            monitor='val_auc_2',
            mode='max',
            patience=5,
            restore_best_weights=True,
            verbose=1
        ),
        TensorBoard(
            log_dir=log_dir,
            histogram_freq=1
        )
    ]

    if with_class_weights == True:
        # Calculate class weights to handle imbalance
        class_counts = train_df['label_binary'].value_counts()
        total = class_counts.sum()
        class_weights = {
```

```
            0: total / (2 * class_counts[0]),
            1: total / (2 * class_counts[1])
        }
        print(f"Class weights: {class_weights}")
    else:
        class_weights = None

    # Set up data generators
    train_generator = create_data_generator(
        train_df,
        batch_size=batch_size,
        target_size=target_size,
        class_weights=class_weights
    )

    val_generator = create_data_generator(
        val_df,
        batch_size=batch_size,
        target_size=target_size,
        shuffle=False
    )

    # Calculate steps per epoch
    steps_per_epoch = len(train_df) // batch_size
    validation_steps = len(val_df) // batch_size

    # Train the model
    print(f"\nTraining {model_name}...")
    history = model.fit(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        validation_data=val_generator,
        validation_steps=validation_steps,
        callbacks=callbacks,
        verbose=1
    )

    # Save the final model
    model_dir = 'models'
    os.makedirs(model_dir, exist_ok=True)
    model.save(os.path.join(model_dir, f'{model_name}_retrained.h5'))

    return model, history

# Create a plots directory
os.makedirs('plots', exist_ok=True)

# Train model v1
print("Training Custom CNN v1...")
model_v1 = build_custom_cnn_v1()
model_v1.summary()
model_v1, history_v1 = train_model(model_v1, 'custom_cnn_v1', train_df, val_df, with_class_weights=False)
```

```
plot_training_history(history_v1, 'Custom CNN v1')

# Train model v2_5
print("\n Training Custom CNN v2_5...")
model_v2_5 = build_custom_cnn_v2_5()
model_v2_5.summary()
model_v2_5, history_v2_5 = train_model(model_v2_5, 'custom_cnn_v2_5', train_df, val_df)
plot_training_history(history_v2_5, 'Custom CNN v2_5')

# Train model v3
print("\nTraining Custom CNN v3...")
model_v3 = build_custom_cnn_v3()
model_v3.summary()
model_v3, history_v3 = train_model(model_v3, 'custom_cnn_v3', train_df, val_df)
plot_training_history(history_v3, 'Custom CNN v3')

# Train model v4
print("\nTraining Custom CNN v4...")
model_v4 = build_custom_cnn_v4()
model_v4.summary()
model_v4, history_v4 = train_model(model_v4, 'custom_cnn_v4', train_df, val_df)
plot_training_history(history_v4, 'Custom CNN v4')
```