

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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

**на тему: «Система класифікації та сегментації медичних зображень для
діагностування переломів»**

Виконав:

Студент IV курсу, групи КА-22

Панченко Дмитро Олександрович _____

Керівник:

Професор, д.т.н. Недашківська Надія Іванівна _____

Консультант з економічного розділу:

Доцент кафедри ЕК, к.е.н. Рощина Надія Василівна _____

Консультант з нормоконтролю:

Асистент кафедри ММСА, доктор філософії

Канцедал Георгій Олегович _____

Рецензент:

Доцент кафедри ШІ, к.т.н.

Шаповал Наталія Віталіївна _____

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

Студент _____

Київ – 2026 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
« ____ » _____ 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Панченку Дмитру Олександровичу

1. Тема роботи «Система класифікації та сегментації медичних зображень для діагностування переломів», керівник роботи професор, д.т.н., професор Недашківська Надія Іванівна, затверджені наказом по університету від 27.05.2026 р. № 1944-с
2. Термін подання студентом роботи «10» червня 2026 р.
3. Вихідні дані до роботи: медичні зображення переломів з анотаціями.
4. Зміст роботи: Дослідження предметної області діагностики переломів на основі рентгенівських знімків. Аналіз методів класифікації та сегментації медичних зображень. Розробка та навчання моделей класифікації і сегментації переломів. Реалізація програмної системи автоматизованої діагностики. Оцінка якості та ефективності розробленої системи.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): зображення наборів даних, ілюстрація створеної системи, презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання 18.02.2026

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми бакалаврської роботи.	18.02.2026-10.04.2026	виконано
2	Дослідження предметної області, огляд літератури, ознайомлення з ДСТУ.	10.04.2026-17.04.2026	виконано
3	Підготовка даних, робота над першим розділом.	17.04.2026-01.05.2026	виконано
4	Проектування архітектури системи, робота над другим розділом.	01.05.2026-10.05.2026	виконано
5	Програма реалізація розробленої системи.	10.05.2026-20.05.2026	виконано
6	Оцінка ефективності моделей, написання третього розділу.	20.05.2026-25.05.2026	виконано
7	Оформлення пояснювальної записки, підготовка графічних матеріалів та проходження нормоконтролю.	26.05.2026-02.06.2026	виконано
8	Підготовка презентації та доповіді.	03.06.2026-10.06.2026	виконано

Студент _____

Дмитро ПАНЧЕНКО

Керівник _____

Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота складається з 142 сторінок, містить 29 рисунків, 8 таблиць, 2 додатків та 20 джерел.

ПЕРЕЛОМИ КІСТОК, СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ, КЛАСИФІКАЦІЯ, РЕНТГЕНОГРАФІЯ, ГЛИБИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

Метою роботи є розробка двостадійного діагностичного пайплайну класифікація і сегментація з інтегрованим механізмом інтерпретованої локалізації на основі теплових карт активацій.

Об'єктом дослідження є процес автоматизованої діагностики переломів кісток на основі рентгенівських знімків.

Предметом дослідження є методи глибокого навчання для класифікації та піксельної сегментації зон переломів на рентгенографічних зображеннях.

У роботі реалізовано та досліджено три архітектури класифікації (ResNet18, ResNet50, ConvNeXtV2-Tiny) та п'ять архітектур сегментації (U-Net, U-Net++, DeepLabV3+, Attention DeepLabV3+, Attention U-Net++). Експерименти проводились на датасеті FracAtlas. Якість моделей оцінювалась за метриками Accuracy, F1, ROC-AUC (класифікація) та Dice Score, IoU (сегментація).

Результати роботи: найкращу класифікацію забезпечує ConvNeXtV2-Tiny. Серед сегментаційних архітектур найкращі показники демонструє Attention DeepLabV3+. Використання автоматичної САМ-локалізації підвищує Dice сегментації без жодного ручного втручання.

Подальший розвиток передбачає розширення навчальної вибірки, вдосконалення механізму локалізації та клінічну валідацію системи на даних із нових медичних закладів.

ABSTRACT

Diploma thesis consists of 142 pages, contains 29 figures, 8 tables, 2 appendices and 20 references.

BONE FRACTURES, IMAGE SEGMENTATION, CLASSIFICATION, RADIOGRAPHY, DEEP LEARNING, ARTIFICIAL INTELLIGENCE, CONVOLUTIONAL NEURAL NETWORKS

The aim of the work is to develop a two-stage diagnostic pipeline – classification and segmentation – with an integrated interpretable localization mechanism based on class activation maps.

The object of research is the process of automated bone fracture diagnosis based on X-ray images.

The subject of research is deep learning methods for classification and pixel-level segmentation of fracture zones in radiographic images.

The thesis implements and investigates three classification architectures (ResNet18, ResNet50, ConvNeXtV2-Tiny) and five segmentation architectures (U-Net, U-Net++, DeepLabV3+, Attention DeepLabV3+, Attention U-Net++). Experiments were conducted on the FracAtlas dataset. Model quality was evaluated using Accuracy, F1, ROC-AUC (classification) and Dice Score, IoU (segmentation).

Results: the best classification performance is achieved by ConvNeXtV2-Tiny. Among segmentation architectures, Attention DeepLabV3+ demonstrates the highest scores. The use of automatic CAM-based localization improves segmentation Dice without any manual annotation.

Further development includes expanding the training dataset, improving the localization mechanism, and clinical validation of the system on data from new medical institutions.

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Актуальність проблеми діагностики та автоматичної сегментації переломів кісток	10
1.2 Використання рентгенографії для первинної візуалізації переломів	12
1.3 Проблематика комп'ютерної обробки та інтерпретації рентгенівських знімків.....	13
1.4 Огляд наборів медичних даних (датасетів) для навчання моделей	14
1.5 Постановка задачі дослідження та вибір методів її розв'язання	15
1.5.1 Загальний опис діагностичного пайплайну	15
1.5.2 Математична постановка задачі	17
1.5.2.1 Формальний опис задачі класифікації зображень.....	18
1.5.2.2 Формальний опис задачі сегментації зображень	19
1.6 Висновки до розділу 1	20
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ТА МАТЕМАТИЧНИЙ АПАРАТ	22
2.1 Згорткові нейронні мережі як основа обробки зображень	22
2.1.1 Операція згортки та ієрархічне представлення ознак	23
2.1.2 Функції активації та регуляризація	24
2.2 Архітектури для класифікації зображень	26
2.2.1 ResNet18: архітектура та залишкове навчання.....	26
2.2.2 ResNet50: bottleneck-блоки та масштабування.....	28
2.2.3 ConvNeXtV2: сучасний підхід до згорткових архітектур	29
2.3 Архітектури для сегментації зображень	30
2.3.1 U-Net: енкодер-декодер з пропусковими з'єднаннями.....	31
2.3.2 U-Net++: щільна мережа вкладених з'єднань	32
2.3.3 DeepLabV3+: дилатовані згортки та ASPP	34
2.3.4 Механізм просторової уваги як розширення сегментаційних архітектур	35
2.3.5 Блок scSE як механізм уваги в архітектурах UNet++ та DeepLabV3+	37
2.4 Функції втрат для задач із дисбалансом класів.....	38
2.5 Трансферне навчання.....	40
2.6 Метрики оцінки якості.....	41

2.6.1 Метрики класифікації	41
2.6.2 Метрики сегментації: Dice Score та IoU.....	42
2.7 Методи інтерпретації рішень згорткових нейронних мереж.....	43
2.7.1 GradCAM та GradCAM++	43
2.7.2 LayerCAM.....	45
2.7 Висновки до розділу 2	45
РОЗДІЛ 3 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ	47
3.1 Збір та підготовка навчальних даних	47
3.1.1 Попередня обробка та стандартизація знімків	47
3.1.2 Застосування аугментації для підвищення варіативності навчальної вибірки	49
3.2 Тренування моделей класифікації	50
3.2.1 Стратегія покрокового розморожування	51
3.3 Аналіз якості моделей класифікації	51
3.3.1 Аналіз функцій втрат	51
3.3.2 Оцінки метрик якості	52
3.3.3 Візуалізація результатів класифікації	54
3.4 Тренування моделей сегментації.....	56
3.4.1 Базові архітектури сегментації.....	56
3.4.2 Архітектури з механізмом просторової уваги.....	57
3.5 Стратегії формування вхідних даних для сегментації	58
3.5.1 Навчання на повних рентгенівських знімках	58
3.5.2 Навчання на фрагментах із автоматичною локалізацією на основі LayerCAM.....	58
3.5.3 Навчання на фрагментах з еталонною локалізацією	59
3.6 Оцінка якості моделей сегментації.....	59
3.6.1 Оцінка функцій втрат	59
3.6.2 Аналіз метрик якості	61
3.6.3 Візуалізація результатів сегментації	62
3.7 Система класифікації та сегментації	63
3.7.1 Архітектура системи	64
3.7.2 Функціональна схема системи	65
3.7.3 Демонстрація роботи системи.....	66
3.8 Висновки до розділу 3	68

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	69
4.1 Постановка задачі проектування	70
4.2 Обґрунтування функцій програмного продукту	71
4.3 Обґрунтування системи параметрів програмного продукту	74
4.4 Аналіз експертного оцінювання параметрів	77
4.5 Аналіз рівня якості варіантів реалізації функцій	80
4.6 Економічний аналіз варіантів розробки ПП	81
4.7 Вибір кращого варіанту ПП за техніко-економічним рівнем	85
4.8 Висновки до розділу 4	86
ВИСНОВКИ	88
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	91
ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО МОДУЛЮ	94
ДОДАТОК В ГРАФІЧНІ МАТЕРІАЛИ	132

ВСТУП

Переломи кісток є одним із найпоширеніших видів травм у світі та становлять значне медичне й соціально-економічне навантаження на суспільство. За даними дослідження Global Burden of Disease 2019 щороку у світі реєструється близько 178 мільйонів нових випадків переломів, а загальна кількість людей, які живуть із наслідками цих травм, перевищує 445 мільйонів [1]. Попри незначне зниження показників захворюваності з 1990 по 2019 рік, абсолютна кількість переломів невинно зростає – передусім через старіння населення та збільшення частки людей, які страждають від остеопорозу [1, 2].

Золотим стандартом діагностики залишається рентгенографія. Проте точність ручної інтерпретації знімків є недосконалою: від 1,1% до 3,7% переломів пропускаються лікарями при першому зверненні (а в окремих групах пацієнтів цей показник сягає 37%), що безпосередньо призводить до помилок у подальшому лікуванні [3, 4].

Ключовим етапом у створенні інформаційних систем аналізу кісткових травм є точна локалізація пошкоджень на знімку – сегментація. Завдяки детальному окресленню меж травми алгоритми здатні ефективно визначати характер зламу, ступінь зміщення та загальну складність перелому.

Метою даної роботи є розв'язання задачі автоматичного виділення зон переломів на рентгенограмах. Сегментація розглядається не лише як засіб локалізації пошкоджень, а і як важливий етап підготовки даних для подальшої багатокласової класифікації. Основну увагу сфокусовано на пошуку оптимальних стратегій навчання моделей для досягнення максимальної точності виділення контурів травми. Це є критично важливим кроком до створення складніших багаторівневих систем діагностики переломів.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми діагностики та автоматичної сегментації переломів кісток

Переломи кісток належать до найпоширеніших і водночас клінічно найрізноманітніших травматичних пошкоджень людського організму. Їх медична значущість визначається не лише частотою виникнення, а й комплексом соціально-економічних наслідків: тривалою непрацездатністю, необхідністю реабілітації, а в разі ускладнень – стійкою інвалідністю. Особливо тривожними є демографічні прогнози: через глобальне старіння населення та поширення остеопорозу річна кількість лише переломів стегна до 2050 року майже подвоїться порівняно з показниками 2018 року. При цьому перелом стегна у літньому віці рідко є ізольованою травмою – він слугує тригером для каскаду ускладнень на тлі вже наявної травми, що й обумовлює загальну смертність протягом першого року після травми на рівні 14–28% залежно від країни та стану системи охорони здоров'я [1].

Переломи нижніх кінцівок і тазу є найпоширенішими та найтяжчими за своїми функціональними наслідками. Зокрема, переломи гомілки, малогомілкової кістки та щиколотки демонструють найвищий у світі показник захворюваності – 419,9 випадку на 100 тисяч населення [2]. Переломи передпліччя також становлять значний тягар: вони зустрічаються в усіх вікових групах, спричиняють суттєві соціально-економічні витрати через тривале лікування та довгострокову втрату функції верхньої кінцівки [2].

Основним інструментом первинної візуалізації кісткових пошкоджень залишається рентгенографія – доступна, швидка та економічно обґрунтована методика. Проте інтерпретація рентгенограм є складним когнітивним завданням, результат якого суттєво залежить від досвіду лікаря, часу доби та завантаженості відділення. Дослідження показують, що від 1,1% до 3,7%

переломів кінцівок не виявляються при першому зверненні до відділення невідкладної допомоги [3]. При цьому лише третина пропущених випадків пояснюється об'єктивною рентгенологічною непомітністю пошкодження – решта є помилками інтерпретації, тобто потенційно усуненими [3]. Найчастіше пропускають переломи стопи у дорослих (7,6%), коліна (6,3%) та ліктя (6,0%) [4]. Показово, що 49% діагностичних помилок припадає на вечірні та нічні години, коли доступність консультації лікаря обмежена, а когнітивне навантаження на людину максимальне. При цьому 86% таких помилок мають безпосередні наслідки для тактики лікування [5].

Відповіддю на цей системний виклик стала автоматизована діагностика на основі методів глибокого навчання. Результати незалежних клінічних випробувань останніх років демонструють, що системи на основі глибокого навчання здатні розпізнавати переломи з точністю, порівнянною з досвідченим лікарем, а їх інтеграція у робочий процес лікаря дозволяє суттєво скоротити кількість діагностичних пропусків – без збільшення хибнопозитивних результатів [6]. Деякі комерційні платформи пройшли офіційну сертифікацію: OsteoDetect та BoneView отримали схвалення U.S. Food and Drug Administration (FDA) – американського регулятора у сфері медичних пристроїв. У 2024 році National Institute for Health and Care Excellence (NICE) – британський орган з оцінки медичних технологій – рекомендував застосування AI-систем виявлення переломів у закладах невідкладної допомоги. Проте попри ці успіхи, ряд принципових викликів залишається невирішеним: упередженість навчальних датасетів, обмежена здатність моделей до узагальнення на дані з нових клінік, недостатня інтерпретованість рішень та відсутність переконливих доказів покращення кінцевих результатів лікування пацієнтів [6].

1.2 Використання рентгенографії для первинної візуалізації переломів

Рентгенографія залишається першою лінією інструментальної діагностики при підозрі на перелом кістки. Попри розвиток альтернативних модальностей – комп'ютерної томографії (КТ), магнітно-резонансної томографії (МРТ) та ультразвукового дослідження (УЗД) – саме рентгенівський знімок є стандартом первинного обстеження у відділеннях невідкладної допомоги, травматологічних пунктах і поліклініках по всьому світу [3]. Таке домінуюче положення обумовлено поєднанням трьох ключових характеристик: широкою доступністю обладнання, швидкістю отримання результату та відносно низькою порівняно з КТ дозою іонізуючого випромінювання для пацієнта.

З фізичної точки зору принцип рентгенографії базується на різному ступені поглинання Х-випромінювання тканинами організму: кісткова тканина, завдяки вмісту кальцію, поглинає значно більше фотонів, ніж м'які тканини, і тому відображається на знімку у вигляді світлих структур на темному фоні. Лінія злому являє собою ділянку з меншою щільністю – через порушення цілісності кортикального або губчастого шару – і проявляється як темна смуга або ділянка зниженої інтенсивності.

Стандартним протоколом рентгенологічного дослідження при травмах є виконання знімків мінімум у двох проекціях – як правило, прямій та боковій [4]. Це зумовлено тим, що деякі переломи, невидимі в одній проекції, чітко візуалізуються в іншій. Зокрема, переломи шийки стегна, компресійні переломи хребців або імпресійні переломи плечової кістки нерідко виявляються лише при ретельному аналізі кількох проекцій. У складних клінічних ситуаціях або при внутрішньосуглобових ушкодженнях може додатково призначатись КТ, яка забезпечує тривимірну реконструкцію та

детальну оцінку фрагментів, однак супроводжується значно вищим рівнем опромінення та є менш доступною в умовах невідкладної допомоги.

1.3 Проблематика комп'ютерної обробки та інтерпретації рентгенівських знімків

Незважаючи на широке впровадження цифрової рентгенографії та зростаючий інтерес до автоматизованого аналізу медичних зображень, саме специфіка рентгенівського знімка породжує принципові труднощі для нейронних мереж – і зокрема для задачі піксельної сегментації зони перелому.

Перша проблема – геометрична. Рентгенограма є проекцією тривимірної анатомічної структури на двовимірну площину, внаслідок чого кісткові та м'якотканинні структури накладаються одна на одну. Це особливо критично при переломах хребців та ребер, де лінія злому може бути повністю перекрита тінню сусідніх органів.

Друга проблема – низький контраст. Лінія злому нерідко являє собою тонку темну смугу шириною в кілька пікселів на сірому фоні, що принципово відрізняє цю задачу від, наприклад, сегментації пухлини на МРТ, де зона патології має чіткий контраст із навколишньою тканиною [6].

Третя проблема – морфологічна варіабельність. Форма та розміри зони перелому можуть кардинально відрізнятися від випадку до випадку: від мікротріщини до складного багатоосколкового ушкодження. Ця варіабельність суттєво ускладнює узагальнення, якого потребує будь-яка нейромережева модель.

Четверта проблема – дисбаланс класів. Пікселі зони злому становлять лічені відсотки від загальної площі знімка, тоді як переважна більшість відповідає нормальній кістковій тканині або фону. У датасеті FracAtlas,

наприклад, лише 717 із 4 083 зображень містять переломи – менше 18% [7]. Без спеціальних заходів (зважені функції втрат, аугментація, балансування вибірки) модель просто навчається ігнорувати рідкісний клас.

П'ята проблема – дефіцит анотованих даних. Розмітка рентгенівських знімків на рівні пікселів потребує залучення кількох незалежних радіологів для отримання консенсусної анотації – це надзвичайно трудомісткий і дорогий процес [7]. Саме тому публічних датасетів із якісною піксельною розміткою переломів досі катастрофічно мало, що суттєво обмежує можливості для навчання та об'єктивного порівняння моделей.

1.4 Огляд наборів медичних даних (датасетів) для навчання моделей

Якість та масштаб навчального датасету є визначальним фактором ефективності моделей глибокого навчання. У сфері автоматизованої діагностики переломів ця залежність особливо критична: медичні зображення потребують експертної анотації, а збір клінічних даних обмежений вимогами конфіденційності та деідентифікації.

FracAtlas – один із перших публічних датасетів із піксельною розміткою переломів, розроблений для задач класифікації, локалізації та сегментації. Він містить 4 083 рентгенограми з трьох лікарень Бангладешу, анотовані двома досвідченими радіологами та ортопедом [7]. З них 717 знімків містять 922 задокументовані переломи – кожен із власною піксельною маскою та bounding box – та 3 366 нормальних знімків. Знімки охоплюють три анатомічні ділянки: руку, ногу та стегно. Анотації зберігаються у форматі COCO JSON, що підтримує множинні фрагменти на одному знімку: в середньому 1.29 зони перелому на знімок. Це робить FracAtlas придатним для навчання сегментаційних архітектур, на відміну від

датасетів із виключно класифікаційними мітками [7]. Саме цей датасет є єдиною основою для всіх експериментів у роботі.

GRAZPEDWRI-DX – масштабний датасет, опублікований Медичним університетом Граца (Австрія) у 2022 році, що містить 20 327 рентгенівських знімків зап'ясть від 6 091 пацієнта дитячого віку. Попри значний обсяг та детальну анотацію датасет охоплює виключно травми зап'ястя і не містить піксельних масок сегментації, що обмежує його застосування задачами детекції та класифікації об'єктів.

Для підготовки навчальних даних полігональні маски COCO JSON конвертувались у бінарні PNG-маски розміром 256×256 пікселів, де 1 відповідає зоні перелому, 0 – фону. З огляду на дисбаланс між нормальними знімками (3 366) та знімками з переломами (717), для навчання відбирались виключно зображення з наявною маскою.

1.5 Постановка задачі дослідження та вибір методів її розв'язання

Задачу автоматизованої діагностики переломів кісток за рентгенівськими знімками можна формально розглядати як послідовність двох взаємопов'язаних підзадач машинного навчання, кожна з яких вирішується окремою моделлю та разом утворює єдиний діагностичний пайплайн.

1.5.1 Загальний опис діагностичного пайплайну

Перша підзадача – бінарна класифікація, яка полягає у визначенні факту наявності перелому на знімку. Вхідне зображення аналізується

моделлю, яка відносить його до одного з двох класів: «норма» або «перелом». При отриманні відповіді «норма» обробка завершується. При виявленні перелому додатково генеруються теплові карти методом LayerCAM [9], що дають грубу локалізацію зони ушкодження у вигляді bounding box..

Друга підзадача – сегментація, яка передбачає точне піксельне виділення зони перелому. На відміну від класифікації, що дає лише відповідь «є / немає», сегментація формує бінарну маску, де кожному пікселю присвоюється мітка належності до зони ушкодження. Результатом є точні межі пошкодженої ділянки, що є критично важливим для клінічного аналізу.

У рамках дослідження розглядаються три підходи до формування вхідних даних для моделі сегментації: подача повного рентгенівського знімка без будь-якої попередньої локалізації; подача фрагмента, вирізаного за обмежувальною рамкою, отриманою автоматично за допомогою LayerCAM; та подача фрагмента, вирізаного за еталонними анотованими рамками з датасету FracAtlas. Перший підхід слугує базовою лінією, другий відображає реалістичний сценарій автоматичної локалізації, третій – є верхньою межею точності, оскільки використовує ідеальну розмітку, недоступну в реальних умовах. Порівняння цих трьох стратегій є одним із ключових завдань роботи.

Для кількісної оцінки якості розроблених моделей використовується стандартний набір метрик. Для задачі класифікації розраховуються Accuracy, Precision, Recall, F1-score та ROC-AUC. Для задачі сегментації основними метриками є Dice Score та Intersection over Union (IoU), що безпосередньо вимірюють якість піксельного збігу передбаченої маски з еталонною розміткою.

Надійна оцінка моделей забезпечується поділом датасету на три незалежні підмножини: навчальну (train), валідаційну (val) та тестову (test). Валідаційна вибірка використовується для контролю перенавчання та ранньої зупинки навчання (early stopping), тестова – відкладається до завершення навчання та використовується виключно для фінального вимірювання

метрик, що відображає реальну здатність моделі узагальнюватися на нові дані.

1.5.2 Математична постановка задачі

Задачу автоматизованої діагностики переломів кісток за рентгенівськими знімками формально можна подати як послідовність двох взаємопов'язаних підзадач машинного навчання: бінарної класифікації та піксельної сегментації.

Введемо простір вхідних даних $\mathcal{X} \subset \mathbb{R}^{H \times W \times C}$, елементами, якого є рентгенівські знімки зображення розміру $H \times W$ пікселів з C каналами. Після застосування препроцесингу CLANE Grayscale маємо $C = 1, H = W = 224$

Кожному зображенню $x \in \mathcal{X}$ відповідає трійка анотацій (y, M, B) , де:

- 1) $y \in \{0, 1\}$ – глобальна мітка класу (0 – норма, 1 – перелом);
- 2) $M \in \{0, 1\}^{H \times W}$ – бінарна маска сегментації, де $M_{ij} = 1$ означає належність пікселя $\{i, j\}$ до зони перелому;
- 3) $B = (x_{min}, y_{min}, x_{max}, y_{max}) \in \mathbb{R}^4$ – обмежувальна рамка переломної зони.

Навчальна вибірка визначається як $\mathcal{D} = \{(x_i, y_i, M_i, B_i)\}$ та розбивається на три підмножини відповідно до стандартної практики оцінки моделей машинного навчання:

$$\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test}, \quad \mathcal{D}_{train} \cap \mathcal{D}_{val} \cap \mathcal{D}_{test} = \emptyset$$

Підмножина $\mathcal{D}_{val}$ використовується для моніторингу перенавчання та ранньої зупинки навчання, $\mathcal{D}_{test}$ відкладається до завершення навчання і використовується виключно для фінального вимірювання метрик.

Загальний діагностичний пайплайн формалізується як двоетапне відображення:

$$x \xrightarrow{f_{cls}} \hat{y} \xrightarrow{[\hat{y}=1]} x' \xrightarrow{g_\varphi} M',$$

де $f_{cls}: \mathcal{X} \rightarrow [0, 1]$ – класифікатор із параметрами θ_{cls} ;
 x' – локалізований фрагмент зображення;
 $g_\varphi: \mathcal{X}' \rightarrow \{0, 1\}^{H' \times W'}$ – сегментатор з параметрами φ .

1.5.2.1 Формальний опис задачі класифікації зображень

Задача бінарної класифікації полягає у відображенні простору вхідних зображень $\mathcal{X}$ у простір міток класів $\mathcal{Y}$. Математично класифікатор описується як параметризована функція $f_{cls}: \mathcal{X} \rightarrow [0, 1]$, яка для заданого зображення x повертає скалярну ймовірність $\hat{y}$ належності до позитивного класу (класу «перелом»):

$$\hat{y}_i = f_{cls}(x_i, \theta_{cls}),$$

де θ_{cls} – вектор параметрів(ваг) згорткової нейронної мережі.

Процес навчання моделі класифікації зводиться до задачі мінімізації функції втрат на тренувальній вибірці $\mathcal{D}_{train}$. Для задачі бінарної класифікації використовується бінарна крос-ентропія, що позначається як H :

$$H(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Метою оптимізації є знаходження таких параметрів θ_{cls}^* , при яких функція втрат $H(y, \hat{y})$ досягає мінімуму. Отримане значення $\hat{y}_i$ конвертується у кінцеве дискретне рішення щодо класу на основі порогу відсікання τ (зазвичай $\tau = 0,5$).

$$\theta_{cls}^* = \underset{\theta_{cls}}{argmin} \frac{1}{|\mathcal{D}_{train}|} \sum_{(x_i, y_i) \in \mathcal{D}_{train}} H_{cls}(f_{cls}(x_i, \theta_{cls}), y_i)$$

Вихідне передбачення класу визначається пороговуванням:

$$C_i = 1[f_{cls}(x_i, \theta_{cls}) \geq \tau], \quad \tau \in [0, 1]$$

При $C_i = 0$ пайплайн завершує обробку; при $C_i = 1$ додатково генерується карта просторової уваги $\mathcal{A} \in \mathbb{R}^{H' \times W'}$ методом LayerCAM:

$$\mathcal{A}(x) = ReLU \left(\sum_k \omega_k * F_k \right),$$

де F_k — карта активацій k -го каналу останнього загорткового шару, ω_k — відповідні вагові коефіцієнти. На основі $\mathcal{A}(x)$ формується обмежувальна рамка:

$$B_{CAM} = arg \min_b \{b: \mathcal{A}(x)|_b \geq \alpha * \max \mathcal{A}(x)\},$$

де α — поріг відсікання фону.

1.5.2.2 Формальний опис задачі сегментації зображень

Задача піксельної сегментації полягає у знаходженні оптимальних параметрів Φ_{seg}^* моделі g_Φ , що мінімізують емпіричний ризик:

$$\Phi_{seg}^* = argmin_{\Phi} \frac{1}{|\mathcal{D}_{train}|} \sum_{(x_i', M_i) \in \mathcal{D}_{train}} H_{seg}(g_\Phi(x_i'), M_i),$$

де x_i' — вхідний параметр згідно з обраною стратегією локалізації, H_{seg} — функція втрат для сегментації.

Для задачі сегментації використовується комбінована функція втрат:

$$H_{seg} = \omega_1 * H_{BCE} + \omega_2 * H_{Dice} + \omega_3 * H_{Tversky}$$

з ваговими коефіцієнтами $\omega_1 = \omega_2 = \omega_3 = 0,35$.

Вхідний простір $\mathcal{X}'$ визначається однією з трьох стратегій:

$$x' = \begin{cases} x & \text{— повний знімок} \\ x|_{B_{CAM}} & \text{— фрагмент за CAM рамкою} \\ x|_{B_{GT}} & \text{— фрагмент за еталонною рамкою} \end{cases}$$

Передбачена маска отримується пороговуванням виходу моделі:

$$M'_{ij} = 1[g_\Phi(x')_{ij} \geq 0,5], \quad i \in [1, H], j \in [1, W]$$

1.6 Висновки до розділу 1

Проведений аналіз сучасного стану проблеми автоматизованої діагностики переломів кісток підтвердив, що рентгенографія залишається незамінним і першочерговим інструментом візуалізації кісткових травм, забезпечуючи оптимальний баланс між інформативністю, швидкістю та доступністю в умовах відділення невідкладної допомоги. Водночас огляд клінічних досліджень виявив системну проблему, яка не може бути вирішена виключно організаційними заходами: від 1.1% до 3.7% переломів стабільно пропускаються при першому зверненні пацієнта, причому більшість таких помилок концентрується у вечірньо-нічний час і безпосередньо впливає на тактику лікування. Це свідчить про те, що людський фактор є структурною вразливістю існуючої системи діагностики, а не випадковим відхиленням, – і саме тут автоматизовані системи підтримки прийняття рішень мають найбільший потенціал клінічного впливу.

Детальний розбір специфіки рентгенівського зображення з точки зору комп'ютерного аналізу дозволив виокремити п'ять принципових викликів, що відрізняють задачу сегментації переломів від стандартних задач комп'ютерного зору: проекційне накладання анатомічних структур, вкрай низький піксельний контраст лінії злому, висока морфологічна варіабельність форми ушкодження, глибокий дисбаланс класів на рівні пікселів та хронічний дефіцит публічних датасетів із якісною піксельною розміткою. Сукупність цих факторів пояснює, чому більшість існуючих підходів обмежуються або бінарним виявленням перелому, або локалізацією за допомогою обмежувальних рамок – і лише одиниці досягають рівня повноцінної піксельної сегментації з клінічно прийнятною точністю.

Аналіз доступних публічних датасетів засвідчив критичну нерівномірність їх покриття: при наявності великих ресурсів для класифікації та детекції (GRAZPEDWRI-DX, 20 327 знімків), датасетів із піксельною

розміткою переломів фактично не існує у публічному доступі – за винятком FracAtlas, що містить лише 717 анотованих випадків із піксельною розміткою. Саме цей датасет обрано єдиною основою для проведення дослідження, що формує фундаментальне обмеження за обсягом даних і вказує на необхідність розробки підходів, ефективних в умовах малих вибірок.

Таким чином, узагальнення матеріалів наукових джерел дозволило сформулювати два ключові напрями, на яких зосереджено дане дослідження: по-перше, побудова ефективного сегментатора зони перелому в умовах обмежених анотованих даних із застосуванням transfer learning та двостадійного навчання; по-друге, інтеграція механізму інтерпретованої локалізації на основі LayerSAM як проміжного етапу між класифікацією та сегментацією – що наближає систему до практично застосовного інструменту підтримки клінічних рішень.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ТА МАТЕМАТИЧНИЙ АПАРАТ

2.1 Згорткові нейронні мережі як основа обробки зображень

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є домінуючим підходом до аналізу зображень у задачах комп'ютерного зору завдяки здатності автоматично виявляти просторові ієрархічні ознаки безпосередньо з сирих пікселів. Архітектура CNN була вперше запропонована у 1998 році для розпізнавання рукописних цифр і з того часу стала основою переважної більшості систем комп'ютерного зору [8].

На відміну від класичних нейронних мереж із повним з'єднанням (Fully Connected Networks), де кожен нейрон з'єднаний з усіма нейронами попереднього шару, CNN використовують три ключові принципи. Локальна зв'язність – кожен нейрон обробляє лише невелику просторову ділянку вхідного зображення, що відповідає біологічній організації зорової кори людини. Розділення ваг – один і той самий фільтр застосовується до всіх позицій зображення, що різко скорочує кількість параметрів і забезпечує інваріантність до зсувів об'єкта в кадрі. Ієрархічність – послідовне стекування шарів дозволяє будувати дедалі складніші уявлення про вхідні дані: від простих країв до складних семантичних структур [8].

У контексті даної роботи CNN слугують основою для всіх розроблених моделей: класифікаційних (ResNet18, ResNet50, ConvNeXtV2-Tiny) та сегментаційних (U-Net, U-Net++, DeepLabV3+). Вибір саме згорткових архітектур обумовлений їхньою доведеною ефективністю на медичних зображеннях, де просторові структури мають чіткі локальні характеристики, що добре вловлюються операцією згортки.

2.1.1 Операція згортки та ієрархічне представлення ознак

Основним обчислювальним блоком CNN є операція дискретної двовимірної згортки. Для вхідного зображення $x \in \mathbb{R}^{H \times W}$ та фільтра (ядра) $k \in \mathbb{R}^{f \times f}$ вихідна карта ознак $\mathcal{F} \in \mathbb{R}^{H' \times W'}$ визначається як:

$$\mathcal{F}_{ij} = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} x_{(i+m)(j+n)} * k_{mn} + b,$$

Де b – зміщення(bias);

f – розмір ядра;

ij – просторові координати вихідної карти.

На практиці мережа містить K паралельних фільтрів, кожен з яких виявляє свій тип просторових патернів, формуючи K карт ознак. На рисунку 2.1 можемо бачити схему операції згортки.

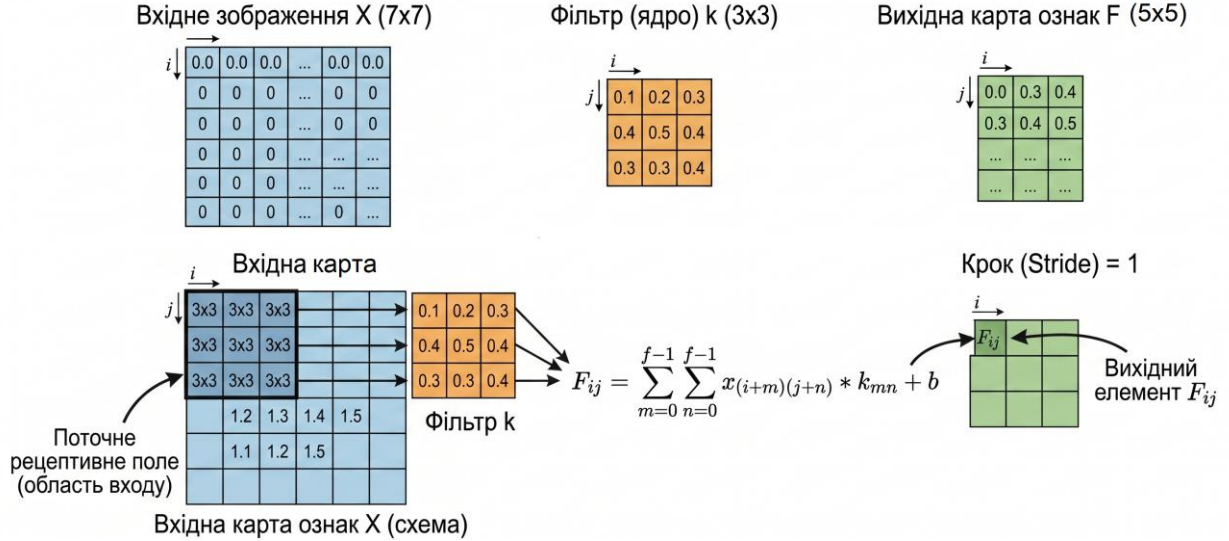


Рисунок 2.1 – Схема операції згортки (адаптовано з [8])

Після кожного згорткового шару зазвичай застосовується операція субдискретизації – max pooling або average pooling, яка зменшує просторову роздільність карт ознак:

$$\mathcal{P}_{ij} = \max_{(m,n) \in \mathcal{R}_{ij}} \mathcal{F}_{mn},$$

де $\mathcal{R}_{ij}$ – локальне вікно розміром $p \times p$ навколо позиції (i, j) . Зменшення просторових розмірів збільшує рецептивне поле – ділянку вхідного зображення, яку бачить нейрон у глибших шарах мережі.

Ключова властивість CNN – ієрархічне представлення ознак. Ранні шари виявляють низькорівневі ознаки (краї, градієнти яскравості, текстури), середні – форми та локальні структури, глибокі – високорівневі семантичні концепції (кістка, суглоб, зона ушкодження). Саме ця ієрархія робить CNN особливо ефективними для медичних зображень, де діагностично значущі структури представлені на різних масштабах.

2.1.2 Функції активації та регуляризація

Після кожного згорткового шару застосовується поелементна нелінійна функція активації, що вводить до мережі нелінійність, необхідну для апроксимації складних залежностей між вхідними даними та цільовими мітками. Без функцій активації послідовність лінійних перетворень зводилася б до єдиного лінійного відображення незалежно від глибини мережі.

Найпоширенішою функцією активації у сучасних CNN є ReLU (Rectified Linear Unit) [8]:

$$ReLU(x) = \max(0, x)$$

Її ключові переваги – обчислювальна простота та відсутність проблеми загасання градієнтів для позитивних активацій, що дозволяє ефективно навчати глибокі мережі. Проте ReLU має недолік «мертвих нейронів»: якщо активація нейрона стає від'ємною, градієнт через нього дорівнює нулю і нейрон перестає навчатися. Для усунення цього недоліку запропоновано модифікації – Leaky ReLU та GELU (Gaussian Error Linear Unit), яка використовується у трансформерних архітектурах [11]:

$$GELU(x) = x * \Phi(x),$$

де $\Phi(x)$ – функція стандартного нормального розподілу. GELU забезпечує плавніший градієнт і демонструє кращу збіжність на задачах із медичними зображеннями.

Регуляризація спрямована на запобігання перенавчанню – ситуації, коли модель добре апроксимує тренувальні дані, але погано узагальнюється на нових прикладах. У CNN використовують кілька підходів.

Dropout під час навчання випадково обнуляє частку активацій із імовірністю p , змушуючи мережу формувати надлишкові та незалежні представлення ознак [8]:

$$\tilde{h} = h \odot m, \quad m_i \sim \text{Bernoulli}(1 - p),$$

де $\odot$ – по елементне множення.

Batch Normalization нормалізує активації всередині міні-батчу, що стабілізує та прискорює навчання [8]:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} * \gamma + \beta,$$

де μ_B, σ_B^2 – середнє і дисперсія по батчу, γ, β – адаптивні параметри масштабування та зсуву. Batch Normalization також виконує неявну регуляризацію, зменшуючи залежність між шарами мережі та дозволяючи використовувати вищі значення швидкості навчання.

L2-регуляризація (weight decay) додає штрафний член до функції втрат [8]:

$$\mathcal{L}_{reg} = \mathcal{L} + \lambda \sum_{\omega} w^2,$$

де λ – коефіцієнт регуляризації. Це стримує ваги від надмірно великих значень і зменшує складність моделі. У контексті даної роботи Batch Normalization та Dropout використовуються у всіх розроблених архітектурах, що є стандартною практикою для медичних задач з обмеженим обсягом навчальних даних.

2.2 Архітектури для класифікації зображень

У даній роботі для задачі класифікації рентгенівських знімків досліджуються три архітектури: ResNet18, ResNet50 та ConvNeXtV2-Tiny. Вибір моделей обумовлений необхідністю охопити різні точки на шкалі «складність – продуктивність»: від легкої та швидкої мережі до глибшої архітектури з сучасними структурними рішеннями. Усі три моделі використовуються із попередньо навченими вагами та адаптуються до цільової задачі за допомогою transfer learning.

2.2.1 ResNet18: архітектура та залишкове навчання

Сімейство ResNet (Residual Networks) було запропоновано у 2016 році дослідниками Microsoft[10] – і це стало фундаментальним проривом у навчанні глибоких нейронних мереж. До появи ResNet збільшення глибини мережі призводило до парадоксального погіршення точності – не через перенавчання, а через деградацію навчання: градієнти загасали або вибухали, проходячи через десятки шарів. Ключова ідея авторів полягала у введенні залишкових з'єднань (skip connections, shortcut connections), які дозволяють градієнту проходити безпосередньо через шари без перетворень.

Формально, замість того щоб навчати пряме відображення $\mathcal{H}(x)$ мережа навчає залишкову функцію:

$$\mathcal{F}(x) = \mathcal{H}(x) - x$$

а вихід блоку визначається як:

$$y = \mathcal{F}(x, \{W_i\}) + x,$$

де x – вхід блоку, $\mathcal{F}$ – залишкове відображення, що реалізується двома або трьома згортковими шарами. Якщо оптимальним є тотожне відображення,

мережі достатньо звести $\mathcal{F}(x)$ до нуля, що значно простіше, ніж апроксимувати тотожність безпосередньо. На рисунку 2.2 зображено приклад схеми залишкового блоку.

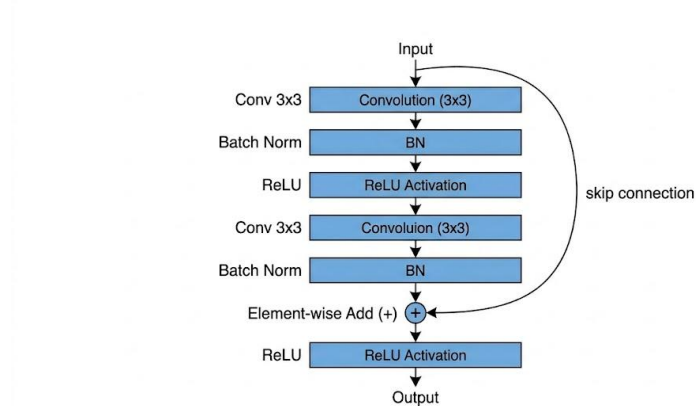


Рисунок 2.2 – Схема залишкового блоку ResNet (адаптовано з [10])

ResNet18 є найлегшою моделлю сімейства ResNet. Архітектура складається з початкового згорткового шару 7×7 з кроком 2, блоку max pooling, чотирьох груп залишкових блоків та фінального шару глобального average pooling з повнозв'язним класифікатором.

Кількість каналів поступово збільшується: $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$, тоді як просторова роздільність зменшується вдвічі на кожній групі. Загальна кількість шарів – 18 (звідси назва), параметрів – близько 11,7 мільйонів.

У даній роботі ResNet18 використовується з попередньо навченими вагами ImageNet (transfer learning). Заморожуються всі шари окрім layer4 та голови класифікатора. Така стратегія дозволяє зберегти низькорівневі та середньорівневі ознаки, навчені на великому датасеті, та адаптувати мережу до специфіки рентгенівських знімків при обмеженій кількості навчальних прикладів.

2.2.2 ResNet50: bottleneck-блоки та масштабування

ResNet50 належить до того ж сімейства залишкових мереж [10], проте відрізняється від ResNet18 архітектурою базового будівельного блоку. Замість basic block використовується bottleneck block – трьохшарова конструкція, яка значно ефективніша за рахунок зменшення обчислювальної вартості.

Перша згортка 1×1 стискає кількість каналів у 4 рази (наприклад, з 256 до 64), згортка 3×3 виконує просторову обробку на зменшеному просторі ознак, а фінальна 1×1 розширює канали назад. Такий підхід дозволяє значно збільшити глибину мережі без пропорційного зростання обчислювальної складності. Порівняння структури basic block та bottleneck block із зазначенням розмірностей тензорів наведено на рисунку 2.3.

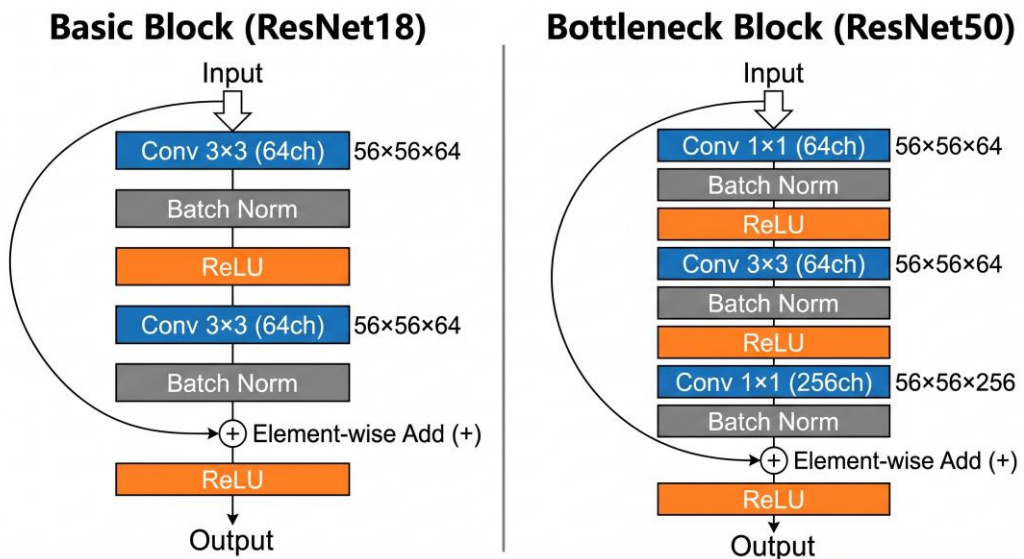


Рисунок 2.3 – Порівняння basic block та bottleneck block (адаптовано з [10])

ResNet50 містить 50 шарів та близько 25,6 млн параметрів. Структура чотирьох груп блоків: 3-4-6-3 bottleneck-блоки відповідно, із тією ж схемою розширення каналів $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$ (але з фактичними розмірностями $256 \rightarrow 512 \rightarrow 1024 \rightarrow 2048$ після розширення bottleneck). Глибша архітектура дозволяє формувати більш абстрактні та семантично

насичені представлення ознак, що особливо важливо для складних медичних патернів.

Порівняно з ResNet18, ResNet50 демонструє вищу точність на задачах із тонкими морфологічними відмінностями між класами, однак потребує більше обчислювальних ресурсів та більшого обсягу навчальних даних для ефективної адаптації. У даній роботі застосовується аналогічна стратегія transfer learning: заморожений backbone із розморожуванням layer4 та заміненою головою класифікатора.

2.2.3 ConvNeXtV2: сучасний підхід до згорткових архітектур

ConvNeXt був запропонований у 2022 році як спроба систематично «модернізувати» класичну ResNet-архітектуру, запозичивши ключові дизайнерські рішення з трансформерів (Swin Transformer), але залишившись у межах чистої CNN-парадигми [11]. ConvNeXtV2 – це вдосконалена версія 2023 року, яка додає механізм Global Response Normalization (GRN) та схему самонавчання FCMAE (Fully Convolutional Masked Autoencoders) [12].

Основні архітектурні зміни порівняно з ResNet:

- 1) заміна традиційного 7×7 stem-блоку на агресивний patchify-шар (згортка 4×4 з кроком 4), аналогічний до patch embedding у ViT;
- 2) використання depthwise-згортки 7×7 замість стандартної 3×3 , що збільшує рецептивне поле без суттєвого росту параметрів;
- 3) заміна BatchNorm на LayerNorm, стабільнішу при малих батчах та задачах із розподіленням, відмінним від ImageNet;
- 4) функція активації GELU замість ReLU – забезпечує плавніший градієнт завдяки стохастичному зважуванню активацій;
- 5) блок з інвертованим вузьким місцем (inverted bottleneck): розширення каналів у 4 рази всередині блоку, а не звуження як у ResNet.

Схему блоку ConvNeXtV2 із позначенням усіх операцій та skip connection наведено на рисунку 2.4.

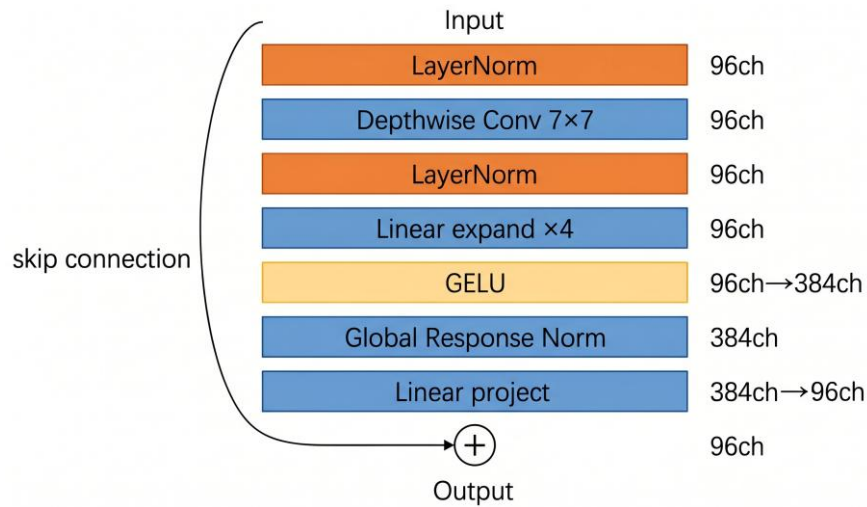


Рисунок 2.4 – Схема блоку ConvNeXtV2 (адаптовано з [12])

Варіант ConvNeXtV2-Tiny містить близько 28,6 млн параметрів і складається з чотирьох стадій із глибинами 3-3-9-3 блоки. Незважаючи на порівнянну кількість параметрів із ResNet50, архітектура демонструє вищу точність завдяки більшому рецептивному полю та ефективнішому використанню каналів. У контексті даної роботи ConvNeXtV2-Tiny використовується з вагами, попередньо навченими за схемою FCMAE на ImageNet-1K, що забезпечує сильніший пріор для медичних текстур порівняно із класичним supervised pretraining.

2.3 Архітектури для сегментації зображень

У даній роботі для задачі сегментації рентгенівських знімків досліджуються три архітектури: U-Net, U-Net++ та DeepLabV3+. На відміну від класифікаційних моделей, які повертають єдину мітку для всього зображення, сегментаційні мережі формують попиксельну маску – що є

критично важливим для точної локалізації зон ушкодження кісткових структур. Вибір архітектур охоплює як класичний енкодер-декодер підхід, так і сучасніші рішення з розширеною просторовою обробкою.

2.3.1 U-Net: енкодер-декодер з пропускними з'єднаннями

U-Net була запропонована у 2015 році спеціально для сегментації біомедичних зображень і стала де-факто стандартом у медичному комп'ютерному зорі [13]. Ключова проблема, яку вирішує U-Net – необхідність одночасно зберігати високорівневий семантичний контекст та низькорівневі просторові деталі. Класичні архітектури втрачають просторову інформацію при субдискретизації, що призводить до розмитих меж на вихідній масці. U-Net вирішує цю проблему через пропускні з'єднання (skip connections) між симетричними шарами енкодера та декодера [13].

Архітектура складається з трьох частин. Енкодер послідовно зменшує просторову роздільність через блоки згортки та max pooling, збільшуючи кількість каналів. На кожному рівні енкодера формується feature map, яка зберігається для подальшого використання. Bottleneck – найглибший рівень мережі без субдискретизації, де представлення має найвищу семантичну насиченість та найменшу просторову роздільність. Декодер (розширювальний шлях) відновлює просторову роздільність через транспоновані згортки (up-convolution) та на кожному рівні конкатенує feature maps із відповідного рівня енкодера [13]. Загальну схему архітектури U-Net наведено на рисунку 2.5.

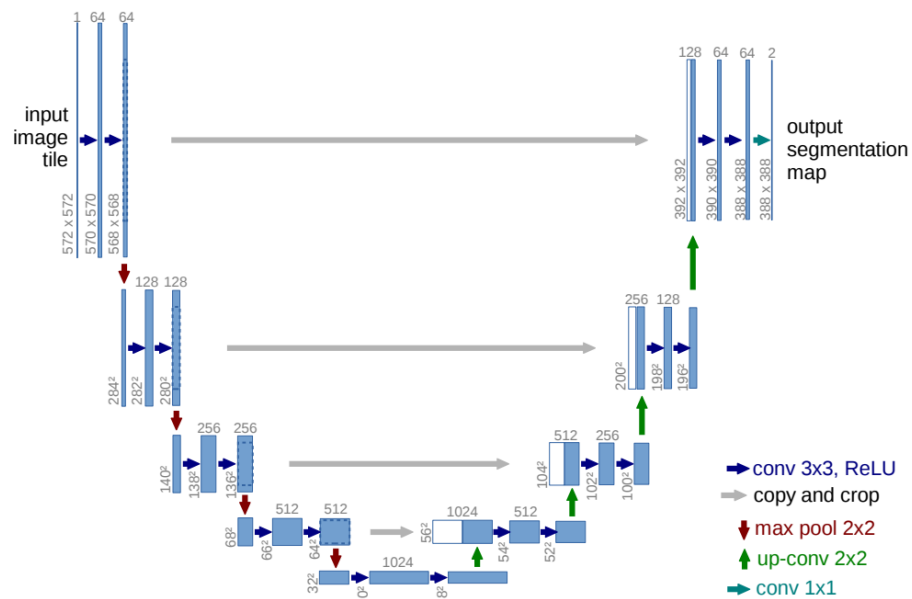


Рисунок 2.5 – Схема архітектури U-Net [13]

Фінальний шар мережі – згортка 1×1 , яка проектує feature maps на кількість цільових класів із подальшою функцією Sigmoid (бінарна сегментація) або Softmax (багатокласова). Оригінальна U-Net не використовує Batch Normalization та попередньо навчені ваги, проте у сучасних реалізаціях обидва покращення є стандартними. У даній роботі як енкодер використовується попередньо навчений ResNet-backbone із ImageNet, що значно прискорює збіжність при обмеженій кількості розмічених рентгеновських знімків.

2.3.2 U-Net++: щільна мережа вкладених з'єднань

U-Net++ була запропонована у 2018 році як розвиток класичної U-Net, спрямований на усунення її ключового недоліку – семантичного розриву між feature maps енкодера та декодера [14]. У класичній U-Net skip connection напряду передає feature map з енкодера до декодера, однак ці два представлення формуються на різних рівнях абстракції: енкодер бачить

низькорівневі деталі, тоді як декодер на тому ж рівні вже обробляє більш семантично насичені ознаки. Конкатенація таких різнорідних представлень є субоптимальною [14].

Ключова ідея U-Net++ – замінити прямі skip connections щільними блоками вкладених згорток (dense nested convolutions), які поступово заповнюють семантичний розрив між енкодером та декодером. Між кожною парою відповідних рівнів вводяться проміжні вузли $x^{i,j}$, де i – глибина рівня, j – крок у щільному блоці:

$$x^{i,j} = \mathcal{H}(x^{i-1,j}, \text{UpConv}(x^{i,j-1})), \quad j \geq 1$$

$$x^{i,0} = \mathcal{H}(x^{i-1,0}),$$

де $\mathcal{H}$ – блок із двох згорток 3×3 з BN та ReLU, $x^{i-1,j}$ – конкатенація всіх попередніх вузлів на тому ж рівні щільності. Таким чином, кожен вузол декодера отримує інформацію не лише з одного рівня енкодера, а з цілої послідовності проміжних перетворень [14]. Схему архітектури U-Net++ із позначенням вкладених вузлів та щільних з'єднань наведено на рисунку 2.6.

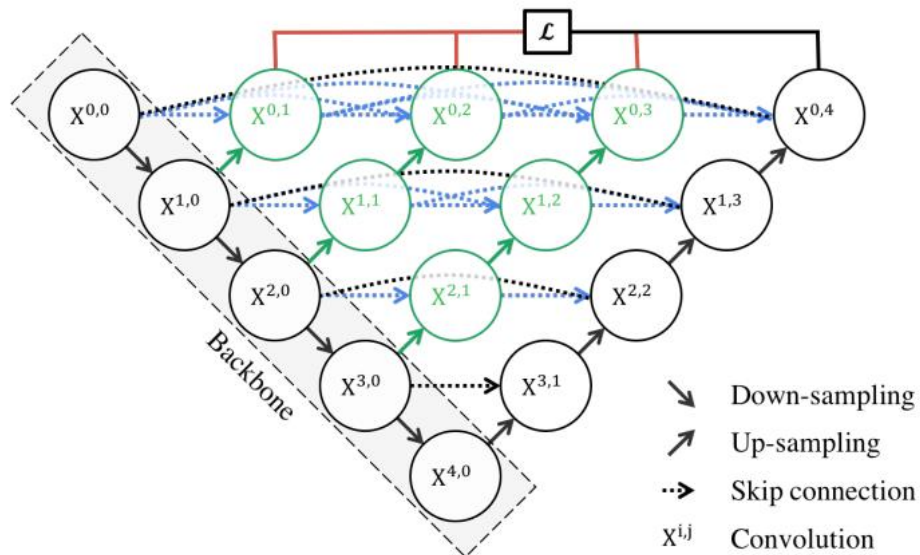


Рисунок 2.6 – Схема архітектури U-Net++ [14]

Додатковою перевагою архітектури є можливість глибокого нагляду (deep supervision): на кожному рівні декодера формується окрема вихідна маска, яка також враховується у функції втрат під час навчання [14]:

$$\mathcal{L} = \frac{1}{L} \sum_{l=1}^L \mathcal{L}_{seg}(\hat{y}^l, y)$$

де L – кількість рівнів декодера, $\hat{y}^l$ – передбачена маска на l -му рівні. Це змушує проміжні представлення бути семантично значущими на кожному масштабі, що особливо важливо для медичних знімків із дрібними структурами.

Під час інференсу U-Net++ може працювати у двох режимах: повна мережа (усі вкладені вузли активні) або обрізана версія, де використовуються лише вузли до певного рівня щільності – що дозволяє гнучко балансувати між точністю та швидкістю. У даній роботі використовується повна версія архітектури з ResNet-backbone як енкодером та глибоким наглядом під час навчання.

2.3.3 DeepLabV3+: дилатовані згортки та ASPP

DeepLabV3+ була запропонована у 2018 році як вдосконалення серії архітектур DeepLab, що принципово відрізняється від U-Net та U-Net++ підходом до збереження просторової роздільності [15]. Замість симетричного енкодер-декодера з багаторівневими skip connections, DeepLabV3+ використовує два ключові механізми: дилатовані згортки для збільшення рецептивного поля без втрати роздільності та Atrous Spatial Pyramid Pooling (ASPP) для захоплення багатомасштабного контексту [15].

Дилатована (atrous) згортка вводить коефіцієнт розширення r , який визначає відстань між елементами ядра:

$$\mathcal{F}_{ij} = \sum_{m=0}^{f-1} \sum_{n=0}^{f-1} x_{(i+r*m)(j+r*n)} * k_{mn}$$

При $r = 1$ отримуємо стандартну згортку, при $r = 2$ – ядро 3×3 охоплює область 5×5 , при $r = 4$ – область 9×9 , не збільшуючи кількість параметрів і зберігаючи повну просторову роздільність feature maps. Це принципова перевага над підходом max pooling + up-convolution, де частина просторової інформації безповоротно втрачається.

Модуль ASPP паралельно застосовує кілька дилатованих згорток із різними коефіцієнтами розширення, глобальний average pooling та звичайну згортку 1×1 , після чого конкатенує результати. такий підхід дозволяє одночасно аналізувати об'єкти різного масштабу – від дрібних деталей (малий r) до великих анатомічних структур (великий r), що особливо важливо для рентгенівських знімків, де зони ушкодження можуть мати різний розмір [15].

Декодер у DeepLabV3+ навмисно спрощений порівняно з U-Net: вихід ASPP білінійно масштабується у 4 рази, після чого конкатенується лише з одним low-level feature map із енкодера (після проєкції згорткою 1×1 для зменшення кількості каналів), і обробляється двома згортками 3×3 .

У даній роботі як backbone DeepLabV3+ використовується попередньо навчений ResNet із atrous згортками у фінальних блоках (output stride = 16), що забезпечує оптимальний баланс між розміром рецептивного поля та обчислювальною ефективністю при обробці рентгенівських знімків кісткових структур.

2.3.4 Механізм просторової уваги як розширення сегментаційних архітектур

Механізми уваги були запозичені у згортковій архітектурі з трансформерів і дозволяють мережі динамічно зосереджуватися на діагностично значущих ділянках зображення, пригнічуючи нерелевантний

фон. У контексті сегментації медичних зображень це особливо важливо: зона ушкодження кістки може займати незначну частину знімка, і без механізму уваги мережа витрачає однакову обчислювальну потужність на обробку як патологічних, так і нормальних тканин [16].

Просторова увага (spatial attention) обчислює карту вагів $\mathcal{A} \in [0, 1]^{H \times W}$, яка визначає відносну важливість кожної просторової позиції feature map. Агрегація по каналах виконується паралельно двома способами, результати конкатенуються та обробляються згорткою із сигмоїдою [17]:

$$[AvgPool(F) \oplus MaxPool(F)] \rightarrow Conv_{7 \times 7} \rightarrow \sigma \rightarrow \mathcal{A}$$

Після чого карта уваги поелементно множиться на вхідний feature map:

$$F \rightarrow \mathcal{A} \otimes F \rightarrow \tilde{F},$$

де $\oplus$ – конкатенація по каналах;

σ – сигмоїда;

$\otimes$ – поелементне множення.

Позиції з високою вагою підсилюються, нерелевантні пригнічуються, без зміни розмірності тензора. Схему модуля просторової уваги наведено на рисунку 2.7.

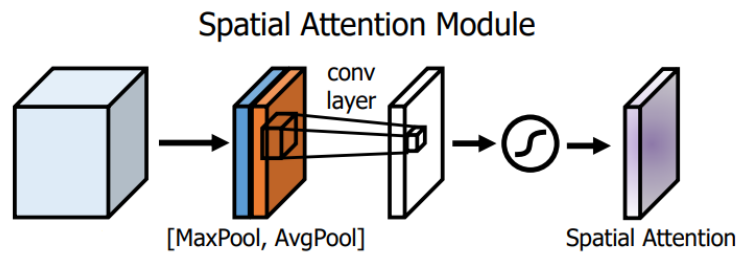


Рисунок 2.7 – Схема модуля просторової уваги [17]

У сегментаційних архітектурах модулі уваги найчастіше вбудовуються у двох місцях. По-перше, в skip connections – перед конкатенацією feature maps енкодера та декодера, що дозволяє відфільтрувати нерелевантні просторові позиції до їх передачі в декодер. Такий підхід реалізований в Attention U-Net [16], де кожне skip connection замінюється блоком уваги:

$$F_{att} = \mathcal{A}(F_{enc}, F_{dec}) \otimes F_{enc}$$

де $\mathcal{A}(F_{enc}, F_{dec})$ – карта уваги, що обчислюється спільно з gate-сигналу декодера та feature map енкодера. По-друге, модулі уваги можуть вбудовуватися всередині блоків декодера після кожної операції up-sampling, що дозволяє мережі на кожному масштабі динамічно переважувати просторові позиції перед подальшою обробкою.

Канальна увага (channel attention) є доповненням до просторової та обчислює ваги важливості для кожного каналу feature map окремо. Разом вони формують модуль CBAM (Convolutional Block Attention Module) [17]:

$$\tilde{F} = \mathcal{A}_{spatial}(F') \otimes F', \quad F' = \mathcal{A}_{channel}(F) \otimes F$$

Спочатку застосовується канальна увага, а потім просторова. У задачах сегментації рентгенівських знімків CBAM демонструє стабільне покращення метрики IoU порівняно з базовими архітектурами без уваги, оскільки одночасно виділяє як релевантні канали (типи ознак), так і релевантні просторові позиції (локалізацію патології).

2.3.5 Блок scSE як механізм уваги в архітектурах UNet++ та DeepLabV3+

Блок scSE (Concurrent Spatial and Channel Squeeze & Excitation) є розширенням ідеї CBAM і поєднує просторову та канальну увагу паралельно, а не послідовно [17]. Блок складається з двох гілок:

cSE (channel SE) – стискає просторові виміри через глобальний усереднювальний пулінг, після чого через двохшарову повнозв'язну мережу з сигмоїдою обчислює вектор канальних ваг $\omega \in [0, 1]^C$:

$$F_{cSE} = \sigma(W_2 ReLU(W_1 AvgPool(F))) \otimes F$$

sSE (spatial SE) – стискає канальний вимір згорткою 1×1 , отримуючи одноканальну карту просторових ваг:

$$F_{sSE} = \sigma(\text{Conv}_{1 \times 1}(F)) \otimes F$$

Вихід блоку scSE – поелементна сума обох гілок [17]:

$$\tilde{F} = F_{cSE} + F_{sSE}$$

Завдяки паралельній обробці блок одночасно виділяє які канали (типи ознак) і де просторово важлива інформація, не збільшуючи розмірність тензора.

У архітектурі UNet++ блок scSE вбудовується у кожен блок декодера через параметр `decoder_attention_type="scse"` бібліотеки `segmentation_models_pytorch`: після кожного кроку up-sampling ознаки переважаються перед подальшою конкатенацією по skip connection.

У архітектурі AttentionDeepLabV3+ застосовано власний механізм уваги на рівні логітів: просторова gate функція обчислює карту ваг $G = \sigma(\text{Conv}_{1 \times 1}(\hat{y}))$, а блок уточнення (refine) Δ коригує вихід:

$$\tilde{y} = \hat{y} + G \otimes \Delta(\hat{y}),$$

де $\hat{y}$ – вихідні логіти базової моделі DeepLabV3+. Такий підхід дозволяє мережі пригнічувати хибні активації на нерелевантних просторових позиціях вже після декодера, без зміни внутрішніх feature maps.

2.4 Функції втрат для задач із дисбалансом класів

У задачах сегментації медичних зображень суттєвою проблемою є дисбаланс класів: зона перелому займає незначну частину пікселів знімка порівняно з нормальною кістковою тканиною та фоном. Стандартні функції втрат, що мінімізують середню помилку по всіх пікселях, у таких умовах схилиються до тривіального рішення – передбачати переважаючий клас. У цьому розділі описано функції втрат, що застосовуються для подолання цієї проблеми.

Зважена крос-ентропія є модифікацією стандартної крос-ентропії, що вводить ваговий коефіцієнт β для компенсації дисбалансу класів [8]:

$$H_{\omega CE}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [\beta * y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)],$$

де $\beta = \frac{N_{neg}}{N_{pos}}$ – відношення кількості негативних пікселів до позитивних.

Збільшений штраф за пропуск пікселів перелому змушує модель приділяти більше уваги міноритарному класу.

Бінарна крос-ентропія (Binary Cross-Entropy) є базовою функцією втрат для бінарної сегментації та відповідає випадку $\beta = 1$ формулі вище [8]:

$$H_{BCE}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

BCE оптимізує попиксельну точність, однак чутлива до дисбалансу класів, оскільки внесок міноритарного класу у загальну суму є незначним.

Dice Loss базується на коефіцієнті Сьоренсена-Дайса і вперше запропонована для медичної сегментації [18]. Функція безпосередньо оптимізує метрику перекриття між передбаченою маскою та еталоном:

$$H_{Dice}(y, \hat{y}) = 1 - \frac{2 \sum_i y_i \hat{y}_i + \varepsilon}{\sum_i y_i + \sum_i \hat{y}_i + \varepsilon},$$

де ε – невелика константа для чисельної стабільності (зазвичай $\varepsilon = 1$). На відміну від BCE, Dice Loss інваріантна до абсолютного розміру маски, що робить її стійкою до дисбалансу класів [18].

Tversky Loss є узагальненням Dice Loss з асиметричними штрафами за хибно-позитивні (FP) та хибно-негативні (FN) помилки [19]:

$$H_{Tversky}(y, \hat{y}) = 1 - \frac{\sum_i y_i \hat{y}_i + \varepsilon}{\sum_i y_i \hat{y}_i + \alpha \sum_i (1 - y_i) \hat{y}_i + \beta \sum_i (1 - y_i) \hat{y}_i + \varepsilon},$$

де $\alpha + \beta = 1$ – параметри, що контролюють баланс між FP та FN. При $\alpha = \beta = 0.5$ функція зводиться до Dice Loss. Для задач сегментації переломів

збільшення $\beta > 0.5$ дозволяє штрафувати пропуски зони перелому сильніше, ніж хибні спрацювання.

Практика комбінування попіксельних та перекривальних функцій втрат є усталеним підходом у медичній сегментації. Застосовується комбінована функція:

$$H_{seg} = \omega_1 * H_{BCE} + \omega_2 * H_{Dice} + \omega_3 * H_{Tversky}$$

BCE забезпечує стабільний градієнт на початку навчання, Dice Loss спрямовує оптимізацію на максимізацію перекриття масок [18], а Tversky Loss додатково акцентує увагу на зменшенні хибно-негативних помилок [19].

2.5 Трансферне навчання

Навчання глибоких нейронних мереж з нуля вимагає великих обсягів розмічених даних та значних обчислювальних ресурсів. У медичній візуалізації обидві умови рідко виконуються одночасно: анотування рентгенівських знімків потребує кваліфікації лікаря, а датасети залишаються відносно невеликими порівняно із задачами комп'ютерного зору загального призначення. Трансферне навчання (transfer learning) дозволяє подолати це обмеження, використовуючи знання, отримані на одній задачі, як початкову точку для навчання на іншій.

У даній роботі для всіх архітектур використовується енкодер ResNet-50 з вагами ImageNet та стратегія повного дообучення зі зменшеним learning rate для енкодера. Вибір ResNet-50 обумовлений його перевіреною ефективністю як backbone для сегментації: залишкові з'єднання вирішують проблему зникаючого градієнта, а п'ятирівнева ієрархія ознак добре узгоджується зі структурою U-Net++ та DeepLabV3+.

2.6 Метрики оцінки якості

Об'єктивне порівняння моделей потребує чітко визначених кількісних критеріїв. Оскільки система розв'язує дві послідовні підзадачі – класифікацію знімка та сегментацію зони перелому – для кожної з них застосовується окремий набір метрик.

2.6.1 Метрики класифікації

Для бінарної класифікації базовою мірою є точність (ассигасу) – частка правильно класифікованих зображень серед усіх:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Проте в умовах дисбалансу класів точність є оманливим показником: модель, що завжди передбачає відсутність перелому, матиме високу ассигасу, але практичної цінності не нестиме.

Тому основними метриками обрано precision та recall:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

де TP – істинно-позитивні;

TN – істинно-негативні;

FP – хибно-позитивні;

FN – хибно-негативні результати класифікації.

Precision відображає, яка частка передбачених переломів є дійсними переломами – тобто характеризує рівень хибних спрацювань.

Recall (sensitivity) відображає, яка частка реальних переломів була виявлена моделлю – тобто характеризує рівень пропусків.

Для інтегральної оцінки використовується F1-score – гармонічне середнє precision та recall:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

У контексті медичної діагностики пріоритет надається recall, оскільки пропуск реального перелому (помилка другого роду) є клінічно значно небезпечнішим, ніж хибне спрацювання (помилка першого роду): ненадана вчасно допомога при переломі може призвести до ускладнень, тоді як хибно-позитивний результат лише ініціює додаткове обстеження [6]. Таким чином, модель із вищим recall є пріоритетною навіть за незначного зниження precision.

2.6.2 Метрики сегментації: Dice Score та IoU

Якість піксельної сегментації оцінюється через ступінь просторового збігу між передбаченою маскою $\hat{M}$ та еталонною розміткою M [18].

Dice Score (коефіцієнт Сьоренсена–Дайса) вимірює відносний розмір перетину двох масок, нормований на їхній сумарний обсяг [18]:

$$Dice = \frac{2|M \cap \hat{M}|}{|M| + |\hat{M}|} = \frac{2 * TP}{2 * TP + FP + FN}$$

Значення змінюється від 0 (повна розбіжність) до 1 (ідеальний збіг). Метрика є стійкою до дисбалансу класів, оскільки враховує лише пікселі, що належать до зони інтересу.

IoU (Intersection over Union, індекс Жаккара) обчислює частку перетину масок відносно їх об'єднання [18]:

$$IoU = \frac{|M \cap \hat{M}|}{|M \cup \hat{M}|} = \frac{TP}{TP + FP + FN}$$

IoU є строгішою метрикою порівняно з Dice Score: однаковій якості сегментації відповідає нижче числове значення IoU, оскільки знаменник більший. Обидві метрики монотонно пов'язані між собою:

$$IoU = \frac{Dice}{2 - Dice}$$

Аналогічно до класифікації, у сегментації також приділяється особлива увага recall на рівні пікселів – здатності моделі не пропускати пікселі зони перелому, навіть якщо це відбувається за рахунок незначного збільшення хибних спрацювань.

2.7 Методи інтерпретації рішень згорткових нейронних мереж

Карти активацій класів (Class Activation Maps, CAM) є інструментом інтерпретації рішень згорткових нейронних мереж: вони дозволяють візуалізувати, які просторові регіони зображення найбільше вплинули на передбачення моделі. У діагностичних системах CAM відіграють подвійну роль – з одного боку, підвищують довіру лікаря до рішення моделі, надаючи просторове обґрунтування класифікації, з іншого – дозволяють отримати приблизну локалізацію патології без явної розмітки bounding box під час навчання [11].

2.7.1 GradCAM та GradCAM++

GradCAM (Gradient-weighted Class Activation Mapping), запропонований Selvaraju et al. є узагальненням оригінального методу CAM на довільні архітектури без модифікації структури мережі [20]. Ключова ідея

полягає у використанні градієнтів цільового класу відносно feature maps останнього згорткового шару як міри важливості кожного каналу.

Для заданого класу c обчислюються ваги важливості каналу k :

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial F_k^{ij}},$$

де y^c – оцінка класу c до softmax;

F_k^{ij} – активація пікселів (i, j) у каналі k ;

Z – кількість просторових позицій.

Карта активацій формується як зважена сума каналів із застосуванням ReLU:

$$\mathcal{A}_{GradCAM}^c = ReLU \left(\sum_k \alpha_k^c * F_k \right)$$

ReLU відфільтровує негативні активації, залишаючи лише регіони, що підтверджують належність до класу [19].

GradCAM++, запропонований Chattopadhyay et al., усуває основний недолік GradCAM – неточну локалізацію при наявності кількох об'єктів одного класу на зображенні. Метод вводить попіксельні ваги градієнтів замість їх глобального усереднення:

$$\alpha_k^c = \sum_i \sum_j \omega_k^{ij} * ReLU \left(\frac{\partial y^c}{\partial F_k^{ij}} \right),$$

де ω_k^{ij} – аналітично обчислені ваги, що враховують внесок кожної просторової позиції. Завдяки цьому GradCAM++ демонструє більш точне та повне покриття об'єкта інтересу, що особливо важливо при локалізації дрібних зон перелому.

2.7.2 LayerCAM

LayerCAM, запропонований Jiang et al., усуває фундаментальне обмеження GradCAM-підходів – прив'язку до єдиного останнього згорткового шару [11]. Ранні шари мережі містять дрібнодеталізовану просторову інформацію, але традиційні методи її ігнорують через нестабільність градієнтів при зворотному поширенні на великі глибини.

Ключова ідея LayerCAM полягає у попіксельному зважуванні активацій безпосередньо у шарі, що аналізується, на основі позитивних градієнтів:

$$\mathcal{A}_{LayerCAM} = ReLU \left(\sum_k ReLU \left(\frac{\partial y^c}{\partial F_k^{ij}} \right) \odot F_k \right),$$

де $\odot$ – по елементне множення. Використання $ReLU$ до градієнтів гарантує, що лише позитивні градієнти (що підтверджують клас) внесуть вклад у карту активацій. Фінальна карта отримується агрегацією результатів з кількох шарів різних масштабів.

2.7 Висновки до розділу 2

У даному розділі викладено теоретичні основи та математичний апарат, що складають фундамент розробленої системи автоматизованої діагностики переломів кісток на рентгенівських знімках.

Розглянуто принципи роботи згорткових нейронних мереж як основного інструменту обробки медичних зображень. Показано, що три ключові властивості CNN – локальна зв'язність, розділення ваг та ієрархічність представлень роблять їх особливо придатними для аналізу

рентгенограм, де діагностично значущі структури представлені на різних просторових масштабах.

Для задачі класифікації рентгенівських знімків проаналізовано три архітектури. ResNet18 та ResNet50 побудовані на механізмі залишкових з'єднань, що вирішує проблему деградації градієнтів у глибоких мережах. ConvNeXtV2-Tiny є архітектурою сучасного покоління, що поєднує переваги CNN та трансформерів

Для задачі піксельної сегментації розглянуто три принципово різні підходи. U-Net вирішує проблему збереження просторових деталей через пропускні з'єднання між симетричними рівнями енкодера та декодера. U-Net++ усуває семантичний розрив між цими з'єднаннями шляхом введення щільних блоків вкладених згорток із механізмом глибокого нагляду. DeepLabV3+ принципово відрізняється від попередніх: замість симетричного енкодер-декодера використовує дилатовані згортки з різними коефіцієнтами розширення та модуль ASPP для одночасного аналізу об'єктів на кількох масштабах без втрати просторової роздільності. А також розглянуто їхні модифікації з механізмом просторової уваги.

Окремо обґрунтовано вибір функцій втрат для задачі з великим дисбалансом класів. Описано принцип трансферного навчання з ImageNet як ключового інструменту подолання дефіциту анотованих медичних даних. Визначено метрики оцінки якості: для класифікації – Accuracy, Precision, Recall, F1 та AUC з пріоритетом на Recall; для сегментації – Dice Score та IoU як основні міри піксельного збігу. Нарешті, розглянуто методи інтерпретації рішень CNN (GradCAM, GradCAM++, LayerCAM), що відіграють подвійну роль у системі: підвищують довіру лікаря до рішення моделі та забезпечують автоматичну локалізацію переломної зони без явної розмітки bounding box під час навчання.

РОЗДІЛ 3 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

3.1 Збір та підготовка навчальних даних

Якість та репрезентативність вхідних даних є визначальним фактором для успішного навчання моделей глибокого навчання, тому етапу підготовки даних приділено особливу увагу. Розглянуто вибір датасету, обґрунтування методів нормалізації та стандартизації, порівняльний аналіз підходів до підвищення контрасту зображень, а також стратегію аугментації з урахуванням специфіки медичних даних.

3.1.1 Попередня обробка та стандартизація знімків

Дослідження базується на датасеті FracAtlas рентгенографічних зображень кістково-м'язової системи, орієнтованому на задачі класифікації, локалізації та сегментації переломів. Він містить 4 083 зображення, серед яких 717 знімків із 922 задокументованими переломами та 3 366 нормальних знімків. На рисунку 3.1 зображено приклад зображень з переломом та маскою, без перелому.

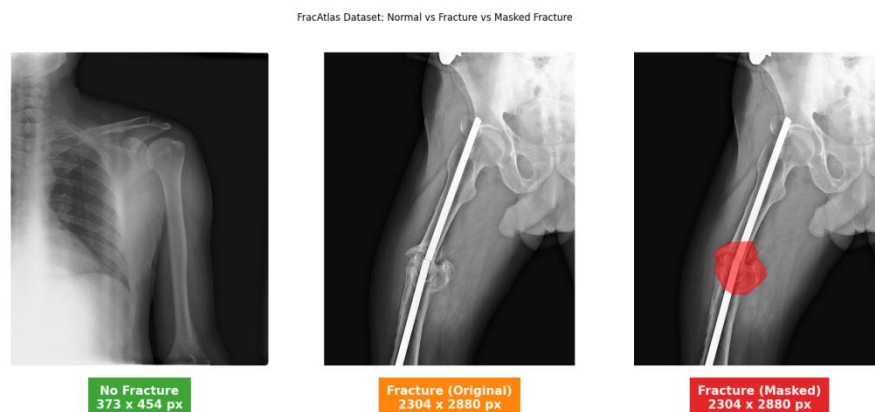


Рисунок 3.1 – Приклад зображень FracAtlas

На відміну від більшості публічних медичних датасетів, FracAtlas має трирівневу анотацію (класифікація, локалізація, сегментація), виконану вручну двома радіологами та ортопедом. Датасет зібрано з трьох великих лікарень, що забезпечує різноманітність обладнання та умов зйомки, а анотації доступні у форматах COCO JSON, VGG, YOLO та Pascal VOC. Для задачі сегментації використовується формат COCO JSON як найінформативніший, що дозволяє конвертацію в інші формати без втрат.

Усі зображення приведено до розміру 224×224 пікселів – стандартного вхідного розміру для архітектур із попереднім навчанням на ImageNet, що забезпечує прийнятний баланс між деталізацією та швидкістю обчислень. Нормалізація значень пікселів до діапазону $[0.0, 1.0]$ забезпечує числову стабільність градієнтного спуску, а стандартизація відносно статистик ImageNet є обов'язковою умовою коректного використання ваг попереднього навчання.

Для вибору оптимального методу підвищення якості зображень було порівняно п'ять підходів (рисунки 3.2).

1. CLANE LAB – CLANE застосовується до каналу яскравості (L) у просторі LAB, зберігаючи кольорові канали незмінними. Підвищує загальний контраст, проте для рентгену, де кольорова інформація є надлишковою, ця перевага не є суттєвою.
2. CLANE Grayscale – зображення переводиться в одноканальний формат, після чого застосовується локальна адаптивна еквалізація гістограми з обмеженням підсилення. Найчіткіше виявляє трабекулярну структуру кістки та лінії переломів.
3. Unsharp Mask – підсилює краї шляхом додавання різниці між оригіналом та його розмитою версією. Добре виділяє контури кістки, але посилює також артефакти зображення та шум.
4. Gamma 0.75 – гамма-корекція освітлює темні ділянки знімка. Корисна при нерівномірній експозиції, але не покращує локальний контраст і не підсилює дрібні деталі структури.

5. Bone BG Removal – видалення чорного фону за допомогою порогової бінаризації Otsu, морфологічних операцій та inpainting. Прибирає нерелевантні темні ділянки, однак ризикує видалити периферійні частини кістки та ускладнює роботу з масками сегментації.

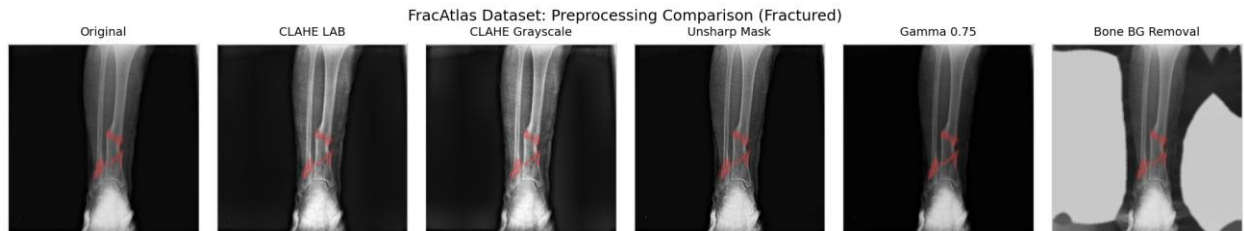


Рисунок 3.2 – Порівняння методів препроцесингу зображень

За результатами візуального порівняння для подальшої роботи обрано CLAHE Grayscale. Метод забезпечує найкращу видимість переломних ліній і трабекулярних структур без артефактів, властивих Unsharp Mask, та без ризику втрати країв, характерного для Bone BG Removal. Перехід до одноканального представлення при цьому не несе діагностичних втрат, оскільки вся клінічно значуща інформація в рентгені закодована у яскравісному каналі.

3.1.2 Застосування аугментації для підвищення варіативності навчальної вибірки

FracAtlas містить лише 717 знімків із переломами, що є недостатнім для стабільного навчання глибоких мереж без ризику перенавчання. Для вирішення цієї проблеми застосовано аугментацію даних – техніку штучного розширення навчальної вибірки шляхом геометричних та фотометричних перетворень оригіналів. При цьому всі геометричні перетворення застосовуються одночасно до зображення та його маски, зберігаючи просторову відповідність між знімком і розміткою.

Використовувалась онлайн-аугментація – перетворення застосовуються випадковим чином безпосередньо в процесі навчання, що забезпечує безперервну варіативність без необхідності зберігати розширений датасет на диску. Аугментацію навмисно зроблено слабкою, щоб не спотворювати тонкі лінії переломів. Застосовувались такі трансформації (рис. 3.3):

- 1) Horizontal Flip – дзеркальне відображення по горизонталі;
- 2) Rotate $\pm 90^\circ$ – поворот на 90° ;
- 3) Shift/Scale/Rotation – комбінація зсуву, масштабування та довільного повороту;
- 4) Brightness/Contrast – випадкова зміна яскравості та контрасту.



Рисунок 3.3 Порівняння видів аугментації

3.2 Тренування моделей класифікації

Для навчання класифікаторів використовувався датасет FracAtlas у повному обсязі – 4 083 зображення з дисбалансом класів 1:4.7. Для компенсації дисбалансу використовувалась зважена крос-ентропія з автоматично обчисленими ваговими коефіцієнтами, що змушує модель приділяти більше уваги класу «перелом».

Для задачі бінарної класифікації обрано три архітектури з попереднім навчанням на ImageNet: ResNet18 (11.2 млн параметрів) як базова лінія, ResNet50 (23.5 млн параметрів) як глибша версія з bottleneck-блоками, та

ConvNeXtV2-Tiny (27.9 млн параметрів) як сучасна архітектура під впливом трансформерів. Детальний опис архітектур наведено в розділі 2.

3.2.1 Стратегія покрокового розморожування

Для уникнення руйнування претренованих ваг backbone великими градієнтами від випадково ініціалізованої голови застосовано стратегію поетапного розморожування з чотирьох стадій:

- 1) Head – навчається лише класифікаційна голова, backbone повністю заморожений;
- 2) Last block – підключається останній блок backbone, що відповідає за високорівневі ознаки, специфічні для задачі;
- 3) Mid block – підключаються ще 2–3 попередні блоки backbone із зменшеним learning rate;
- 4) Full – розморожується вся мережа, backbone навчається з дуже малим learning rate.

3.3 Аналіз якості моделей класифікації

3.3.1 Аналіз функцій втрат

Криві валідаційних втрат усіх трьох моделей (рисунок 3.4) демонструють схожу динаміку: різке зниження впродовж перших 4–5 епох із подальшим виходом на плато приблизно на епосі 10. Це свідчить про те, що для даного датасету 10 епох є достатнім для стабільної збіжності, а подальше навчання лише незначно покращує результат.

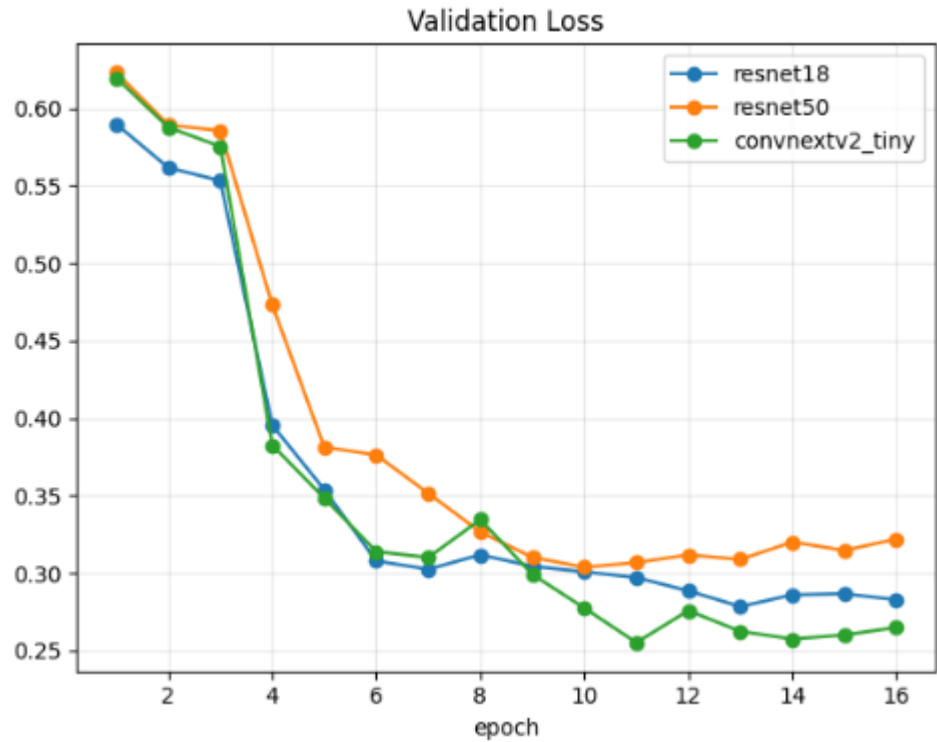


Рисунок 3.4 Графік функції втрат на валідаційні вибірці

3.3.2 Оцінки метрик якості

Підсумкові метрики на тестовій вибірці (1 451 зображення) наведено в таблиці 3.1. Для оцінювання використовувався набір метрик, такі як Accuracy, Precision, Recall та AUC.

Таблиця 3.1 – Порівняння метрик якості кожної з моделей

Модель	Accuracy	Precision	Recall	F1	AUC
ResNet18	92,3 %	88,8 %	85 %	86,9 %	96 %
ResNet50	92,8 %	89,8 %	86,1 %	87,8 %	95,7 %
ConvNeXtV2-Tiny	93,5 %	90,2 %	87,6 %	88,8 %	95,5 %

Динаміка валідаційної повноти (рисунок 3.5) демонструє виражені коливання протягом перших 8 епох навчання для всіх трьох архітектур, після чого графіки стабілізуються і виходять на плато.

Впродовж усього процесу навчання модель ConvNeXtV2-Tiny стабільно демонструє найкращі результати, суттєво випереджаючи аналоги та досягаючи максимального значення Recall на рівні близько 0.86–0.87 (із піковим значенням понад 0.90 на другій епісі). Моделі ResNet50 та ResNet18 показують схожий характер поведінки, проте фіксуються на нижчих позиціях: ResNet50 виходить на плато в діапазоні 0.84–0.85, тоді як найпростіша архітектура ResNet18 замикає рейтинг із показниками повноти на рівні 0.82–0.83.



Рисунок 3.5 – Графік порівняння повноти моделей

Схожу картину відображає крива валідаційного F1 (рисунок 3.6): різкий приріст до епохи 4, після чого значення стабілізуються в діапазоні 0.86–0.90.

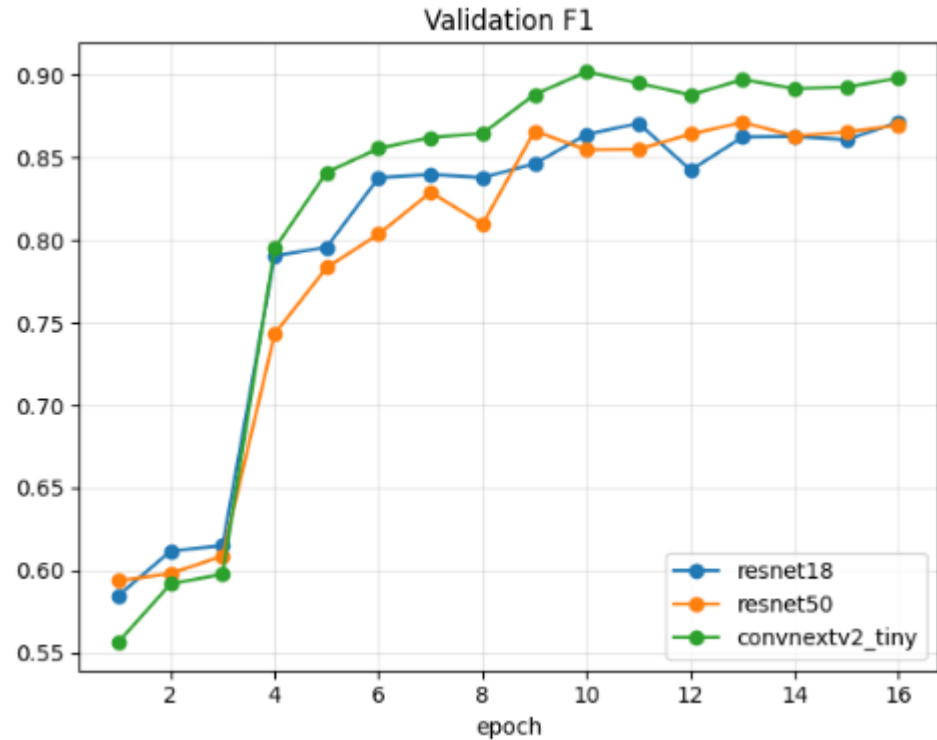


Рисунок 3.6 – Графік порівняння метрики F1

3.3.3 Візуалізація результатів класифікації

На рисунку 3.7 наведено приклади знімків із переломами, які модель впевнено класифікувала правильно з імовірністю $p(\text{fracture}) = 1.0$. Обидва знімки належать до FracAtlas і зображують кісткові структури гомілки та стопи з візуально помітними лініями переломів – саме на таких випадках модель демонструє максимальну впевненість.



Рисунок 3.7 – Приклад правильно визначеного класу “перелом”

Для інтерпретації рішень моделі застосовано три методи візуалізації активацій: GradCAM, GradCAM++ та LayerCAM. На рисунку 3.8 показано приклад для FN випадку – знімок зап'ястка, який модель помилково класифікувала як норму. Теплові карти всіх трьох методів вказують на область зап'ястка та прилеглих кісток, що свідчить про те, що модель все ж аналізувала релевантну зону, однак не змогла виявити ознаки перелому з достатньою впевненістю. На нижньому ряду зображено автоматично згенеровані bounding box-и на основі CAM-активацій – вони охоплюють центральну частину знімка, де зосереджені ознаки, що вплинули на рішення.

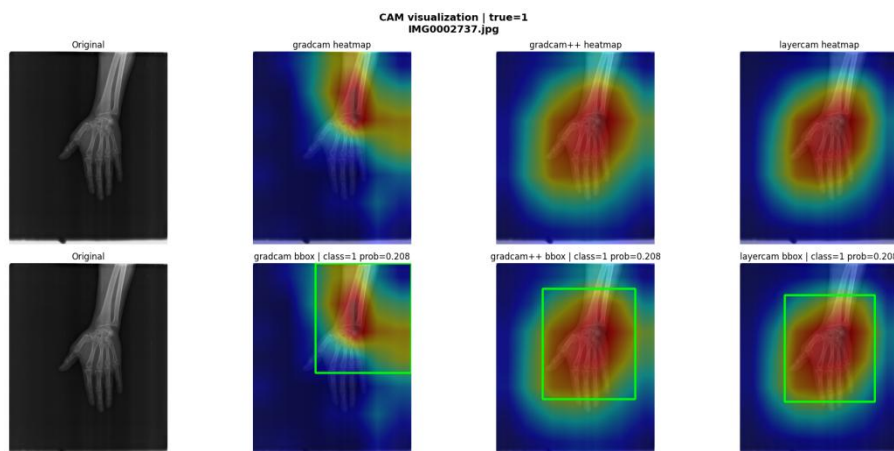


Рисунок 3.8 – Візуальне порівняння CAM активацій

За результатами візуального порівняння трьох методів для подальшої роботи обрано LayerCAM. На відміну від GradCAM та GradCAM++, які формують згладжені теплові карти з широкою зоною активації, LayerCAM генерує більш локалізовані та деталізовані карти з чіткішими межами активної зони.

3.4 Тренування моделей сегментації

3.4.1 Базові архітектури сегментації

Для задачі піксельної сегментації переломних зон обрано три архітектури класу «енкодер-декодер», загальновизнані у задачах медичної сегментації.

U-Net – базова архітектура для сегментації біомедичних зображень [12]. Ключова ідея – прямі пропуски з'єднання (skip connections), що передають просторово точну інформацію з ранніх шарів у декодер, компенсуючи втрату деталей при субдискретизації. Слугує базовою лінією для порівняння.

U-Net++ – розширення U-Net із переробленою системою пропускових з'єднань [13]. Замість прямих з'єднань вводиться щільна мережа проміжних вузлів, що забезпечує семантично однорідніші ознаки у декодері та ефективніше поєднання просторової і семантичної інформації.

DeepLabV3+ – архітектура з принципово іншим підходом до розширення рецептивного поля [14]. Дилатовані згортки з різними коефіцієнтами розширення об'єднані у модулі ASPP, що дозволяє аналізувати контекст на кількох масштабах без втрати просторової роздільності.

3.4.2 Архітектури з механізмом просторової уваги

На основі двох найкращих за результатами базових архітектур побудовано їх модифікації з інтегрованим механізмом уваги (attention) – Attention U-Net++ та Attention DeepLabV3+.

Механізм уваги [15] вводить у пропускні з'єднання або проміжні шари спеціальні ваги, що визначають, яким просторовим регіонам і каналам ознак приділяти більше уваги при формуванні передбачення. Коефіцієнти уваги обчислюються за формулою:

$$\alpha_i = \sigma_2(\psi^T * \sigma_1(W_g * g + W_x * x_i + b) + b_\psi),$$

Де x_i – вхідні ознаки з пропускового з'єднання;

g – контекстний сигнал з декодера;

W_g, W_x, ψ – матриці ваг;

σ_1 – ReLU;

σ_2 – сигмоїда.

Вихід attention-блоку: $\hat{x}_i = \alpha_i * x_i$.

На відміну від базових архітектур, де всі просторові позиції обробляються рівноправно, attention-блоки явно пригнічують активації у фонових ділянках і підсилюють їх у зонах, що з більшою імовірністю містять патологію. Для рентгенівських знімків кісток це особливо актуально: переломна зона може займати менше 1–2% площі знімка, і без додаткового механізму фокусування модель схильна ігнорувати її на користь домінуючого фону.

В Attention U-Net++ блоки уваги інтегруються у вузли щільної мережі пропускних з'єднань, дозволяючи адаптивно зважувати ознаки ще на етапі їх передачі між субмережами. В Attention DeepLabV3+ механізм уваги застосовується до виходів ASPP-модуля, що дозволяє зосередити мультимасштабний аналіз саме на релевантних просторових регіонах.

3.5 Стратегії формування вхідних даних для сегментації

Одним із ключових експериментальних питань роботи є те, як саме подавати зображення на вхід сегментаційної моделі. Розглядалися три стратегії, кожна з яких відображає різний рівень доступної інформації про локалізацію перелому. Порівняння стратегій проводиться на однакових архітектурах і дає змогу кількісно оцінити внесок попередньої локалізації у якість сегментації та визначити, наскільки автоматична САМ-локалізація наближається до еталонної.

3.5.1 Навчання на повних рентгенівських знімках

Модель отримує повне зображення розміром 224×224 без попередньої локалізації і повинна самостійно знайти переломну зону. Головний виклик – піксельний дисбаланс: зона перелому може займати менше 1–2% площі знімка. Підхід слугує базовою лінією для оцінки ефективності наступних стратегій.

3.5.2 Навчання на фрагментах із автоматичною локалізацією на основі LayerCAM

На вхід моделі подається фрагмент знімка, вирізаний за обмежувальною рамкою від класифікатора на основі LayerCAM. Це зменшує кількість фонових пікселів і дозволяє моделі зосередитись на релевантній

ділянці. Водночас якість обрізання залежить від точності САМ-локалізації, що вносить додаткову похибку.

3.5.3 Навчання на фрагментах з еталонною локалізацією

Зображення обрізається на основі анотованих вручну рамок із FracAtlas із розширенням області приблизно у п'ять разів для захоплення достатнього анатомічного контексту. Стратегія є верхньою межею точності локалізації та дозволяє оцінити залежність якості сегментації від точності попередньої локалізації.

3.6 Оцінка якості моделей сегментації

Для комплексної оцінки якості сегментаційних моделей аналізуються три аспекти: динаміка функції втрат у процесі навчання, підсумкові метрики на тестовій вибірці та візуальна якість передбачених масок. Оцінка проводиться окремо для кожної з трьох стратегій формування вхідних даних і охоплює всі п'ять архітектур.

3.6.1 Оцінка функцій втрат

Криві валідаційних втрат усіх моделей демонструють стійку спадну динаміку впродовж навчання з виходом на плато в діапазоні 30–40 епох

залежно від стратегії та архітектури. Характер збіжності суттєво відрізняється між стратегіями.

При навчанні на повних знімках всі моделі збігаються повільніше і виходять на значно вищі значення втрат порівняно з іншими стратегіями (рисунок 3.9). Це пояснюється складністю задачі: мережа змушена одночасно локалізувати та сегментувати переломну зону на тлі великої кількості фонових пікселів.

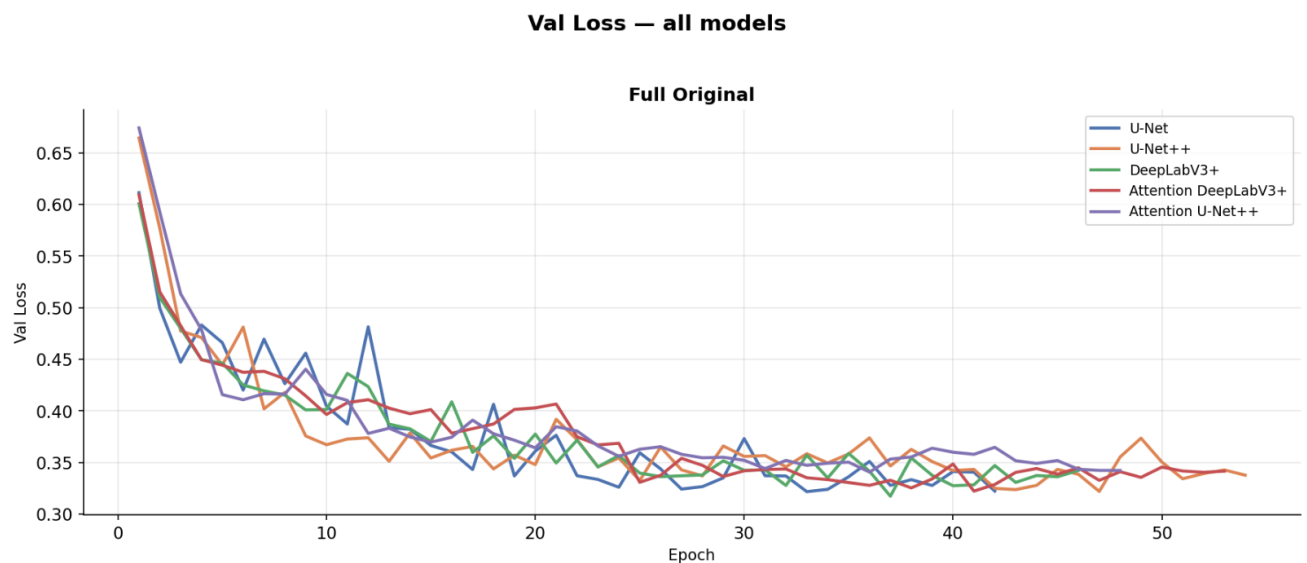


Рисунок 3.9 – Криві функцій втрат моделей на повних знімках

При навчанні на LayerCAM-фрагментах збіжність є швидшою, а фінальні значення втрат – нижчими, що відображено на рисунок 3.10. DeepLabV3+ та Attention DeepLabV3+ тут демонструють найкращу динаміку, тоді як U-Net та U-Net++ збігаються помітно гірше, що може свідчити про чутливість цих архітектур до якості автоматичної локалізації.

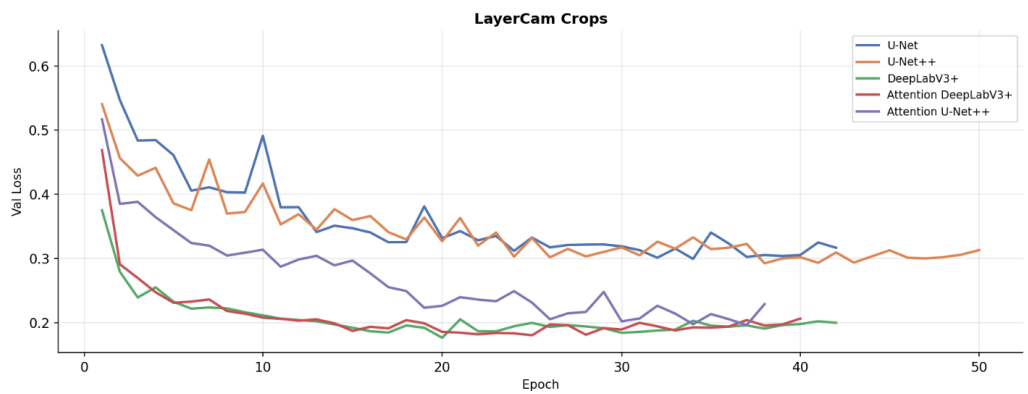


Рисунок 3.10 – Криві функцій втрат моделей на обрізаних знімках через LayerCAM

При навчанні на фрагментах, обрізаних за еталонними bounding box всі моделі показують найшвидшу та найстабільнішу збіжність з найнижчими значеннями тестових втрат, що наочно демонструє рисунок 3.11.

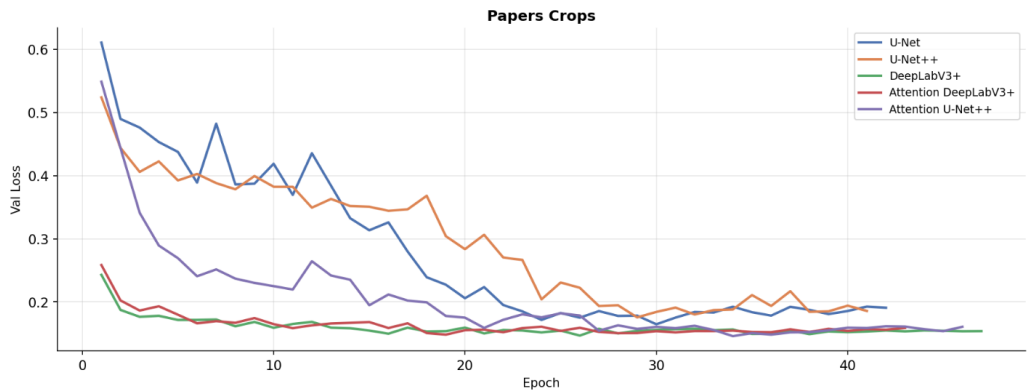


Рисунок 3.11 – Криві функцій втрат моделей на обрізаних знімках за еталонним рамками

3.6.2 Аналіз метрик якості

Підсумкові метрики якості сегментації на тестовій вибірці для всіх комбінацій стратегій та архітектур наведено в таблиці 3.2. Основними метриками оцінки є Dice Score та IoU, що безпосередньо вимірюють якість піксельного збігу передбаченої маски з еталонною розміткою.

Таблиця 3.2 – Метрики якості сегментації на тестовій вибірці

Модель	Dice Full	IoU Full	Dice LayerCam Crops	IoU LayerCam Crops	Dice Annotated Crops	IoU Annotated Crops
UNET	0,518	0,349	0,618	0,447	0,787	0,649
UNET++	0,536	0,366	0,596	0,425	0,773	0,630
DeepLabV3+	0,534	0,364	0,735	0,581	0,817	0,691
Attention DeepLabV3+	0,534	0,364	0,725	0,569	0,816	0,689
Attention UNET++	0,507	0,339	0,691	0,527	0,804	0,672

Результати підтверджують, що якість локалізації є визначальним фактором точності сегментації. При стратегії Full Original усі моделі показують найнижчі результати (Dice 0.507–0.536, IoU 0.339–0.366). Перехід до LayerCAM-фрагментів забезпечує суттєвий приріст: Dice зростає до 0.596–0.735, причому архітектури на основі DeepLabV3+ демонструють значно більший приріст завдяки мультимасштабному аналізу через ASPP.

За максимальною точністю сегментації лідирує DeepLabV3+, Attention-моделі мають перевагу за Recall, а найбільш практично застосовним варіантом для повністю автоматичного пайплайну є LayerCAM Crops + DeepLabV3+ з Dice = 0.735 без жодної ручної розмітки.

3.6.3 Візуалізація результатів сегментації

Кількісні метрики – Dice та IoU – є суворими мірами піксельного збігу: навіть незначне зміщення передбаченої маски відносно еталонної розмітки суттєво знижує їхнє значення. Водночас у клінічному контексті важливішою

є здатність моделі правильно локалізувати зону перелому, а не досягти ідеального піксельного перекриття з ручною анотацією, яка сама по собі є суб'єктивною оцінкою радіолога. Саме тому візуальний аналіз передбачених масок є необхідним доповненням до числових метрик і дозволяє отримати більш повне уявлення про реальну якість сегментації.

На рисунку 3.12 наведено результати інференсу моделі для трьох стратегій подачі вхідних даних: повний знімок, обрізане зображення за LayerCAM та обрізане зображення за еталонною розміткою. Візуально видно, що зі збільшенням точності локалізації вхідного фрагмента модель точніше відтворює форму та межі зони перелому – передбачені маски стають чіткішими та ближчими до еталону.

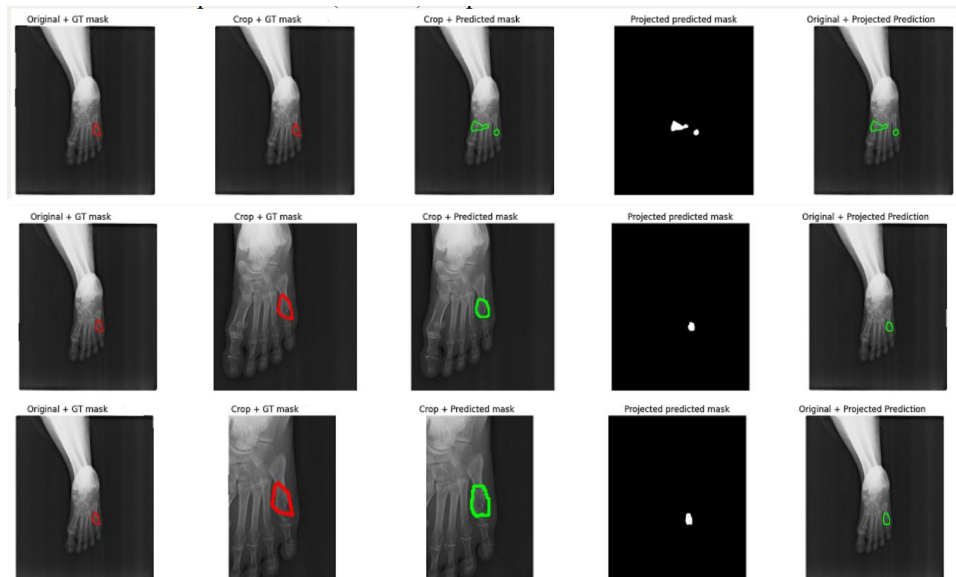


Рисунок 3.12 – Візуалізація передбачених масок на різних знімках

3.7 Система класифікації та сегментації

За результатами проведених досліджень було розроблено програмну систему автоматизованої діагностики переломів кісток на рентгенівських знімках. Система реалізує повний pipeline обробки медичного зображення –

від завантаження знімку до отримання сегментаційної маски ураженої ділянки – та об'єднує в єдиному застосунку модуль класифікації на основі ConvNeXtV2, модуль локалізації на основі LayerCAM і модуль сегментації на основі AttentionDeepLabV3+.

3.7.1 Архітектура системи

Розроблена система має трирівневу клієнт-серверну архітектуру, що забезпечує чіткий розподіл відповідальності між компонентами та спрощує масштабування і супровід системи. Загальну структуру архітектури наведено на рисунку 3.13.

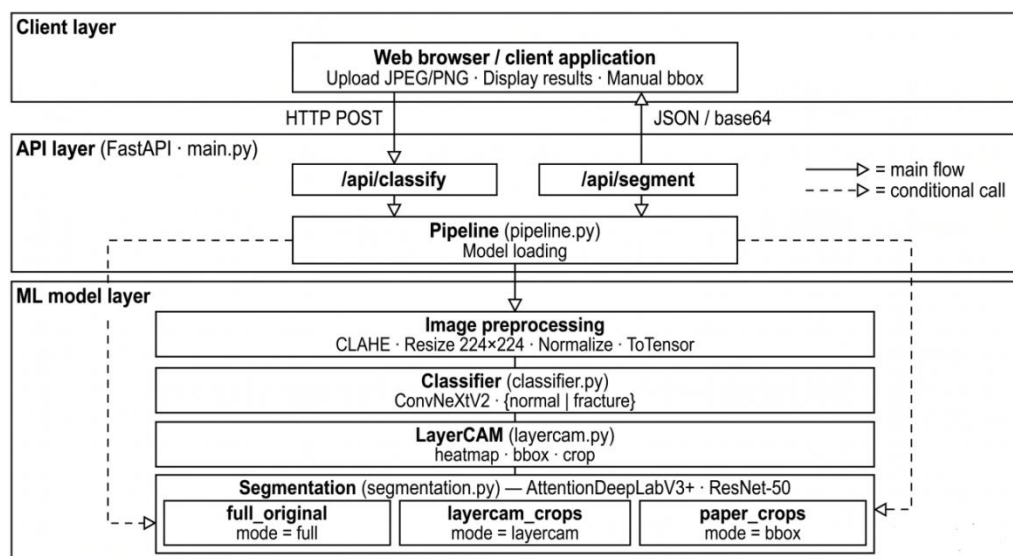


Рисунок 3.13 – Архітектурна схема створеної системи

Перший рівень – клієнтський, який представлений веб-браузером або клієнтським застосунком. На цьому рівні користувач завантажує рентгенівський знімок у форматі JPEG або PNG, переглядає результати класифікації та сегментації, а також за потреби вручну визначає область інтересу за допомогою bounding box.

Другий рівень – рівень API, реалізований на основі фреймворку FastAPI і містить два ендпоінти: `classify` для виконання класифікації та генерації теплової карти, і `segment` для запуску сегментації в одному з трьох режимів. Обидва ендпоінти взаємодіють через спільний компонент Pipeline, який відповідає за завантаження моделей при старті сервера.

Третій рівень – рівень ML-моделей, який включає послідовно з'єднані модулі передобробки зображення, класифікації, локалізації (LayerCAM) та сегментації. Модуль сегментації представлений трьома окремими checkpoint-ами моделі AttentionDeepLabV3+, кожен з яких навчений на відповідному типі вхідних даних.

3.7.2 Функціональна схема системи

Функціональна схема відображає послідовність обробки запиту в системі від моменту завантаження зображення до формування кінцевого результату. Схему наведено на рисунку 3.14.

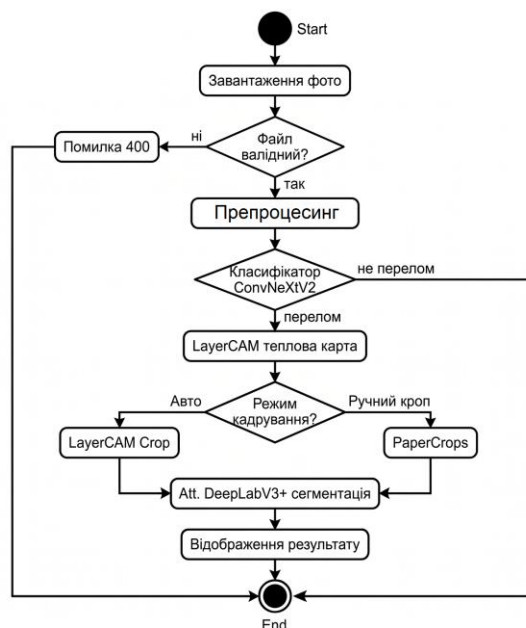


Рисунок 3.14 – Функціональна схема створеної системи

Після завантаження знімку система перевіряє коректність формату, після чого зображення проходить передобробку: CLAHE, масштабування до 224×224 та нормалізацію за статистиками ImageNet. Оброблене зображення подається на класифікатор ConvNeXtV2, який визначає наявність перелому та запускає модуль LayerCAM для формування теплової карти і визначення bounding box зони ураження.

На другому етапі користувач обирає режим сегментації – повний знімок, CAM-кроп або ручний bbox – після чого активується відповідний checkpoint моделі, результатом якого є бінарна маска, накладена на вхідне зображення.

3.7.3 Демонстрація роботи системи

Для демонстрації роботи системи використовувався рентгенівський знімок гомілки. Після завантаження зображення користувач натискає кнопку «Analyze X-Ray» для запуску аналізу. На рисунку 3.15 зображено інтерфейс програми.

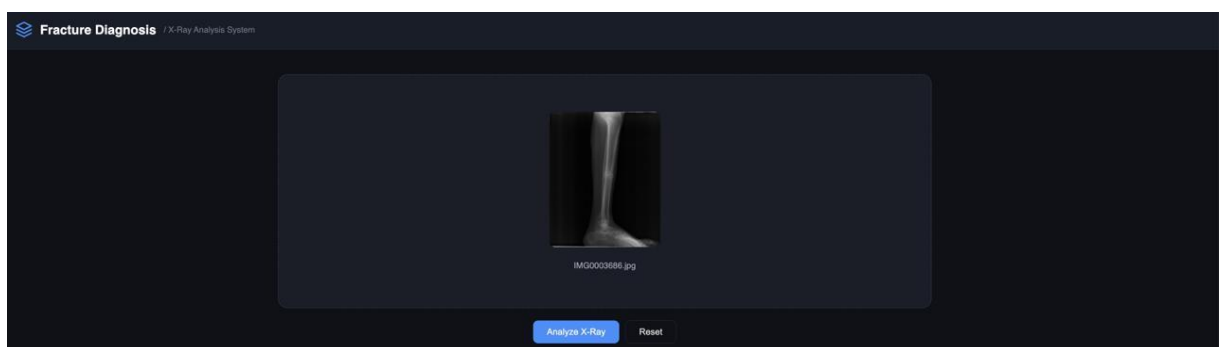


Рисунок 3.15 – Інтерфейс програми

Класифікатор на основі ConvNeXtV2 визначив наявність перелому з показником впевненості 98.0%. Модуль LayerCAM сформував теплову карту активацій, яка візуально підтверджує локалізацію зони ураження в середній

частині великогомілкової кістки. На рисунку 3.16 зображено приклад результату класифікації та генерації теплової карти.



Рисунок 3.16 – Результат класифікації та телова карта

На основі САМ-кропу модель сегментації AttentionDeepLabV3+ побудувала бінарну маску, що точно окреслює межі зони перелому. Маска накладається на вхідне зображення у вигляді напівпрозорого червоного контуру. Маску сегментації зображено на рисунку 3.17

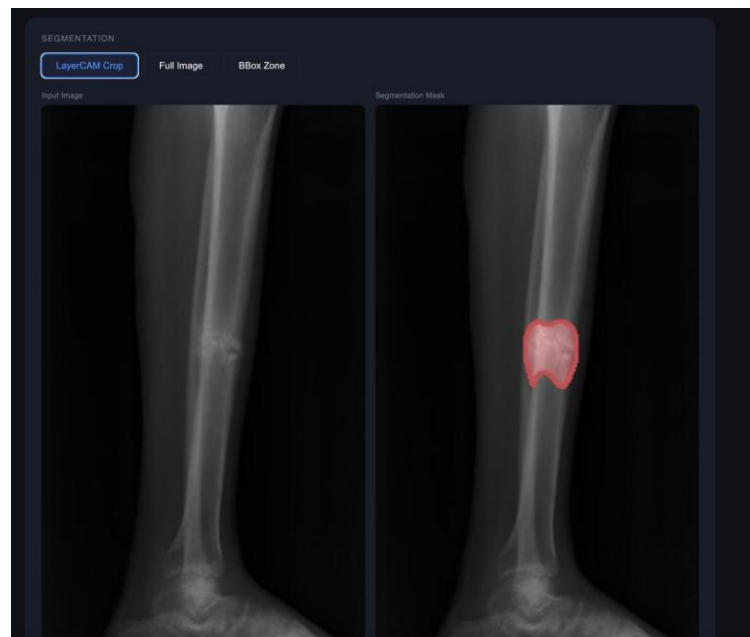


Рисунок 3.17 – Приклад генерації маски місця перелому

3.8 Висновки до розділу 3

У розділі 3 описано повний цикл проведених досліджень – від підготовки даних до навчання та оцінки моделей класифікації і сегментації переломів.

На етапі підготовки даних використовувався датасет FracAtlas (4 083 рентгенограми). Зображення приведено до розміру 224×224 , виконано нормалізацію та стандартизацію відносно статистик ImageNet, обрано CLAHE Grayscale як найефективніший метод підвищення контрасту, застосовано онлайн-аугментацію зі слабкими перетвореннями.

На етапі класифікації навчено три архітектури – ResNet18, ResNet50 та ConvNeXtV2-Tiny – із поетапним розморожуванням та зваженою крос-ентропією. Найкращу якість показала ConvNeXtV2-Tiny з Accuracy = 93.0% та F1 = 87.7%. За результатами порівняння методів візуалізації активацій обрано LayerCAM, bounding box-и якого використано як підказки для сегментації.

На етапі сегментації навчено п'ять архітектур за трьома стратегіями вхідних даних. Якість локалізації виявилась визначальним фактором: перехід від Full Original до GT Crops дає приріст Dice на 0.27–0.30, а LayerCAM Crops без ручної розмітки забезпечує Dice до 0.735, що підтверджує практичну цінність запропонованого двостадійного пайплайну.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться техніко-економічне обґрунтування розробки програмного продукту для автоматизованої діагностики переломів кісток на рентгенівських знімках. Розроблена система реалізує двостадійний діагностичний пайплайн: на першому етапі виконується бінарна класифікація знімка з одночасною генерацією теплових карт активацій для локалізації зони ушкодження, на другому – піксельна сегментація переломної зони на основі отриманої локалізації.

Оскільки медичні інтелектуальні системи вимагають не лише високої точності прогнозів, а й раціонального використання апаратних ресурсів (зокрема при навчанні та інференсі моделей глибокого навчання), розробка такого продукту потребує комплексного підходу до вибору архітектурних та програмних рішень.

Для вибору оптимального варіанту проектування застосовується метод функціонально-вартісного аналізу (ФВА) – технологія, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури, виявити резерви зниження витрат та забезпечити краще співвідношення між споживчою цінністю виробу та витратами на його реалізацію. Для проведення аналізу використовується економічна, технічна та конструкторська інформація. Аналіз проводиться поетапно: спочатку визначаються основні та допоміжні функції продукту, формуються альтернативні варіанти їх реалізації, після чого на основі технічних параметрів та розрахованої собівартості обирається оптимальний варіант для впровадження.

4.1 Постановка задачі проектування

Об'єктом проектування є програмний продукт для автоматизованої діагностики переломів кісток на рентгенівських знімках. Дана система призначена для допомоги медичним спеціалістам (радіологам, травматологам) шляхом швидкого виявлення наявності ушкоджень та точної локалізації їхніх меж, що дозволяє знизити ймовірність лікарської помилки та прискорити процес прийняття клінічних рішень.

Головна мета проектування полягає в розробці високонадійного, точного та обчислювально ефективного програмного забезпечення, яке реалізує двостадійний діагностичний пайплайн (бінарна класифікація з генерацією карт активації та наступна піксельна сегментація) і забезпечує оптимальне співвідношення між технічними характеристиками системи та витратами на її створення й експлуатацію.

Для досягнення поставленої мети програмний продукт повинен задовольняти такі основні технічні вимоги:

- 1) забезпечення стабільного функціонування програмного забезпечення на персональних комп'ютерах зі стандартним набором апаратних компонентів та поширеними операційними системами.
- 2) створення інтуїтивно зрозумілого інтерфейсу користувача, що забезпечує легкість навігації, наочність представлення результатів та мінімізує час, необхідний для освоєння системи.
- 3) висока швидкість обробки вхідних даних, оптимізація обчислювальних процесів та забезпечення доступу до результатів аналізу й супутньої інформації в режимі реального часу.
- 4) реалізація модульної архітектури програмного продукту, яка дозволяє зручно розширювати функціональні можливості системи, адаптувати її до зростаючих обсягів даних, а також забезпечує простоту технічного обслуговування, тестування та оновлення.

- 5) мінімізація капітальних та експлуатаційних витрат на впровадження, розгортання та подальше супроводження програмного продукту в цільовому середовищі.

4.2 Обґрунтування функцій програмного продукту

Головною функцією (об'єктом аналізу) є F_0 — розробка програмного продукту для автоматизованої діагностики переломів кісток на рентгенівських знімках.

Для реалізації головної функції виділено наступні базові підфункції:

- 1) F_1 — вибір мови програмування та інструментарію для машинного навчання.
- 2) F_2 — вибір архітектури моделі для стадії бінарної класифікації та генерації карт активацій (локалізації).
- 3) F_3 — вибір архітектури моделі для стадії піксельної сегментації переломів.
- 4) F_4 — вибір способу візуалізації результатів (інтерфейсу) для медичного персоналу.

Кожна з виділених підфункцій може бути реалізована різними способами. Морфологічну карту зображено на рисунку 4.1. Розглянемо можливі варіації для кожної з них:

Підфункція F_1 — вибір мови програмування та інструментарію для машинного навчання:

- 1) використання мови Python разом із фреймворком PyTorch;
- 2) використання мови C++ із бібліотекою LibTorch.

Підфункція F_2 — вибір архітектури моделі для стадії бінарної класифікації та генерації карт активацій (локалізації):

- 1) класичні глибокі згорткові нейромережі (CNN);

2) сучасні ієрархічні архітектури з елементами Transformers.

Підфункція F_3 – вибір архітектури моделі для стадії піксельної сегментації переломів:

- 1) спеціалізовані медичні енкодер-декодер архітектури;
- 2) фундаментальні універсальні моделі сегментації (MedSAM).

Підфункція F_4 – вибір способу візуалізації результатів (інтерфейсу) для медичного персоналу:

- 1) інтерактивний веб-інтерфейс;
- 2) локальний десктоп-додаток.

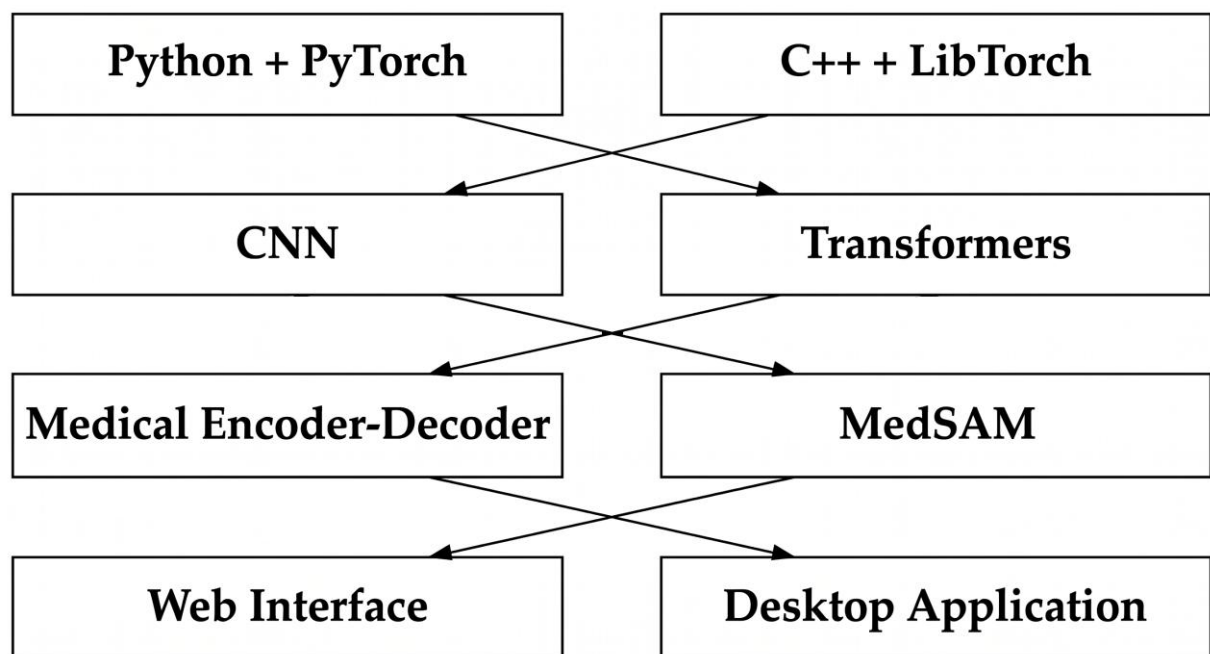


Рисунок 4.1 – Морфологічна карта

Для вибору найбільш оптимальних варіантів реалізації кожної підфункції складемо позитивно-негативну матрицю (таблиця 4.1), в якій наведено переваги та недоліки кожного з можливих підходів.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Швидка розробка, велика кількість бібліотек для обробки медичних зображень	Нижча швидкість виконання коду, проблеми з пам'яттю
	Б	Максимальна продуктивність, ефективне управління пам'яттю	Складність написання коду, менша кількість готових рішень
F2	А	Низькі вимоги до обчислювальних ресурсів	Можуть поступатися новим моделям у здатності розуміти глобальний контекст.
	Б	Вища точність, краще розпізнавання складних патернів.	Висока ресурсоемність, тривалий час інференсу
F3	А	Економне споживання пам'яті, оптимізація під сегментацію дрібних структур	Вузька спеціалізація, необхідність ретельного налаштування архітектури
	Б	Висока здатність до узагальнення, універсальність	Надмірна архітектурна складність, потреба у значних ресурсних потужностях
F4	А	Кросплатформність, легкість оновлення та підтримки.	Залежність від браузера та якості мережевого з'єднання
	Б	Повна автономність роботи на локальній машині	Складність дистрибуції та розгортання

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути через те, що вони не відповідають поставленим перед програмним продуктом задачам або є надмірно ресурсозатратними.

Функція F1. Перевага надається варіанту А (Python) завдяки наявності великої кількості готових бібліотек, що значно пришвидшує розробку. Варіант Б відкидається.

Функція F2. Обидва варіанти (А і Б) є допустимими, оскільки дозволяють порівняти різні підходи до балансу між швидкістю та точністю класифікації.

Функція F3. Обидва варіанти (А і Б) залишаються для порівняння спеціалізованого медичного підходу з універсальними фундаментальними моделями.

Функція F4. Перевага надається варіанту А (веб-інтерфейс), оскільки він забезпечує кросплатформність та легкість доступу без встановлення додаткового ПЗ. Варіант Б відкидається.

Таким чином, розглядатимемо наступні варіанти комплексної реалізації програмного продукту:

$$F_1A - F_2A - F_3A - F_4A$$

$$F_1A - F_2B - F_3B - F_4A$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначимо основні параметри для оцінки та вибору оптимального варіанту реалізації програмного продукту. Оскільки обидва найбільш перспективні варіанти базуються на спільному

базовому технологічному стеку, ключові відмінності полягають у характеристиках обраних нейромережових архітектур для класифікації та сегментації.

Для характеристики програмного продукту будемо використовувати наступні параметри:

- 1) X_1 – швидкодія мови програмування;
- 2) X_2 – об'єм пам'яті для обчислень та збереження даних;
- 3) X_3 – час обробки даних;
- 4) X_4 – потенційний об'єм програмного коду.

Граничні значення кожного параметра визначаються виходячи з реальних умов розгортання системи в медичному середовищі: апаратних обмежень робочих станцій, вимог до часу відповіді при інтерактивній роботі лікаря зі знімком. Конкретні значення параметрів наведено у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовне позначення	Одиниці виміру	Гірше	Середнє	Краще
Швидкодія мови програмування	X_1	оп/мс	600	800	1500
Об'єм пам'яті	X_2	МБ	700	400	150
Час обробки даних	X_3	мс	175	85	45
Об'єм програмного коду	X_4	Кількість рядків коду	4200	2500	1500

За даними таблиці 4.2 побудовано графічні залежності параметрів програмного продукту (рисунок 4.2 – рисунок 4.5).

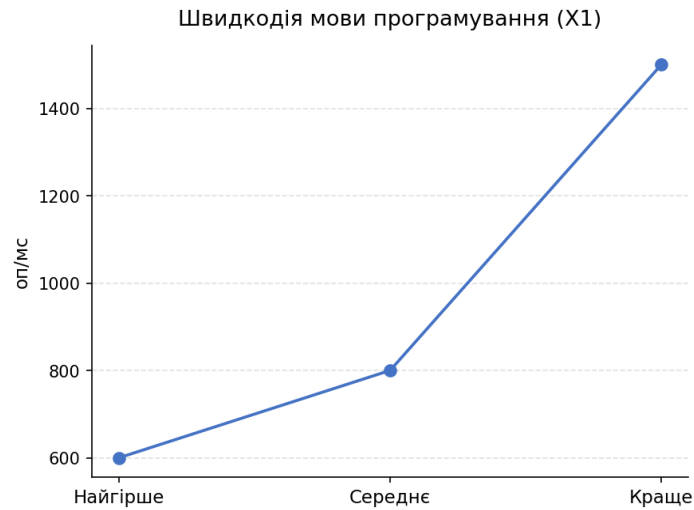


Рисунок 4.2 – Швидкодія мови програмування.

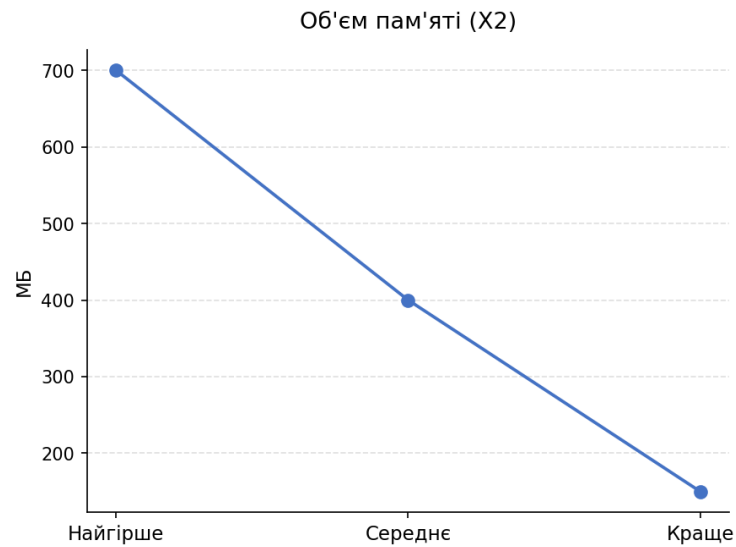


Рисунок 4.3 – Об'єм пам'яті.

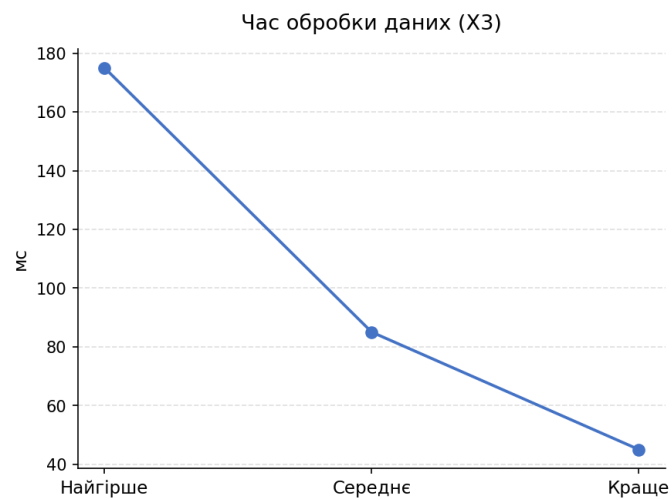


Рисунок 4.4 – Час обробки даних.

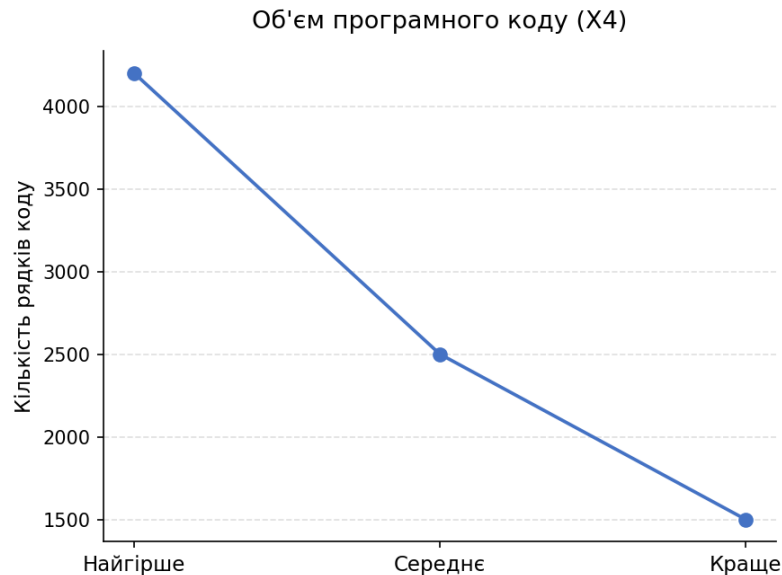


Рисунок 4.5 – Об'єм програмного коду.

4.4 Аналіз експертного оцінювання параметрів

Для визначення значущості кожного з параметрів системи автоматизованої діагностики переломів кісток залучено експертну групу з 7 фахівців у сфері медичної візуалізації, комп'ютерного зору та розробки програмного забезпечення для медичних застосунків. На основі власного досвіду та професійних знань кожен експерт присвоїв параметрам ранги відповідно до їхньої важливості для досягнення поставленої мети.

Ступінь важливості кожного параметру оцінювалась з позиції конкретної цілі – розробка програмного продукту, який забезпечує високу точність діагностики переломів при мінімальних обчислювальних витратах та прийнятному часі відповіді в умовах реальної клінічної практики.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати експертного ранжування

Позначення параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X1	2	1	1	1	2	2	1	10	-7,5	56,25
X2	3	3	2	3	1	3	3	18	0,5	0,25
X3	1	2	3	2	3	1	2	14	-3,5	12,25
X4	4	4	4	4	4	4	4	28	10,5	110,25
	10	10	10	10	10	10	10	70	0	179

Для перевірки ступеня достовірності експертних оцінок визначимо параметри.

Загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70$$

де N – число експертів;

n – кількість параметрів.

Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5$$

Відхилення суми рангів кожного параметра від середньої:

$$\Delta_i = R_i - T$$

$$\Delta_1 = 11 - 17,5 = -7,5;$$

$$\Delta_2 = 16 - 17,5 = 0,5;$$

$$\Delta_3 = 15 - 17,5 = -3,5;$$

$$\Delta_4 = 28 - 17,5 = 10,5.$$

Сума відхилень: $-7,5 + 0,5 - 3,5 + 10,5 = 0$

Загальна сума квадратів відхилень:

$$S = \sum_{i=1}^N \Delta_i^2 = 179.$$

Коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0,731 > W_k = 0,67$$

Отримане значення коефіцієнта узгодженості перевищує нормативне, тому результати ранжування вважаються достовірними.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4

Ступінь переваги i -го параметра над j -тим визначається числовим значенням a_{ij} по формулі:

$$a_{ij} = \begin{cases} 1,5 & \text{при } X_i > X_j \\ 1,0 & \text{при } X_i = X_j \\ 0,5 & \text{при } X_i < X_j \end{cases}$$

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	>	<	<	<	0,5
X1 і X3	>	<	<	<	<	>	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	<	>	<	>	>	>	1,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

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

Для кожного параметра зробимо розрахунок вагомості K_{vi} за наступними формулами:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^n b_i}$$

$$b_i' = \sum_{i=1}^N a_{ij} b_j$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	b_i	K_{Bi}	b_i^1	K_{Bi}^1
X1	1	1,5	1,5	1,5	5,5	0,3438	21,25	0,3603
X2	0,5	1	0,5	1,5	3,5	0,2188	12,25	0,2076
X3	0,5	1,5	1	1,5	4,5	0,2812	16,25	0,2754
X4	0,5	0,5	0,5	1	2,5	0,1562	9,25	0,1568
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту реалізації окремо. Коефіцієнт технічного рівня для кожного варіанта розраховується за формулою:

$$K_K(j) = \sum_{i=1}^n K_{Bi,j} B_{i,j}$$

де K_{Bi} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра;

n – кількість параметрів.

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}]$$

Бальні оцінки визначаються з графіків (рис. 4.2 – 4.5) на основі абсолютних значень параметрів для кожного варіанту.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП.

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F ₂	А	X1	1500	95	0,3603	34,22
		X2	150	92	0,2076	19,09
		X3	45	90	0,2754	24,79
		X4	1500	88	0,1568	13,80
F ₃	Б	X1	800	55	0,3603	19,81
		X2	400	52	0,2076	10,80
		X3	85	50	0,2754	13,77
		X4	2500	45	0,1568	7,06

За формулою визначаємо рівень якості кожного варіанту:

$$K_{K1} = 34,22 + 19,09 + 24,79 + 13,80 = 91,91$$

$$K_{K2} = 19,81 + 10,80 + 13,77 + 7,06 = 51,44$$

За результатами розрахунку, варіант 1 має значно вищий коефіцієнт рівня якості. Таким чином, для подальшої реалізації обрано його (Python + CNN + енкодер-декодер архітектури + веб-інтерфейс).

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки системи виконаємо розрахунок трудомісткості. Обидва варіанти охоплюють два завдання:

- 1) розробка проекту програмного продукту;
- 2) розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, оскільки передбачає використання принципово нових методів розробки – двостадійного діагностичного пайплайну на основі глибокого навчання, що не має прямих аналогів у клінічній практиці. Завдання 2 відноситься до групи Б, оскільки являє собою розробку оригінальної системи, яка не має готових аналогів. За складністю алгоритмів обидва завдання належать до групи 1, адже реалізують алгоритми оптимізації та моделювання нейромережових систем.

Загальна трудомісткість обчислюється за формулою:

$$T_O = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}$$

де T_P – базова трудомісткість;

K_{Π} – поправочний коефіцієнт;

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

K_M – коефіцієнт рівня мови програмування;

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

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

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 90$ людино-днів. Для завдання 2: $T_P = 64$ людино-днів

Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.7$. Для завдання 2: $K_{\Pi} = 1.021$

Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки розробка ведеться мовою Python, то $K_M = 0.85$. При розробці

використовуються стандартні модулі та пакети прикладних програм: для варіанту I $K_{CT} = 0.8$, для варіанту II $K_{CT} = 0.7$. Коефіцієнт стандартного математичного забезпечення: $K_{CT.M} = 1$.

Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 * 1.7 * 1 * 0.85 * 0.8 * 1 = 104,04 \text{ людино днів}$$

$$T_2 = 64 * 1.021 * 1 * 0.85 * 0.8 * 1 = 44,44 \text{ людино днів}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість.

$$T_I = (104,04 + 44,44) \cdot 8 = 1187,84 \text{ людино-годин}$$

$$T_{II} = (104,04 + 44,44 + 14,8) \cdot 8 = 1306,24 \text{ людино-годин}$$

У розробці беруть участь розробник з місячним окладом 20 000 грн та науковий керівник з окладом 40 000 грн. Середня погодинна ставка, рахується за формулою:

$$CЧ = \frac{M}{T_m \cdot t} \text{ грн.}$$

де M – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$C_{ч} = \frac{20\,000 + 40\,000}{2 \cdot 21,1 \cdot 8} = 177,75 \text{ грн/год}$$

Заробітна плата вираховується по даній формулі:

$$CЗП = C_{ч} \cdot T_i \cdot K_d$$

де $C_{ч}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K_d – норматив, який враховує додаткову заробітну плату.

Заробітна плата з урахуванням додаткової ($K_d = 1,2$):

$$1) \quad C_{ЗП} = 177,75 \cdot 1\,187,84 \cdot 1,2 = 253\,274,59 \text{ грн;}$$

$$2) \quad C_{ЗП} = 177,75 \cdot 1\,306,28 \cdot 1,2 = 284\,916,31 \text{ грн.}$$

Відрахування на єдиний соціальний внесок (22%):

$$1) \quad C_{\text{ВІД}} = 253\,274,59 \cdot 0,22 = 55\,720,41 \text{ грн};$$

$$2) \quad C_{\text{ВІД}} = 284\,916,31 \cdot 0,22 = 62\,681,59 \text{ грн.}$$

Далі визначимо витрати на оплату однієї машино-години. ЕОМ обслуговує розробника з окладом 20 000 грн, коефіцієнт зайнятості $K_3 = 0,2$:

$$C_{\text{Г}} = 12 \cdot 20\,000 \cdot 0,2 = 48\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} (\text{ЕОМ}) = 48\,000 \cdot (1 + 0,2) = 57\,600 \text{ грн}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} (\text{ЕОМ}) = 57\,600 \cdot 0,22 = 12\,672 \text{ грн}$$

Амортизаційні відрахування при нормі амортизації 25% та вартості GPU-станції 60 000 грн:

$$C_{\text{А}} = K_{\text{ТМ}} \cdot K_{\text{А}} \cdot C_{\text{ПР}} = 1,15 \cdot 0,25 \cdot 60\,000 = 17\,250 \text{ грн}$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$K_{\text{А}}$ – річна норма амортизації;

$C_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику:

$$C_{\text{Р}} = K_{\text{ТМ}} \cdot C_{\text{ПР}} \cdot K_{\text{Р}} = 1,15 \cdot 60\,000 \cdot 0,05 = 3\,450 \text{ грн}$$

де $K_{\text{Р}}$ – відсоток витрат на поточні ремонти.

Ефективний річний фонд часу роботи ПК:

$$T_{\text{ЕФ}} = (D_{\text{К}} - D_{\text{В}} - D_{\text{С}} - D_{\text{Р}}) \cdot t_{\text{З}} \cdot K_{\text{В}} = 1\,645,6 \text{ годин}$$

де $D_{\text{К}}$ – календарна кількість днів у році;

$D_{\text{В}}, D_{\text{С}}$ – відповідно кількість вихідних та святкових днів;

$D_{\text{Р}}$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$K_{\text{В}}$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на електроенергію (тариф 13,64 грн/кВт·год, потужність 0,2 кВт):

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_{\text{С}} \cdot K_{\text{З}} \cdot C_{\text{ЕН}} = 1\,645,6 \cdot 0,4 \cdot 0,2 \cdot 13,64 = 1\,795,68 \text{ грн}$$

де $N_{\text{С}}$ – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу,

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

Накладні витрати:

$$C_H = \Pi_{ПР} \cdot 0,67 = 60\,000 \cdot 0,67 = 40\,200 \text{ грн}$$

Річні експлуатаційні витрати:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_H$$

$$C_{ЕКС} = 57\,600 + 12\,672 + 17\,250 + 3\,450 + 1\,795,68 + 40\,200 = 132\,967,68 \text{ грн}$$

Собівартість однієї машино-години:

$$C_{М-Г} = C_{ЕКС} / T_{ЕФ} = 132\,967,68 / 1\,645,6 = 80,80 \text{ грн/год}$$

Витрати на машинний час:

$$C_M = C_{М-Г} \cdot T$$

$$1) \quad C_M = 80,80 \cdot 1\,187,84 = 95\,977,47 \text{ грн};$$

$$2) \quad C_M = 80,80 \cdot 1\,336,28 = 107\,971,42 \text{ грн.}$$

Накладні витрати (67% від заробітної плати):

$$C_H = C_{ЗП} \cdot 0,67$$

$$1) \quad C_H = 253\,274,59 \cdot 0,67 = 169\,693,98 \text{ грн};$$

$$2) \quad C_H = 284\,916,31 \cdot 0,67 = 190\,893,93 \text{ грн.}$$

Повна вартість розробки ПП:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_M + C_H$$

$$1) \quad C_{ПП} = 253\,274,59 + 55\,720,41 + 95\,977,47 + 169\,693,98 = 574\,666,45 \text{ грн};$$

$$2) \quad C_{ПП} = 284\,916,31 + 62\,681,59 + 107\,971,42 + 190\,893,93 = 646\,463,25 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП за техніко-економічним рівнем

Для остаточного вибору оптимального варіанту реалізації розраховується коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕРj} = K_{Kj} / C_{Фj}$$

де K_{kj} – коефіцієнт рівня якості j -го варіанту; $C_{\Phi j}$ – вартість розробки j -го варіанту, грн

$$K_{\text{TEP1}} = 91,91 / 574\,666,45 = 1,599 \cdot 10^{-4}$$

$$K_{\text{TEP2}} = 51,44 / 646\,463,25 = 0,796 \cdot 10^{-4}$$

Порівнявши отримані значення, бачимо що перший варіант має вищий коефіцієнт техніко-економічного рівня. Це означає що Варіант 1 забезпечує кращу якість при менших витратах на розробку.

Таким чином, оптимальним для реалізації є Варіант 1 – система на основі мови програмування Python із фреймворком PyTorch, класичних глибоких згорткових нейромереж (CNN) для стадії бінарної класифікації та генерації карт активацій, спеціалізованих медичних енкодер-декодер архітектур для стадії піксельної сегментації переломів, а також інтерактивного веб-інтерфейсу для медичного персоналу. Обрана архітектура забезпечує необхідну точність діагностики при оптимальних обчислювальних витратах, прийнятному часі відповіді системи в умовах реальної клінічної практики та найкраще співвідношення між технічним рівнем і вартістю розробки.

4.8 Висновки до розділу 4

У цьому розділі проведено функціонально-вартісний аналіз програмного продукту для автоматизованої діагностики переломів кісток на рентгенівських знімках. За результатами дослідження зроблено такі висновки.

Визначено функціональну структуру продукту: на основі аналізу вимог виділено 4 ключові підфункції – вибір мови програмування та інструментарію для машинного навчання (F1), вибір архітектури моделі для стадії бінарної класифікації та генерації карт активацій (F2), вибір

архітектури моделі для стадії піксельної сегментації переломів (F3), вибір способу візуалізації результатів для медичного персоналу (F4). На основі позитивно-негативної матриці сформовано два варіанти комплексної реалізації: Варіант 1 (Python + CNN + енкодер-декодер архітектури + веб-інтерфейс) та Варіант 2 (Python + Transformer-архітектури + MedSAM + веб-інтерфейс).

Проведено експертне оцінювання параметрів: залучено групу з 7 фахівців у сфері медичної візуалізації, комп'ютерного зору та розробки програмного забезпечення. Встановлено, що найбільш пріоритетним параметром є швидкодія мови програмування ($K_B = 0,3603$). Отриманий коефіцієнт узгодженості думок експертів $W = 0,731$ перевищує нормативне значення $W_K = 0,67$, що підтверджує достовірність результатів ранжування.

Оцінено технічний рівень варіантів: розрахунок коефіцієнтів рівня якості засвідчив суттєву перевагу Варіанту 1 ($K_{K1} = 91,91$ проти $K_{K2} = 51,44$), що зумовлено вищою швидкодією, меншим споживанням оперативної пам'яті, коротшим часом обробки даних та меншим обсягом програмного коду порівняно з ресурсоємними Transformer-рішеннями та фундаментальними моделями сегментації.

Виконано економічний розрахунок вартості розробки: повна вартість розробки Варіанту 1 при місячних окладах розробника 20 000 грн та наукового керівника 40 000 грн склала 574 666,45 грн, що є на 71 796,80 грн менше, ніж вартість розробки Варіанту 2 (646 463,25 грн).

Обрано оптимальний варіант за критерієм техніко-економічного рівня: $K_{\text{ТЕР}_1} = 1,599 \cdot 10^{-4}$ проти $K_{\text{ТЕР}_2} = 0,796 \cdot 10^{-4}$. Варіант 1 забезпечує вдвічі вищий техніко-економічний рівень, тобто кращу якість при менших витратах на розробку. Саме цей варіант – система на основі Python із PyTorch, класичних CNN, спеціалізованих медичних енкодер-декодер архітектур та веб-інтерфейсу – прийнято до реалізації у практичній частині дипломного проєкту.

ВИСНОВКИ

У роботі вирішено задачу автоматизованої діагностики переломів кісток за рентгенівськими знімками шляхом побудови двостадійного пайплайну «класифікація і сегментація» з інтегрованим механізмом інтерпретованої локалізації на основі теплових карт активацій.

За результатами аналізу предметної області встановлено, що переломи кісток є одним із найпоширеніших видів травм у світі з щорічною кількістю нових випадків близько 178 мільйонів, і ця тенденція зростатиме через старіння населення та поширення остеопорозу. Незважаючи на те що рентгенографія залишається золотим стандартом первинної діагностики завдяки доступності та швидкості, від 1,1% до 3,7% переломів пропускаються лікарями при першому зверненні, причому 49% помилок концентруються у вечірньо-нічний час і 86% із них безпосередньо впливають на тактику лікування. Це підтверджує, що людський фактор є структурною вразливістю системи, а не випадковим відхиленням, і саме тут автоматизовані системи мають найбільший потенціал клінічного впливу.

Виявлено п'ять принципових труднощів, що відрізняють задачу сегментації переломів від стандартних задач комп'ютерного зору: проекційне накладання анатомічних структур, вкрай низький піксельний контраст лінії злому, висока морфологічна варіабельність форми ушкодження, глибокий дисбаланс класів на рівні пікселів та хронічний дефіцит публічних датасетів із якісною піксельною розміткою. Єдиним придатним для задачі сегментації публічним ресурсом є датасет FracAtlas (2023), що містить лише 717 анотованих випадків із піксельною розміткою, – це формує фундаментальне обмеження за обсягом даних і визначає необхідність підходів, ефективних в умовах малих вибірок.

На етапі підготовки даних усі зображення приведено до розміру 224×224 пікселів із нормалізацією та стандартизацією відносно статистик

ImageNet. За результатами порівняльного аналізу п'яти методів покращення контрасту обрано CLAHE Grayscale як найбільш ефективний для виявлення переломних ліній і трабекулярних структур без артефактів і без ризику втрати анатомічних деталей. Для розширення навчальної вибірки застосовано онлайн-аугментацію з навмисно слабкими перетвореннями з метою уникнення спотворення тонких структур.

На етапі класифікації навчено три архітектури – ResNet18, ResNet50 та ConvNeXtV2-Tiny – із застосуванням стратегії поетапного розморожування та зваженої крос-ентропії для компенсації дисбалансу класів 1:4,7. Усі три моделі продемонстрували стабільні результати в діапазоні Accuracy 92,3–93,5% та AUC 95,5–96,0%, що підтверджує надійність обраного підходу до навчання. Найкращу якість показала ConvNeXtV2-Tiny з Accuracy = 93,5% та F1 = 88,8%. За результатами порівняння трьох методів візуалізації активацій (GradCAM, GradCAM++, LayerCAM) обрано LayerCAM як такий, що забезпечує найточнішу просторову локалізацію переломної зони завдяки більш деталізованим і локалізованим картам активацій із чіткішими межами.

На етапі сегментації реалізовано та навчено п'ять архітектур – U-Net, U-Net++, DeepLabV3+, Attention DeepLabV3+ та Attention U-Net++ – за трьома стратегіями формування вхідних даних. Порівняння стратегій підтвердило, що якість попередньої локалізації є визначальним фактором точності сегментації. При навчанні на повних знімках без локалізації всі моделі показують Dice у діапазоні 0,507–0,536, що відображає складність одночасного пошуку та виділення переломної зони на тлі глибокого піксельного дисбалансу. Використання автоматичної САМ-локалізації (LayerCAM Crops) забезпечує суттєвий приріст якості до Dice = 0,596–0,735 без жодного ручного втручання, що підтверджує практичну цінність запропонованого пайплайну. Еталонна локалізація за анотованими рамками (GT Crops) дає найвищі результати – Dice до 0,817 – і є верхньою межею точності, недосяжною в реальних умовах без ручної розмітки.

Серед усіх архітектур стабільне лідерство в усіх трьох стратегіях демонструє DeepLabV3+, що підтверджує ефективність мультимасштабного аналізу через ASPP-модуль при роботі з рентгенівськими знімками. Моделі з механізмом уваги (Attention DeepLabV3+ та Attention U-Net++) показують вищі значення Recall, що є перевагою у клінічному контексті, де пропуск перелому є критичнішою помилкою, ніж хибно-позитивний результат.

Таким чином, запропонований підхід – класифікація з САМ-локалізацією як проміжний етап перед сегментацією – є обґрунтованим і практично застосовним. Автоматична локалізація через LayerCAM суттєво підвищує якість сегментації порівняно з роботою на повних знімках і при цьому не потребує додаткової ручної розмітки, що є критично важливим в умовах хронічного дефіциту анотованих медичних даних. Подальші дослідження доцільно спрямувати на розширення навчальної вибірки, вдосконалення механізму САМ-локалізації та клінічну валідацію системи на даних із нових медичних закладів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Johnell O., Kanis J. A. An estimate of the worldwide prevalence and disability associated with osteoporotic fractures. *Osteoporosis International*. 2006. Vol. 17, No. 12. P. 1726–1733. DOI: 10.1007/s00198-006-0172-4.
2. GBD 2019 Fracture Collaborators. Global, regional, and national burden of bone fractures in 204 countries and territories, 1990–2019. *The Lancet Healthy Longevity*. 2021. Vol. 2, No. 9. P. e580–e592. DOI: 10.1016/S2666-7568(21)00172-0.
3. Guly H. R. Diagnostic errors in an accident and emergency department. *Emergency Medicine Journal*. 2001. Vol. 18, No. 4. P. 263–269. URL: <https://pubmed.ncbi.nlm.nih.gov/11435359/> (last accessed: 10.05.2025).
4. Pinto A. et al. Missed diagnoses and misdiagnoses of trauma patients. *European Journal of Radiology*. 2006. Vol. 59, No. 1. P. 2–9. DOI: 10.23750/abm.v89i1-s.7015.
5. Juocevicius A. et al. Delayed diagnosis of fractures in patients with multiple traumas. *Medicina*. 2007. Vol. 43, No. 2. P. 119–124. DOI: 10.1111/j.1553-2712.1996.tb03527.x.
6. Lindsey R. et al. Deep neural network improves fracture detection by clinicians. *Proceedings of the National Academy of Sciences*. 2018. Vol. 115, No. 45. P. 11591–11596. URL: <https://pubmed.ncbi.nlm.nih.gov/30348771/> (last accessed: 14.05.2025).
7. Löffler M. T. et al. FracAtlas: A Dataset for Fracture Classification, Localization and Segmentation of Musculoskeletal Radiographs. *Scientific Data*. 2023. Vol. 10. Article 521. DOI: 10.1038/s41597-023-02432-4.
8. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge: MIT Press, 2016. 775 p. URL: <https://www.deeplearningbook.org> (last accessed: 15.05.2025).

9. Jiang P.-T. et al. LayerCAM: Exploring Hierarchical Class Activation Maps for Localization. *IEEE Transactions on Image Processing*. 2021. Vol. 30. P. 5875–5888. URL: https://mftp.mmcheng.net/Papers/21TIP_LayerCAM.pdf (last accessed: 19.05.2025).
10. He K. et al. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, Las Vegas, Nevada, USA, 27–30 June 2016: Piscataway, NJ: IEEE, 2016. P. 770–778. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf (last accessed: 20.05.2025).
11. Liu Z. et al. A ConvNet for the 2020s. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2022)*, New Orleans, Louisiana, USA, 19–24 June 2022: Piscataway, NJ: IEEE, 2022. P. 11976–11986. DOI: 10.48550/arXiv.2201.03545.
12. Woo S. et al. ConvNeXt V2: Co-designing and Scaling ConvNeXts with Masked Autoencoders. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2023)*, Vancouver, BC, Canada, 18–22 June 2023: Piscataway, NJ: IEEE, 2023. P. 16133–16142. DOI: 10.48550/arXiv.2301.00808.
13. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Proceedings of the 18th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015)*, Munich, Germany, 5–9 October 2015: Cham: Springer, 2015. P. 234–241. DOI: 10.48550/arXiv.2301.00808.
14. Zhou Z. et al. UNet++: A Nested U-Net Architecture for Medical Image Segmentation. *Proceedings of the Deep Learning in Medical Image Analysis Workshop (MICCAI 2018)*, Granada, Spain, 20 September 2018: Cham: Springer, 2018. P. 3–11. DOI: 10.1007/978-3-030-00889-5_1.
15. Chen L.-C. et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *Proceedings of the 15th European*

- Conference on Computer Vision (ECCV 2018)*, Munich, Germany, 8–14 September 2018: Cham: Springer, 2018. P. 833–851.DOI: 10.48550/arXiv.1802.02611.
16. Oktay O. et al. Attention U-Net: Learning Where to Look for the Pancreas. *Proceedings of the Medical Imaging with Deep Learning Conference (MIDL 2018)*, Amsterdam, Netherlands, 4–6 July 2018: 2018. DOI: 10.48550/arXiv.1804.03999.
 17. Woo S. et al. CBAM: Convolutional Block Attention Module. *Proceedings of the 15th European Conference on Computer Vision (ECCV 2018)*, Munich, Germany, 8–14 September 2018: Cham: Springer, 2018.DOI: 10.48550/arXiv.1807.06521.
 18. Milletari F., Navab N., Ahmadi S.-A. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. *Proceedings of the 4th International Conference on 3D Vision (3DV 2016)*, Stanford, California, USA, 25–28 October 2016: Piscataway, NJ: IEEE, 2016. P. 565–571.DOI: 10.48550/arXiv.1606.04797.
 19. Salehi S. S. M., Erdogmus D., Gholipour A. Tversky Loss Function for Image Segmentation Using 3D Fully Convolutional Deep Networks. *Proceedings of the 8th International Workshop on Machine Learning in Medical Imaging (MLMI 2017)*, Quebec City, Canada, 10 September 2017: Cham: Springer, 2017. P. 379–387.DOI: 10.48550/arXiv.1706.05721.
 20. Selvaraju R. R. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *Proceedings of the IEEE International Conference on Computer Vision (ICCV 2017)*, Venice, Italy, 22–29 October 2017: Piscataway, NJ: IEEE, 2017. P. 618–626.DOI: 10.48550/arXiv.1610.02391.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО МОДУЛЮ

```
# classify_models.py
import os
import gc
import math
import json
import random
import warnings
from pathlib import Path
from dataclasses import dataclass, asdict
from typing import Dict, List, Tuple, Optional, Any
from collections import Counter, defaultdict

import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

from tqdm.auto import tqdm
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, roc_curve,
    confusion_matrix, classification_report,
)

import torch
import torch.nn as nn
import torch.optim as optim
from torch.cuda.amp import autocast, GradScaler
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from torchvision import transforms, models

import timm
from IPython.display import display

warnings.filterwarnings("ignore")

@dataclass
class CFG:
    seed: int = 42
    image_size: int = 224
    batch_size: int = 32
    num_workers: int = 2
    epochs: int = 15
    patience: int = 5
```

```

lr: float = 1e-4
weight_decay: float = 1e-4
use_amp: bool = True
use_clahe: bool = True
use_weighted_sampler: bool = True
use_weighted_loss: bool = False
drop_exact_duplicate_fingerprints: bool = True
fingerprint_size: int = 8
positive_class_name: str = "fracture"
negative_class_name: str = "normal"
convnextv2_model_name: str = "convnextv2_tiny.fcmae_ft_in22k_in1k"
data_dir: Path = Path("data")
checkpoints_dir: Path = Path("checkpoints_classifier_v2")
results_dir: Path = Path("classifier_results_v2")

cfg = CFG()

random.seed(cfg.seed)
np.random.seed(cfg.seed)
torch.manual_seed(cfg.seed)
torch.cuda.manual_seed_all(cfg.seed)
torch.backends.cudnn.benchmark = True

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("DEVICE:", DEVICE)

cfg.data_dir.mkdir(parents=True, exist_ok=True)
cfg.checkpoints_dir.mkdir(parents=True, exist_ok=True)
cfg.results_dir.mkdir(parents=True, exist_ok=True)

# — Label / path utilities —————

IMG_EXTS = {".jpg", ".jpeg", ".png", ".bmp", ".tif", ".tiff", ".webp"}

def is_image_file(path: Path) -> bool:
    return path.is_file() and path.suffix.lower() in IMG_EXTS

def normalize_text(s: Any) -> str:
    return str(s).lower().replace("_", " ").replace("-", " ").strip()

def parse_label_value(value: Any) -> Optional[int]:
    if pd.isna(value):
        return None
    if isinstance(value, (int, float, np.integer, np.floating)):
        return int(float(value) > 0)
    s = normalize_text(value)
    if s in {"0", "false", "no", "negative", "normal", "not fractured",
            "non fractured", "nonfractured", "no fracture"}:
        return 0
    if s in {"1", "true", "yes", "positive", "fracture", "fractured", "broken"}:
        return 1

```

```

    if "not" in s and "fract" in s: return 0
    if "non" in s and "fract" in s: return 0
    if "normal" in s or "negative" in s: return 0
    if "fract" in s or "broken" in s or "positive" in s: return 1
    return None

def infer_label_from_near_parent(path: Path) -> Optional[int]:
    near_parts = [normalize_text(p.name) for p in list(path.parents)[:5]]
    for part in near_parts:
        if ("not" in part and "fract" in part) or ("non" in part and "fract" in part):
            return 0
        if part in {"normal", "negative", "no fracture", "not fractured",
                    "non fractured", "nonfractured"}:
            return 0
    for part in near_parts:
        if part in {"fractured", "fracture", "positive", "broken"}:
            return 1
        if "fract" in part and "not" not in part and "non" not in part:
            return 1
    return None

def infer_split_from_path(path: Path) -> Optional[str]:
    for parent in list(path.parents)[:8]:
        p = normalize_text(parent.name)
        if p in {"train", "training"}: return "train"
        if p in {"val", "valid", "validation"}: return "val"
        if p in {"test", "testing"}: return "test"
    return None

def should_skip_image_path(path: Path) -> bool:
    joined = "/".join(normalize_text(p.name) for p in path.parents)
    return any(k in joined for k in
               ["annotation", "coco", "mask", "segmentation", "utility"])

def _try_read_table(path: Path) -> Optional[pd.DataFrame]:
    try:
        if path.suffix.lower() == ".csv": return pd.read_csv(path)
        if path.suffix.lower() == ".tsv": return pd.read_csv(path, sep="\t")
        if path.suffix.lower() in {".xlsx", ".xls"}: return pd.read_excel(path)
    except Exception:
        return None

# — Dataset scanning —————

def build_label_map_from_tables(root: Path) -> Dict[str, int]:
    table_paths = [p for ext in [".csv", ".tsv", ".xlsx", ".xls"]
                   for p in Path(root).rglob(ext)]
    label_map: Dict[str, int] = {}
    name_cands = ["image_id", "file_name", "filename", "image", "name", "path"]
    label_cands = ["label", "class", "target", "fractured", "fracture", "has_fracture"]
    for tp in table_paths:
        df_t = _try_read_table(tp)

```

```

    if df_t is None or df_t.empty: continue
    cols = {normalize_text(c): c for c in df_t.columns}
    nc = next((cols[c] for c in name_cands if c in cols), None)
    lc = next((cols[c] for c in label_cands if c in cols), None)
    if nc is None or lc is None: continue
    for _, row in df_t[[nc, lc]].dropna().iterrows():
        lbl = parse_label_value(row[lc])
        if lbl is None: continue
        name = Path(str(row[nc]).replace("\\", "/")).name
        label_map[name.lower()] = lbl
        label_map[Path(name).stem.lower()] = lbl
    if label_map:
        print(f"Label map: {tp.name} | mapped={len(label_map)}")
    return label_map

def scan_dataset_generic(root: Path, source_name: str) -> pd.DataFrame:
    root = Path(root)
    if not root.exists():
        print(f"Missing root: {root}")
        return pd.DataFrame(columns=["path", "label", "label_name", "source"])
    label_map = build_label_map_from_tables(root)
    rows = []
    for path in root.rglob("*"):
        if not is_image_file(path) or should_skip_image_path(path): continue
        label = infer_label_from_near_parent(path)
        if label is None:
            label = label_map.get(path.name.lower()) or label_map.get(path.stem.lower())
        if label is None: continue
        rows.append({
            "path": str(path),
            "file_name": path.name,
            "stem": path.stem.lower(),
            "label": int(label),
            "label_name":      cfg.positive_class_name      if      int(label)      ==      1      else
cfg.negative_class_name,
            "source": source_name,
            "external_split": infer_split_from_path(path),
        })
    df_out = pd.DataFrame(rows).drop_duplicates(subset=["path"]).reset_index(drop=True)
    print(f"{source_name}: {len(df_out)} labeled images")
    return df_out

# — Preprocessing / split —————

def image_fingerprint(path: str, hash_size: int = 8) -> str:
    try:
        img = Image.open(path).convert("L").resize(
            (hash_size + 1, hash_size), Image.Resampling.LANCZOS)
        arr = np.asarray(img, dtype=np.int16)
        bits = "".join("1" if v else "0" for v in (arr[:, 1:] > arr[:, :-1]).flatten())
        return f"{int(bits, 2):0{hash_size * hash_size // 4}x}"

```

```

except Exception:
    return f"bad_{Path(path).name.lower()}"

def clean_duplicates_and_conflicts(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy().reset_index(drop=True)
    print("Computing image fingerprints...")
    df["fingerprint"] = [image_fingerprint(p, cfg.fingerprint_size)
                          for p in tqdm(df["path"].tolist())]
    df["group_key"] = df["fingerprint"]
    conflicts = set(df.groupby("fingerprint")["label"].nunique().pipe(
        lambda s: s[s > 1].index))
    if conflicts:
        print(f"Removing {len(conflicts)} conflicting fingerprint groups.")
        df = df[~df["fingerprint"].isin(conflicts)].copy()
    if cfg.drop_exact_duplicate_fingerprints:
        before = len(df)
        df["_has_split"] = df["external_split"].notna().astype(int)
        df = (df.sort_values(["fingerprint", "_has_split"], ascending=[True, False])
              .drop_duplicates(subset=["fingerprint", "label"], keep="first")
              .drop(columns=["_has_split"], errors="ignore")
              .reset_index(drop=True))
        print(f"Dropped {before - len(df)} duplicate fingerprints.")
    print("Cleaned size:", len(df))
    display(pd.crosstab(df["source"], df["label_name"], margins=True))
    return df

def make_group_stratified_split(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy().reset_index(drop=True)
    y, groups = df["label"].astype(int).values, df["group_key"].astype(str).values
    trainval_idx, test_idx = next(
        StratifiedGroupKFold(5, shuffle=True, random_state=cfg.seed).split(df, y, groups))
    trainval = df.iloc[trainval_idx].reset_index(drop=True)
    test = df.iloc[test_idx].reset_index(drop=True)
    train_idx, val_idx = next(
        StratifiedGroupKFold(5, shuffle=True, random_state=cfg.seed + 1).split(
            trainval, trainval["label"].astype(int).values,
            trainval["group_key"].astype(str).values))
    train, val = trainval.iloc[train_idx].copy(), trainval.iloc[val_idx].copy()
    train["split"], val["split"], test["split"] = "train", "val", "test"
    return pd.concat([train, val, test], ignore_index=True)

def prepare_final_dataframe(combined_df: pd.DataFrame) -> pd.DataFrame:
    clean_df = clean_duplicates_and_conflicts(combined_df)
    out = make_group_stratified_split(clean_df).reset_index(drop=True)
    display(pd.crosstab(out["split"], out["label_name"], margins=True))
    return out

```

— Load datasets (assumed local) —

```

fracatlas_dir = cfg.data_dir / "fracatlas"
multiregion_dir = cfg.data_dir / "fracture_multi_region"

frac_df = scan_dataset_generic(fracatlas_dir, "fracatlas")
multi_df = scan_dataset_generic(multiregion_dir, "multi_region")
combined_df = (pd.concat([frac_df, multi_df], ignore_index=True)
               .drop_duplicates(subset=["path"]).reset_index(drop=True))
if combined_df.empty:
    raise RuntimeError("No images found. Check dataset paths.")

df = prepare_final_dataframe(combined_df)
df.to_csv(cfg.results_dir / "metadata.csv", index=False)

counts = pd.crosstab(df["split"], df["label_name"])
ax = counts.plot(kind="bar", figsize=(8, 4))
ax.set_title("Class balance by split"); ax.set_xlabel("split"); ax.set_ylabel("images")
plt.xticks(rotation=0); plt.tight_layout(); plt.show()

# ——— Augmentation / transforms —————

IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]

class CLAHETransform:
    def __init__(self, enabled: bool = True, clip_limit: float = 2.0,
                 tile_grid_size: Tuple[int, int] = (8, 8)):
        self.enabled, self.clip_limit, self.tile_grid_size = (
            enabled, clip_limit, tile_grid_size)

    def __call__(self, img: Image.Image) -> Image.Image:
        if not self.enabled:
            return img.convert("RGB")
        arr = np.array(img.convert("RGB"))
        lab = cv2.cvtColor(arr, cv2.COLOR_RGB2LAB)
        l, a, b = cv2.split(lab)
        l2 = cv2.createCLAHE(clipLimit=self.clip_limit,
                             tileGridSize=self.tile_grid_size).apply(l)
        return Image.fromarray(cv2.cvtColor(cv2.merge([l2, a, b]), cv2.COLOR_LAB2RGB))

train_tfms = transforms.Compose([
    CLAHETransform(enabled=cfg.use_clahe),
    transforms.RandomResizedCrop(cfg.image_size, scale=(0.92, 1.0), ratio=(0.95, 1.05)),
    transforms.RandomRotation(degrees=5),
    transforms.RandomHorizontalFlip(p=0.25),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
])

valid_tfms = transforms.Compose([
    CLAHETransform(enabled=cfg.use_clahe),

```

```

        transforms.Resize((cfg.image_size, cfg.image_size)),
        transforms.ToTensor(),
        transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD),
    ])

# — Dataset & DataLoader —————

class SafeXRayDataset(Dataset):
    def __init__(self, dataframe: pd.DataFrame, transform=None):
        self.df, self.transform = dataframe.reset_index(drop=True).copy(), transform

    def __len__(self): return len(self.df)

    def __getitem__(self, idx: int):
        row = self.df.iloc[idx]
        try:
            img = Image.open(row["path"]).convert("RGB")
        except Exception:
            img = Image.fromarray(
                np.zeros((cfg.image_size, cfg.image_size, 3), dtype=np.uint8))
        if self.transform: img = self.transform(img)
        return img, torch.tensor(int(row["label"]), dtype=torch.long)

def make_weighted_sampler(train_df: pd.DataFrame) -> WeightedRandomSampler:
    labels = train_df["label"].astype(int).values
    counts = np.maximum(np.bincount(labels, minlength=2), 1)
    return WeightedRandomSampler(
        torch.DoubleTensor((1.0 / counts)[labels]), len(labels), replacement=True)

def make_loaders(df: pd.DataFrame) -> Dict[str, DataLoader]:
    splits = {s: df[df["split"] == s].reset_index(drop=True) for s in ["train", "val", "test"]}
    sampler = make_weighted_sampler(splits["train"]) if cfg.use_weighted_sampler else None
    common = dict(batch_size=cfg.batch_size, num_workers=cfg.num_workers,
                  pin_memory=True, persistent_workers=cfg.num_workers > 0)
    return {
        "train": DataLoader(SafeXRayDataset(splits["train"], train_tfms),
                           sampler=sampler, shuffle=(sampler is None), **common),
        "val":      DataLoader(SafeXRayDataset(splits["val"],      valid_tfms),  shuffle=False,
                              **common),
        "test":     DataLoader(SafeXRayDataset(splits["test"],     valid_tfms),  shuffle=False,
                              **common),
    }

loaders = make_loaders(df)
print("Loader sizes:", {k: len(v.dataset) for k, v in loaders.items()})

# — Model definitions —————

```

```

def build_resnet18(num_classes: int = 2, pretrained: bool = True) -> nn.Module:
    m = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1 if pretrained else None)
    m.fc = nn.Linear(m.fc.in_features, num_classes)
    return m

def build_resnet50(num_classes: int = 2, pretrained: bool = True) -> nn.Module:
    m = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2 if pretrained else None)
    m.fc = nn.Linear(m.fc.in_features, num_classes)
    return m

def build_convnextv2(num_classes: int = 2, pretrained: bool = True) -> nn.Module:
    name = cfg.convnextv2_model_name
    available = timm.list_models("*convnextv2*")
    if name not in available:
        name = available[0] if available else None
        if name is None:
            raise RuntimeError("No ConvNeXtV2 models in timm.")
        print("Using fallback:", name)
    return timm.create_model(name, pretrained=pretrained, num_classes=num_classes)

def create_model(model_key: str) -> nn.Module:
    builders = {"resnet18": build_resnet18, "resnet50": build_resnet50,
                "convnextv2": build_convnextv2}
    if model_key not in builders:
        raise ValueError(f"Unknown model_key: {model_key}")
    model = builders[model_key]()
    total = sum(p.numel() for p in model.parameters())
    print(f"{model_key}: params={total:,}")
    return model

# — Training utilities —————

def compute_binary_metrics(y_true, y_pred, y_prob) -> Dict[str, float]:
    m = {
        "accuracy": accuracy_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred, zero_division=0),
        "recall": recall_score(y_true, y_pred, zero_division=0),
        "f1": f1_score(y_true, y_pred, zero_division=0),
    }
    try: m["auc"] = roc_auc_score(y_true, y_prob)
    except Exception: m["auc"] = float("nan")
    return m

def safe_torch_load(path, map_location=DEVICE):
    try: return torch.load(path, map_location=map_location, weights_only=True)
    except TypeError: return torch.load(path, map_location=map_location)

def make_criterion(train_df: pd.DataFrame) -> nn.Module:
    if not cfg.use_weighted_loss:
        return nn.CrossEntropyLoss()
    counts = np.maximum(np.bincount(train_df["label"].astype(int).values, minlength=2), 1)
    weights = torch.tensor(counts.sum() / (2.0 * counts), dtype=torch.float32).to(DEVICE)

```

```

return nn.CrossEntropyLoss(weight=weights)

@torch.no_grad()
def collect_predictions(model, loader, criterion=None) -> Dict[str, Any]:
    model.eval()
    losses, y_true, y_pred, y_prob = [], [], [], []
    for images, labels in tqdm(loader, leave=False):
        images, labels = images.to(DEVICE, non_blocking=True), labels.to(DEVICE,
non_blocking=True)
        logits = model(images)
        if criterion is not None:
            losses.append(criterion(logits, labels).item() * images.size(0))
        y_true.extend(labels.cpu().numpy().tolist())
        y_pred.extend(torch.argmax(logits, dim=1).cpu().numpy().tolist())
        y_prob.extend(torch.softmax(logits, dim=1)[: , 1].cpu().numpy().tolist())
    y_true, y_pred, y_prob = map(np.asarray, [y_true, y_pred, y_prob])
    avg_loss = np.sum(losses) / max(1, len(loader.dataset)) if losses else float("nan")
    return {"loss": float(avg_loss),
            "metrics": compute_binary_metrics(y_true, y_pred, y_prob),
            "y_true": y_true, "y_pred": y_pred, "y_prob": y_prob}

def train_one_epoch(model, loader, criterion, optimizer, scaler) -> Dict[str, float]:
    model.train()
    total_loss = 0.0
    y_true, y_pred, y_prob = [], [], []
    for images, labels in tqdm(loader, leave=False):
        images, labels = images.to(DEVICE, non_blocking=True), labels.to(DEVICE,
non_blocking=True)
        optimizer.zero_grad(set_to_none=True)
        with autocast(enabled=cfg.use_amp and DEVICE.type == "cuda"):
            logits = model(images)
            loss = criterion(logits, labels)
        scaler.scale(loss).backward(); scaler.step(optimizer); scaler.update()
        total_loss += loss.item() * images.size(0)
        y_true.extend(labels.cpu().numpy().tolist())
        y_pred.extend(torch.argmax(logits.detach(), dim=1).cpu().numpy().tolist())
        y_prob.extend(torch.softmax(logits.detach(), dim=1)[: , 1].cpu().numpy().tolist())
    m = compute_binary_metrics(*map(np.asarray, [y_true, y_pred, y_prob]))
    m["loss"] = total_loss / max(1, len(loader.dataset))
    return m

def train_model(model_key: str, loaders: Dict[str, DataLoader],
                df: pd.DataFrame) -> Dict[str, Any]:
    model = create_model(model_key).to(DEVICE)
    criterion = make_criterion(df[df["split"] == "train"].reset_index(drop=True))
    optimizer = optim.AdamW(model.parameters(), lr=cfg.lr, weight_decay=cfg.weight_decay)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=cfg.epochs, eta_min=1e-6)
    scaler = GradScaler(enabled=cfg.use_amp and DEVICE.type == "cuda")
    ckpt_path = cfg.checkpoints_dir / f"{model_key}_best.pth"
    history, best_val_loss, best_epoch, bad_epochs = [], float("inf"), 0, 0

```

```

print(f"\n==== Training {model_key} =====")
for epoch in range(1, cfg.epochs + 1):
    train_m = train_one_epoch(model, loaders["train"], criterion, optimizer, scaler)
    val_out = collect_predictions(model, loaders["val"], criterion)
    val_m, val_loss = val_out["metrics"], val_out["loss"]
    row = {"epoch": epoch, "lr": optimizer.param_groups[0]["lr"],
          **{f"train_{k}": v for k, v in train_m.items()}},
          **{f"val_{k}": v for k, v in {**val_m, "loss": val_loss}.items()}}
    history.append(row)
    if val_loss < best_val_loss - 1e-5:
        best_val_loss, best_epoch, bad_epochs = val_loss, epoch, 0
        torch.save({"model_key": model_key, "epoch": epoch,
                    "model_state_dict": model.state_dict(),
                    "optimizer_state_dict": optimizer.state_dict(),
                    "val_loss": best_val_loss}, ckpt_path)
    else:
        bad_epochs += 1
    print(f"[{model_key}] {epoch:02d}/{cfg.epochs} | "
          f"train loss={row['train_loss']:.4f} acc={row['train_accuracy']:.3f} | "
          f"val loss={row['val_loss']:.4f} acc={row['val_accuracy']:.3f} "
          f"auc={row['val_auc']:.3f} | bad={bad_epochs}/{cfg.patience}")
    scheduler.step()
    if bad_epochs >= cfg.patience:
        print(f"Early stopping. Best epoch={best_epoch}, val_loss={best_val_loss:.4f}")
        break

ckpt = safe_torch_load(ckpt_path)
model.load_state_dict(ckpt["model_state_dict"])
result = {"model_key": model_key, "model": model, "checkpoint": str(ckpt_path),
          "history": pd.DataFrame(history), "best_epoch": best_epoch,
          "best_val_loss": best_val_loss,
          "val_eval": collect_predictions(model, loaders["val"], criterion),
          "test_eval": collect_predictions(model, loaders["test"], criterion)}
result["history"].to_csv(cfg.results_dir / f"{model_key}_history.csv", index=False)
return result

```

— Visualisation —————

```

def plot_history(result: Dict[str, Any]) -> None:
    key, hist = result["model_key"], result["history"]
    fig, axes = plt.subplots(2, 2, figsize=(13, 9))
    fig.suptitle(f"{key} - training curves", fontsize=15, fontweight="bold")
    for ax, (tc, vc, title) in zip(axes.flatten(), [
        ("train_loss", "val_loss", "Loss"),
        ("train_accuracy", "val_accuracy", "Accuracy"),
        ("train_precision", "val_precision", "Precision"),
        ("train_recall", "val_recall", "Recall")]):
        ax.plot(hist["epoch"], hist[tc], label="train")
        ax.plot(hist["epoch"], hist[vc], label="val")
        ax.set_title(title); ax.set_xlabel("epoch")
        ax.legend(); ax.grid(True, alpha=0.3)

```

```

plt.tight_layout()
plt.savefig(cfg.results_dir / f"{key}_training_curves.png", dpi=180, bbox_inches="tight")
plt.show()

def plot_confusion_matrix(result: Dict[str, Any], split: str = "test") -> None:
    key, ev = result["model_key"], result[f"{split}_eval"]
    cm = confusion_matrix(ev["y_true"], ev["y_pred"], labels=[0, 1])
    fig, ax = plt.subplots(figsize=(5, 4))
    im = ax.imshow(cm)
    ax.set_title(f"{key} - confusion matrix ({split})")
    ax.set_xlabel("Predicted"); ax.set_ylabel("True")
    ax.set_xticks([0, 1], [cfg.negative_class_name, cfg.positive_class_name])
    ax.set_yticks([0, 1], [cfg.negative_class_name, cfg.positive_class_name])
    for i in range(2):
        for j in range(2):
            ax.text(j, i, str(cm[i, j]), ha="center", va="center")
    plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    plt.tight_layout()
    plt.savefig(cfg.results_dir / f"{key}_{split}_confusion_matrix.png", dpi=180,
bbox_inches="tight")
    plt.show()

def plot_roc_curve(result: Dict[str, Any], split: str = "test") -> None:
    key, ev = result["model_key"], result[f"{split}_eval"]
    try:
        fpr, tpr, _ = roc_curve(ev["y_true"], ev["y_prob"])
        auc = roc_auc_score(ev["y_true"], ev["y_prob"])
    except Exception as e:
        print("ROC error:", e); return
    fig, ax = plt.subplots(figsize=(6, 5))
    ax.plot(fpr, tpr, label=f"AUC={auc:.3f}")
    ax.plot([0, 1], [0, 1], linestyle="--", label="random")
    ax.set_title(f"{key} - ROC ({split})")
    ax.set_xlabel("FPR"); ax.set_ylabel("TPR / Recall")
    ax.legend(); ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(cfg.results_dir / f"{key}_{split}_roc_curve.png", dpi=180, bbox_inches="tight")
    plt.show()

def print_model_report(result: Dict[str, Any], split: str = "test") -> None:
    ev = result[f"{split}_eval"]
    print(f"\n===== {result['model_key']} - {split} =====")
    print("Loss:", ev["loss"]); print(ev["metrics"])
    print(classification_report(ev["y_true"], ev["y_pred"],
        target_names=[cfg.negative_class_name, cfg.positive_class_name], zero_division=0))

def show_full_evaluation(result: Dict[str, Any]) -> None:
    plot_history(result)
    print_model_report(result, split="test")
    plot_confusion_matrix(result, split="test")

```

```

plot_roc_curve(result, split="test")

def compare_histories(results: Dict[str, Dict[str, Any]]) -> pd.DataFrame:
    metrics = [("val_loss", "Val Loss"), ("val_accuracy", "Val Accuracy"),
               ("val_precision", "Val Precision"), ("val_recall", "Val Recall")]
    fig, axes = plt.subplots(2, 2, figsize=(14, 9))
    fig.suptitle("Model comparison - validation curves", fontsize=16, fontweight="bold")
    for ax, (col, title) in zip(axes.flatten(), metrics):
        for key, res in results.items():
            h = res["history"]
            ax.plot(h["epoch"], h[col], marker="o", linewidth=1.5, label=key)
            ax.set_title(title); ax.set_xlabel("epoch")
            ax.grid(True, alpha=0.3); ax.legend()
    plt.tight_layout()
    plt.savefig(cfg.results_dir / "all_models_validation_comparison.png", dpi=180,
                bbox_inches="tight")
    plt.show()

    rows = []
    for key, res in results.items():
        best = res["history"].loc[res["history"]["val_loss"].idxmin()].to_dict()
        tm = res["test_eval"]["metrics"]
        rows.append({"model": key,
                     "best_epoch": int(best["epoch"]),
                     "best_val_loss": float(best["val_loss"]),
                     "best_val_auc": float(best["val_auc"]),
                     "test_accuracy": float(tm["accuracy"]),
                     "test_precision": float(tm["precision"]),
                     "test_recall": float(tm["recall"]),
                     "test_f1": float(tm["f1"]),
                     "test_auc": float(tm["auc"])})
    summary = pd.DataFrame(rows).sort_values("test_auc", ascending=False).reset_index(drop=True)
    display(summary)
    summary.to_csv(cfg.results_dir / "all_models_summary.csv", index=False)
    return summary

def compare_roc_curves(results: Dict[str, Dict[str, Any]], split: str = "test") -> None:
    fig, ax = plt.subplots(figsize=(7, 6))
    for key, res in results.items():
        ev = res[f"{split}_eval"]
        try:
            fpr, tpr, _ = roc_curve(ev["y_true"], ev["y_prob"])
            auc = roc_auc_score(ev["y_true"], ev["y_prob"])
            ax.plot(fpr, tpr, linewidth=2, label=f"{key} AUC={auc:.3f}")
        except Exception:
            continue
    ax.plot([0, 1], [0, 1], linestyle="--", label="random")
    ax.set_title(f"ROC comparison ({split})")
    ax.set_xlabel("FPR"); ax.set_ylabel("TPR"); ax.grid(True, alpha=0.3); ax.legend()
    plt.tight_layout()

```

```

plt.savefig(cfg.results_dir / f'all_models_{split}_roc_comparison.png',
            bbox_inches="tight",
            dpi=180,
            plt.show()

# ——— Run —————

ALL_RESULTS: Dict[str, Dict[str, Any]] = {}

for key in ["resnet18", "resnet50", "convnextv2"]:
    result = train_model(key, loaders, df)
    ALL_RESULTS[key] = result
    show_full_evaluation(result)

summary_df = compare_histories(ALL_RESULTS)
compare_roc_curves(ALL_RESULTS, split="test")

# segmentation_models.py
import os
import gc
import cv2
import json
import math
import time
import random
import warnings
from pathlib import Path
from typing import Dict, List, Tuple, Optional, Any

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm.auto import tqdm

from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

import albumentations as A
from albumentations.pytorch import ToTensorV2

try:
    import segmentation_models_pytorch as smp
except ImportError as exc:
    raise ImportError("Please install segmentation-models-pytorch first.") from exc

warnings.filterwarnings("ignore")

```

```

# — Paths & config —————

WORK_DIR      = Path(".")
DATA_ROOT     = WORK_DIR / "data"
WORKING_ROOT  = WORK_DIR / "fracture_working"
RUNS_ROOT     = WORK_DIR / "fracture_runs"

for d in [DATA_ROOT, WORKING_ROOT, RUNS_ROOT]:
    d.mkdir(parents=True, exist_ok=True)

ORIGINAL_DOWNLOAD_DIR = DATA_ROOT / "fracatlas"
FRACSEG_DOWNLOAD_DIR  = DATA_ROOT / "fracatlas_seg"
CLASSIFIER_CHECKPOINT = WORK_DIR / "best_model_classifier_91_acc.pth"

SEED              = 42
IMG_SIZE          = 224
BATCH_SIZE        = 8
NUM_WORKERS       = 2
ENCODER_NAME      = "resnet50"
ENCODER_WEIGHTS   = "imagenet"
DEVICE            = torch.device("cuda" if torch.cuda.is_available() else "cpu")
USE_AMP           = DEVICE.type == "cuda"
THRESHOLD         = 0.5
GRAD_CLIP         = 1.0
WEIGHT_DECAY      = 2e-4
EARLY_STOP_PATIENCE = 12
CLAHE_CLIP_LIMIT  = 2.0
CLAHE_TILE_GRID_SIZE = (8, 8)
LAYER_CAM_INPUT_SIZE = 224
CAM_PERCENTILE    = 70
CAM_MARGIN        = 28
MIN_CROP_AREA_RATIO = 0.03
MAX_CROP_AREA_RATIO = 0.75
PAPER_CROP_CONTEXT_FACTOR = 5.0

def seed_everything(seed: int = SEED) -> None:
    random.seed(seed); np.random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    torch.manual_seed(seed); torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.benchmark = True

seed_everything(SEED)
print(f"Device: {DEVICE} | AMP: {USE_AMP} | IMG_SIZE: {IMG_SIZE} | BATCH: {BATCH_SIZE}")

# — Image / mask utilities —————

IMAGE_EXTS = {".jpg", ".jpeg", ".png", ".bmp", ".tif", ".tiff", ".webp"}

def read_image_rgb(path: Path) -> np.ndarray:
    img = cv2.imread(str(path), cv2.IMREAD_COLOR)

```

```

    if img is None: raise FileNotFoundError(f"Could not read: {path}")
    return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

def read_mask(path: Path) -> np.ndarray:
    m = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)
    if m is None: raise FileNotFoundError(f"Could not read mask: {path}")
    return (m > 0).astype(np.uint8)

def apply_clahe_grayscale_rgb(img_rgb: np.ndarray) -> np.ndarray:
    gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY) if img_rgb.ndim == 3 else
img_rgb.astype(np.uint8)
    enhanced = cv2.createCLAHE(clipLimit=CLAHE_CLIP_LIMIT,
tileGridSize=CLAHE_TILE_GRID_SIZE).apply(gray)
    return cv2.cvtColor(enhanced, cv2.COLOR_GRAY2RGB)

def write_mask(path: Path, mask: np.ndarray) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    cv2.imwrite(str(path), (mask > 0).astype(np.uint8) * 255)

def list_images(root: Path) -> List[Path]:
    return sorted([p for p in Path(root).rglob("*")
if p.suffix.lower() in IMAGE_EXTS]) if Path(root).exists() else []

def normalize_id(value: Any) -> str:
    return Path(str(value).replace("\\", "/").split("/")[-1]).stem.lower().strip()

def build_file_index(paths: List[Path]) -> Dict[str, Path]:
    return {normalize_id(p.name): p for p in paths}

# — Split helpers —————

def find_split_csvs(root: Path) -> Dict[str, Path]:
    out = {}
    for p in (list(root.rglob("*.csv")) if root.exists() else []):
        name = p.name.lower()
        if name in {"train.csv", "training.csv"}: out["train"] = p
        elif name in {"valid.csv", "val.csv", "validation.csv"}: out["val"] = p
        elif name in {"test.csv", "testing.csv"}: out["test"] = p
    return out

def collect_ids_from_csv(csv_path: Path) -> set:
    df = pd.read_csv(csv_path)
    return {normalize_id(v) for col in df.columns for v in df[col].dropna().astype(str) if
v.strip()}

def assign_splits_from_csvs(records: pd.DataFrame, split_csvs: Dict[str, Path]) -> pd.DataFrame:
    records = records.copy(); records["split"] = None
    split_to_ids = {sp: collect_ids_from_csv(p) for sp, p in split_csvs.items()}
    for sp, ids in split_to_ids.items(): print(f"{sp}: {len(ids)} ids")
    for idx, row in records.iterrows():
        for sp, ids in split_to_ids.items():
            if normalize_id(row["image_id"]) in ids or normalize_id(row["image_path"]) in ids:

```

```

        records.at[idx, "split"] = sp; break
missing = records["split"].isna().sum()
if missing:
    print(f"{missing} records not assigned — using fallback random split.")
    records = assign_random_splits(records, keep_existing=True)
return records

def assign_random_splits(records: pd.DataFrame, seed: int = SEED, keep_existing: bool = False) ->
pd.DataFrame:
    records = records.copy()
    rng = np.random.default_rng(seed)
    idx = records.index[records["split"].isna()].tolist() if (keep_existing and "split" in
records.columns) else records.index.tolist()
    if not keep_existing: records["split"] = None
    arr = np.array(idx); rng.shuffle(arr)
    n, n_train, n_val = len(arr), int(0.80 * len(arr)), int(0.10 * len(arr))
    records.loc[arr[:n_train], "split"] = "train"
    records.loc[arr[n_train:n_train + n_val], "split"] = "val"
    records.loc[arr[n_train + n_val:], "split"] = "test"
    return records

# — Bounding box helpers —————

def bbox_from_mask(mask: np.ndarray, margin: int = 0) -> Optional[Tuple[int, int, int, int]]:
    ys, xs = np.where(mask > 0)
    if not len(xs): return None
    h, w = mask.shape[:2]
    x1, y1 = max(0, int(xs.min()) - margin), max(0, int(ys.min()) - margin)
    x2, y2 = min(w, int(xs.max()) + 1 + margin), min(h, int(ys.max()) + 1 + margin)
    return (x1, y1, x2, y2) if x2 > x1 and y2 > y1 else None

def expand_bbox_context(bbox: Tuple, img_w: int, img_h: int,
                        factor: float = PAPER_CROP_CONTEXT_FACTOR) -> Tuple[int, int, int, int]:
    x1, y1, x2, y2 = bbox
    cx, cy = (x1 + x2) / 2.0, (y1 + y2) / 2.0
    bw, bh = max(1, x2 - x1), max(1, y2 - y1)
    nx1, ny1 = int(round(cx - bw * factor / 2)), int(round(cy - bh * factor / 2))
    nx2, ny2 = int(round(cx + bw * factor / 2)), int(round(cy + bh * factor / 2))
    if nx1 < 0: nx2 -= nx1; nx1 = 0
    if ny1 < 0: ny2 -= ny1; ny1 = 0
    if nx2 > img_w: nx1 = max(0, nx1 - (nx2 - img_w)); nx2 = img_w
    if ny2 > img_h: ny1 = max(0, ny1 - (ny2 - img_h)); ny2 = img_h
    return int(nx1), int(ny1), int(nx2), int(ny2)

def parse_bbox(value: Any) -> Optional[Tuple[int, int, int, int]]:
    if value is None or (isinstance(value, float) and np.isnan(value)): return None
    if isinstance(value, (list, tuple)) and len(value) == 4: return tuple(map(int, value))
    try:
        parsed = json.loads(value)
        if isinstance(parsed, list) and len(parsed) == 4: return tuple(map(int, parsed))
    except Exception: pass
    return None

```

```

def crop_with_bbox(img: np.ndarray, bbox: Tuple) -> np.ndarray:
    x1, y1, x2, y2 = bbox; return img[y1:y2, x1:x2].copy()

def project_crop_mask_to_full(pred_crop: np.ndarray, bbox: Tuple, full_h: int, full_w: int) ->
np.ndarray:
    x1, y1, x2, y2 = bbox
    resized = cv2.resize(pred_crop.astype(np.float32), (max(1, x2 - x1), max(1, y2 - y1)),
interpolation=cv2.INTER_LINEAR)
    full = np.zeros((full_h, full_w), dtype=np.float32)
    full[y1:y2, x1:x2] = resized[:y2 - y1, :x2 - x1]
    return full

# — COCO mask generation —————

def polygon_or_rle_to_mask(segmentation: Any, height: int, width: int) -> np.ndarray:
    mask = np.zeros((height, width), dtype=np.uint8)
    if isinstance(segmentation, list):
        for poly in segmentation:
            arr = np.array(poly, dtype=np.float32).reshape(-1, 2)
            if len(arr) >= 3:
                cv2.fillPoly(mask, [np.round(arr).astype(np.int32)], 1)
        return mask
    if isinstance(segmentation, dict):
        try:
            from pycocotools import mask as mask_utils
            decoded = mask_utils.decode(segmentation)
            return (decoded.max(axis=2) if decoded.ndim == 3 else decoded > 0).astype(np.uint8)
        except Exception as e: print(f"RLE decode failed: {e}")
    return mask

# — Dataset building —————

def build_records_from_existing_image_mask_dirs(images_dir: Path, masks_dir: Path,
    split_csvs=None, dataset_type: str = "full_original") -> pd.DataFrame:
    mask_index = build_file_index(list_images(masks_dir))
    rows = []
    for img_path in list_images(images_dir):
        image_id = normalize_id(img_path.name)
        mask_path = mask_index.get(image_id)
        if mask_path is None: continue
        try: mask = read_mask(mask_path)
        except Exception: continue
        if mask.sum() == 0: continue
        h, w = mask.shape[:2]
        rows.append({"image_id": image_id, "image_path": str(img_path), "mask_path":
str(mask_path),
                    "orig_image_path": str(img_path), "orig_mask_path": str(mask_path),
                    "crop_bbox": json.dumps([0, 0, w, h]), "orig_w": w, "orig_h": h,
                    "bbox_source": "full_image", "dataset_type": dataset_type})
    df = pd.DataFrame(rows)

```

```

return assign_splits_from_csvs(df, split_csvs) if split_csvs else assign_random_splits(df)

def build_records_from_coco(fracatlas_root: Path) -> pd.DataFrame:
    coco_json = next(fracatlas_root.rglob("COCO_fracture_masks.json"), None)
    images_dir = next((p for p in fracatlas_root.rglob("*")
                        if p.is_dir() and "fractured" in str(p).lower() and list_images(p)), None)
    if images_dir is None or coco_json is None:
        raise FileNotFoundError("Could not locate FracAtlas images or COCO JSON.")
    with open(coco_json) as f: coco = json.load(f)
    image_index = build_file_index(list_images(images_dir))
    image_meta = {img["id"]: img for img in coco.get("images", [])}
    anns_by_image = {}
    for ann in coco.get("annotations", []): anns_by_image.setdefault(ann["image_id"],
[]).append(ann)
    out_mask_dir = WORKING_ROOT / "generated_original_masks"
    out_mask_dir.mkdir(parents=True, exist_ok=True)
    rows = []
    for image_id_num, meta in tqdm(image_meta.items(), desc="Generating masks from COCO"):
        fname = meta.get("file_name") or str(image_id_num)
        img_path = image_index.get(normalize_id(fname))
        if img_path is None: continue
        img = read_image_rgb(img_path); h, w = img.shape[:2]
        height, width = int(meta.get("height", h)), int(meta.get("width", w))
        mask = np.zeros((height, width), dtype=np.uint8)
        for ann in anns_by_image.get(image_id_num, []):
            mask |= polygon_or_rle_to_mask(ann.get("segmentation", []), height, width)
        if mask.sum() == 0: continue
        if mask.shape[:2] != (h, w): mask = cv2.resize(mask, (w, h),
interpolation=cv2.INTER_NEAREST)
        out_mask = out_mask_dir / f"{normalize_id(fname)}.png"
        write_mask(out_mask, mask)
        rows.append({"image_id": normalize_id(fname), "image_path": str(img_path),
                    "mask_path": str(out_mask), "orig_image_path": str(img_path),
                    "orig_mask_path": str(out_mask), "crop_bbox": json.dumps([0, 0, w, h]),
                    "orig_w": w, "orig_h": h, "bbox_source": "full_image", "dataset_type":
"full_original"})
    df = pd.DataFrame(rows)
    split_csvs = find_split_csvs(FRACSEG_DOWNLOAD_DIR)
    return assign_splits_from_csvs(df, split_csvs) if split_csvs else assign_random_splits(df)

def build_full_original_records() -> pd.DataFrame:
    # Try FracAtlas-SEG full images/masks first; fall back to COCO mask generation
    full_img_dir = next((p for p in [FRACSEG_DOWNLOAD_DIR / "data" / "images" / "Fractured" /
"Images",
                                FRACSEG_DOWNLOAD_DIR / "images" / "Fractured" / "Images"]
                        if p.exists()), None)
    full_mask_dir = next((p for p in [FRACSEG_DOWNLOAD_DIR / "data" / "images" / "Fractured" /
"Masks",
                                FRACSEG_DOWNLOAD_DIR / "images" / "Fractured" / "Masks"] if
p.exists()), None)
    split_csvs = find_split_csvs(FRACSEG_DOWNLOAD_DIR)
    if full_img_dir and full_mask_dir:
        print(f"FracAtlas-SEG: images={full_img_dir} masks={full_mask_dir}")

```

```

        df = build_records_from_existing_image_mask_dirs(full_img_dir, full_mask_dir, split_csvs)
    else:
        print("Full dirs not found - falling back to COCO mask generation.")
        fracatlas_root = next((p for p in [ORIGINAL_DOWNLOAD_DIR / "FracAtlas",
ORIGINAL_DOWNLOAD_DIR]
                                if (p / "dataset.csv").exists() or (p / "Annotations").exists()),
None)
        if fracatlas_root is None: raise FileNotFoundError("Could not locate FracAtlas root.")
        df = build_records_from_coco(fracatlas_root)
    out_csv = WORKING_ROOT / "records_full_original.csv"
    df.to_csv(out_csv, index=False)
    print(f"Full original: {len(df)} records | saved: {out_csv}")
    from IPython.display import display
    display(df["split"].value_counts().rename_axis("split").to_frame("count"))
    return df

```

```
full_original_records = build_full_original_records()
```

```
# ——— LayerCAM crop generation —————
```

```

try:
    from efficientnet_pytorch import EfficientNet
except ImportError:
    EfficientNet = None

```

```

class EfficientNetB4BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        if EfficientNet is None:
            raise ImportError("efficientnet-pytorch is required for LayerCAM.")
        self.backbone = EfficientNet.from_pretrained("efficientnet-b4")
        self.backbone._fc = nn.Linear(self.backbone._fc.in_features, 1)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.backbone(x).view(-1)

```

```

class ManualLayerCAM:
    def __init__(self, model: nn.Module, target_layer: nn.Module):
        self.model = model.eval(); self.activations = self.gradients = None
        self.fwd_handle = target_layer.register_forward_hook(
            lambda m, i, o: setattr(self, "activations", o.detach()))
        self.bwd_handle = target_layer.register_full_backward_hook(
            lambda m, gi, go: setattr(self, "gradients", go[0].detach()))

    def close(self): self.fwd_handle.remove(); self.bwd_handle.remove()

    def __call__(self, x: torch.Tensor) -> np.ndarray:
        self.model.zero_grad(set_to_none=True)
        self.model(x).reshape(-1)[0].backward()

```

```

        if self.activations is None or self.gradients is None:
            raise RuntimeError("LayerCAM hooks did not capture activations/gradients.")
        cam = F.relu((F.relu(self.gradients[0]) * self.activations[0]).sum(dim=0))
        cam_np = cv2.resize(cam.detach().cpu().numpy(), (x.shape[-1], x.shape[-2]),
interpolation=cv2.INTER_LINEAR)
        cam_np -= cam_np.min(); cam_np /= (cam_np.max() + 1e-8)
        return cam_np.astype(np.float32)

def load_classifier_for_layercam(checkpoint_path: Path) -> Optional[nn.Module]:
    if checkpoint_path is None or not checkpoint_path.exists():
        print(f"Classifier checkpoint not found: {checkpoint_path}"); return None
    model = EfficientNetB4BinaryClassifier().to(DEVICE)
    ckpt = torch.load(checkpoint_path, map_location=DEVICE)
    state = ckpt.get("model_state_dict") or ckpt.get("state_dict") or ckpt
    cleaned = {k.replace("module.", "").lstrip("model."): v for k, v in state.items()}
    model.load_state_dict(cleaned, strict=False)
    return model.eval()

layercam_preprocess = A.Compose([
    A.Resize(LAYER_CAM_INPUT_SIZE, LAYER_CAM_INPUT_SIZE),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2(),
])

def cam_to_bbox(cam_small: np.ndarray, orig_w: int, orig_h: int,
                margin: int = CAM_MARGIN) -> Optional[Tuple]:
    cam = cv2.GaussianBlur(cv2.resize(cam_small, (orig_w, orig_h),
interpolation=cv2.INTER_LINEAR), (15, 15), 0)
    binary = cv2.dilate(cv2.morphologyEx((cam >= np.percentile(cam,
CAM_PERCENTILE)).astype(np.uint8),
                                cv2.MORPH_CLOSE, np.ones((7, 7), np.uint8)),
                                np.ones((7, 7), np.uint8))
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if not contours: return None
    x, y, w, h = cv2.boundingRect(max(contours, key=cv2.contourArea))
    x1, y1 = max(0, x - margin), max(0, y - margin)
    x2, y2 = min(orig_w, x + w + margin), min(orig_h, y + h + margin)
    ratio = ((x2 - x1) * (y2 - y1)) / float(orig_w * orig_h + 1e-8)
    return (int(x1), int(y1), int(x2), int(y2)) if MIN_CROP_AREA_RATIO <= ratio <=
MAX_CROP_AREA_RATIO else None

def rescue_bbox_from_gt(mask: np.ndarray) -> Optional[Tuple]:
    gt_bbox = bbox_from_mask(mask, margin=CAM_MARGIN)
    if gt_bbox is None: return None
    h, w = mask.shape[:2]
    return expand_bbox_context(gt_bbox, img_w=w, img_h=h, factor=2.5)

def build_layercam_crops(records: pd.DataFrame,
```

```

        checkpoint_path: Path = CLASSIFIER_CHECKPOINT) -> pd.DataFrame:
    out_img_dir = WORKING_ROOT / "layercam_crops" / "images"
    out_mask_dir = WORKING_ROOT / "layercam_crops" / "masks"
    out_img_dir.mkdir(parents=True, exist_ok=True)
    out_mask_dir.mkdir(parents=True, exist_ok=True)

    classifier = load_classifier_for_layercam(checkpoint_path)
    cam_extractor = ManualLayerCAM(classifier, classifier.backbone.blocks[-1]) if classifier
else None

    rows = []
    for _, row in tqdm(records.iterrows(), total=len(records), desc="LayerCAM crops"):
        image_id = row["image_id"]
        img, mask = read_image_rgb(Path(row["image_path"])), read_mask(Path(row["mask_path"]))
        orig_h, orig_w = img.shape[:2]
        bbox, bbox_source = None, "layercam"
        if cam_extractor:
            try:
                tensor = layercam_preprocess(image=apply_clahe_grayscale_rgb(img))["image"].unsqueeze(0).to(DEVICE)
                bbox = cam_to_bbox(cam_extractor(tensor), orig_w, orig_h)
            except Exception as e: print(f"LayerCAM failed {image_id}: {e}")
            if bbox is None:
                bbox, bbox_source = rescue_bbox_from_gt(mask), "gt_rescue"
            if bbox is None: continue
            crop_img, crop_mask = crop_with_bbox(img, bbox), crop_with_bbox(mask, bbox)
            if crop_mask.sum() == 0:
                bbox = rescue_bbox_from_gt(mask)
                if bbox is None: continue
                bbox_source = "gt_rescue_empty_mask"
            crop_img, crop_mask = crop_with_bbox(img, bbox), crop_with_bbox(mask, bbox)
            cv2.imwrite(str(out_img_dir / f"{image_id}.png"), cv2.cvtColor(crop_img,
cv2.COLOR_RGB2BGR))
            write_mask(out_mask_dir / f"{image_id}.png", crop_mask)
            rows.append({"image_id": image_id,
                        "image_path": str(out_img_dir / f"{image_id}.png"),
                        "mask_path": str(out_mask_dir / f"{image_id}.png"),
                        "orig_image_path": row["orig_image_path"], "orig_mask_path":
row["orig_mask_path"],
                        "crop_bbox": json.dumps(list(map(int, bbox))),
                        "orig_w": orig_w, "orig_h": orig_h, "bbox_source": bbox_source,
                        "dataset_type": "layercam_crops", "split": row["split"]})
        if cam_extractor: cam_extractor.close()
    df = pd.DataFrame(rows)
    out_csv = WORKING_ROOT / "layercam_crops" / "records_layercam_crops.csv"
    df.to_csv(out_csv, index=False)
    print(f"LayerCAM crops: {len(df)} records | saved: {out_csv}")
    return df

layercam_crop_records = build_layercam_crops(full_original_records, CLASSIFIER_CHECKPOINT)

def build_paper_crop_records(full_records: pd.DataFrame) -> pd.DataFrame:

```

```

        crop_img_dir = next((p for p in [DATA_ROOT /
"fracatlas_seg/images/images/croppedImages/Images"] if p.exists()), None)
        crop_mask_dir = next((p for p in [DATA_ROOT /
"fracatlas_seg/images/images/croppedImages/Masks"] if p.exists()), None)
        if crop_img_dir is None or crop_mask_dir is None:
            raise FileNotFoundError("Could not locate croppedImages/Images and croppedImages/Masks.")
        mask_index = build_file_index(list_images(crop_mask_dir))
        full_index = {normalize_id(row["image_id"]): row for _, row in full_records.iterrows()}
        rows = []
        for crop_img_path in list_images(crop_img_dir):
            image_id = normalize_id(crop_img_path.name)
            crop_mask_path = mask_index.get(image_id)
            if crop_mask_path is None or image_id not in full_index: continue
            full_row = full_index[image_id]
            orig_img = read_image_rgb(Path(full_row["orig_image_path"]))
            orig_mask = read_mask(Path(full_row["orig_mask_path"]))
            orig_h, orig_w = orig_img.shape[:2]
            gt_bbox = bbox_from_mask(orig_mask, margin=0)
            if gt_bbox is None: continue
            bbox = expand_bbox_context(gt_bbox, orig_w, orig_h, PAPER_CROP_CONTEXT_FACTOR)
            rows.append({"image_id": image_id, "image_path": str(crop_img_path), "mask_path":
str(crop_mask_path),
                        "orig_image_path": str(full_row["orig_image_path"]), "orig_mask_path":
str(full_row["orig_mask_path"]),
                        "crop_bbox": json.dumps(list(map(int, bbox))), "orig_w": orig_w, "orig_h":
orig_h,
                        "bbox_source": "reconstructed_from_gt_bbox", "dataset_type": "paper_crops",
                        "split": full_row["split"]})
        df = pd.DataFrame(rows)
        df.to_csv(WORKING_ROOT / "records_paper_crops.csv", index=False)
        print(f"Paper crop records: {len(df)}")
        return df

```

```
paper_crop_records = build_paper_crop_records(full_original_records)
```

```
# — Augmentation & Dataset —————
```

```

train_transform = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
    A.HorizontalFlip(p=0.5), A.VerticalFlip(p=0.15), A.RandomRotate90(p=0.25),
    A.ShiftScaleRotate(shift_limit=0.04, scale_limit=0.08, rotate_limit=12,
                        border_mode=cv2.BORDER_REFLECT, p=0.45),
    A.RandomGamma(gamma_limit=(85, 115), p=0.30),
    A.GaussianBlur(blur_limit=(3, 5), p=0.12),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2(),
])

valid_transform = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),

```

```

        ToTensorV2(),
    ])

class FractureSegmentationDataset(Dataset):
    def __init__(self, df: pd.DataFrame, transform: A.Compose):
        self.df, self.transform = df.reset_index(drop=True), transform

    def __len__(self) -> int: return len(self.df)

    def __getitem__(self, idx: int) -> Dict[str, Any]:
        row = self.df.iloc[idx]
        img = apply_clahe_grayscale_rgb(read_image_rgb(Path(row["image_path"])))
        mask = read_mask(Path(row["mask_path"])).astype(np.float32)
        t = self.transform(image=img, mask=mask)
        return {"image": t["image"].float(), "mask": t["mask"].float().unsqueeze(0),
                "image_id": row["image_id"]}

def build_dataloaders(df: pd.DataFrame) -> Tuple[Dict[str, DataLoader], Dict[str, pd.DataFrame]]:
    splits = {
        "train": df[df["split"] == "train"].reset_index(drop=True),
        "val": df[df["split"].isin(["val", "valid", "validation"])].reset_index(drop=True),
        "test": df[df["split"] == "test"].reset_index(drop=True),
    }
    common = dict(batch_size=BATCH_SIZE, num_workers=NUM_WORKERS,
                  pin_memory=DEVICE.type == "cuda", drop_last=False, persistent_workers=True)
    loaders = {
        "train": DataLoader(FractureSegmentationDataset(splits["train"], train_transform),
                           shuffle=True, **common),
        "val": DataLoader(FractureSegmentationDataset(splits["val"], valid_transform),
                          shuffle=False, **common),
        "test": DataLoader(FractureSegmentationDataset(splits["test"], valid_transform),
                           shuffle=False, **common),
    }
    for k, v in splits.items(): print(f"{k}: {len(v)}")
    return loaders, splits

CURRENT_DATASET_NAME = CURRENT_RECORDS = CURRENT_LOADERS = CURRENT_SPLITS = None

def set_current_dataset(name: str, records: pd.DataFrame):
    global CURRENT_DATASET_NAME, CURRENT_RECORDS, CURRENT_LOADERS, CURRENT_SPLITS
    CURRENT_DATASET_NAME = name
    CURRENT_RECORDS = records.reset_index(drop=True)
    CURRENT_LOADERS, CURRENT_SPLITS = build_dataloaders(CURRENT_RECORDS)
    print(f"Active dataset: {CURRENT_DATASET_NAME}")
    return CURRENT_LOADERS, CURRENT_SPLITS

# — Model definitions —————

class AttentionDeepLabV3Plus(nn.Module):

```

```

def __init__(self, encoder_name=ENCODER_NAME, encoder_weights=ENCODER_WEIGHTS,
              in_channels=3, classes=1):
    super().__init__()
    self.base = smp.DeepLabV3Plus(encoder_name=encoder_name, encoder_weights=encoder_weights,
                                  in_channels=in_channels, classes=classes, activation=None)

    self.gate = nn.Sequential(nn.Conv2d(classes, classes, 7, padding=3, bias=True),
                              nn.Sigmoid())

    self.refine = nn.Sequential(nn.Conv2d(classes, 16, 3, padding=1, bias=False),
                                nn.BatchNorm2d(16), nn.ReLU(inplace=True),
                                nn.Conv2d(16, classes, 1))

def forward(self, x: torch.Tensor) -> torch.Tensor:
    logits = self.base(x)
    return logits + self.gate(logits) * self.refine(logits)

MODEL_DISPLAY_NAMES = {
    "unet": "U-Net", "unetpp": "U-Net++", "deeplabv3plus": "DeepLabV3+",
    "attention_deeplabv3plus": "Attention DeepLabV3+", "attention_unetpp": "Attention U-Net++",
}

def build_model(model_key: str) -> nn.Module:
    key = model_key.lower()
    common = dict(encoder_name=ENCODER_NAME, encoder_weights=ENCODER_WEIGHTS,
                  in_channels=3, classes=1, activation=None)
    if key == "unet":
        return smp.Unet(**common)
    if key == "unetpp":
        return smp.UnetPlusPlus(**common,
    decoder_attention_type=None)
    if key == "deeplabv3plus":
        return smp.DeepLabV3Plus(**common)
    if key == "attention_deeplabv3plus":
        return AttentionDeepLabV3Plus()
    if key == "attention_unetpp":
        return smp.UnetPlusPlus(**common,
    decoder_attention_type="scse")
    raise ValueError(f"Unknown model_key: {model_key}")

def get_encoder(model: nn.Module) -> nn.Module:
    core = model.base if hasattr(model, "base") else model
    return core.encoder

# — Loss & metrics —————

class DiceBCEtverskyLoss(nn.Module):
    def __init__(self, bce_weight=0.35, dice_weight=0.35, tversky_weight=0.30,
                  alpha=0.70, beta=0.30, smooth=1e-6):
        super().__init__()
        self.bce = nn.BCEWithLogitsLoss()
        self.bce_w, self.dice_w, self.tv_w = bce_weight, dice_weight, tversky_weight
        self.alpha, self.beta, self.smooth = alpha, beta, smooth

    def forward(self, logits: torch.Tensor, targets: torch.Tensor) -> torch.Tensor:
        bce = self.bce(logits, targets)
        probs = torch.sigmoid(logits)
        pf, tf = probs.view(probs.size(0), -1), targets.view(targets.size(0), -1)

```

```

    tp, fp, fn = (pf * tf).sum(1), (pf * (1 - tf)).sum(1), ((1 - pf) * tf).sum(1)
    dice_loss = 1 - ((2 * tp + self.smooth) / (2 * tp + fp + fn + self.smooth)).mean()
    tversky_loss = 1 - ((tp + self.smooth) / (tp + self.alpha * fn + self.beta * fp +
self.smooth)).mean()
    return self.bce_w * bce + self.dice_w * dice_loss + self.tv_w * tversky_loss

def metric_sums(logits: torch.Tensor, targets: torch.Tensor,
                threshold: float = THRESHOLD) -> Dict[str, float]:
    preds = (torch.sigmoid(logits) > threshold).float()
    t = (targets > 0.5).float()
    tp = (preds * t).sum().item(); fp = (preds * (1 - t)).sum().item()
    fn = ((1 - preds) * t).sum().item(); tn = ((1 - preds) * (1 - t)).sum().item()
    return {"tp": tp, "fp": fp, "fn": fn, "tn": tn}

def finalize_metrics(s: Dict, eps: float = 1e-7) -> Dict[str, float]:
    tp, fp, fn, tn = s["tp"], s["fp"], s["fn"], s["tn"]
    return {"dice": (2 * tp + eps) / (2 * tp + fp + fn + eps),
            "iou": (tp + eps) / (tp + fp + fn + eps),
            "precision": (tp + eps) / (tp + fp + eps),
            "recall": (tp + eps) / (tp + fn + eps),
            "accuracy": (tp + tn + eps) / (tp + tn + fp + fn + eps)}

def merge_metric_sums(a: Dict, b: Dict) -> Dict:
    return {k: a.get(k, 0.0) + float(b.get(k, 0.0)) for k in set(a) | set(b)}

# — Optimizer / scheduler —————

def set_requires_grad(module: nn.Module, value: bool) -> None:
    for p in module.parameters(): p.requires_grad = value

def set_trainable_stage(model: nn.Module, encoder_blocks: List[str]) -> None:
    encoder = get_encoder(model)
    enc_ids = {id(p) for p in encoder.parameters()}
    for p in model.parameters(): p.requires_grad = id(p) not in enc_ids
    for name, child in encoder.named_children():
        if name in encoder_blocks: set_requires_grad(child, True)

def make_optimizer(model: nn.Module, lr_encoder: float, lr_decoder: float) -> optim.Optimizer:
    encoder = get_encoder(model); enc_ids = {id(p) for p in encoder.parameters()}
    dec_params = [p for p in model.parameters() if p.requires_grad and id(p) not in enc_ids]
    enc_params = [p for p in model.parameters() if p.requires_grad and id(p) in enc_ids]
    groups = []
    if dec_params: groups.append({"params": dec_params, "lr": lr_decoder})
    if enc_params: groups.append({"params": enc_params, "lr": lr_encoder})
    return optim.AdamW(groups, weight_decay=WEIGHT_DECAY)

# — Training config (progressive unfreezing) —————

MODEL_TRAIN_CONFIGS = {
    "unet": {"stages": [

```

```

        ("head",    [],
4),
        ("layer4", ["layer4"],
4),
        ("layer3", ["layer4","layer3"],
4),
        ("layer2", ["layer4","layer3","layer2"],
4),
        ("layer1", ["layer4","layer3","layer2","layer1"],
1.5e-4),
        ("full",   ["conv1","bn1","layer1","layer2","layer3","layer4"],
4),
    ]},
    "unetpp": {"stages": [
        ("head",    [],
4),
        ("layer4", ["layer4"],
3.5e-4),
        ("layer3", ["layer4","layer3"],
2.5e-4),
        ("layer2", ["layer4","layer3","layer2"],
1.8e-4),
        ("layer1", ["layer4","layer3","layer2","layer1"],
1.2e-4),
        ("full",    ["conv1","bn1","layer1","layer2","layer3","layer4"],
8e-5),
    ]},
    "deeplabv3plus": {"stages": [
        ("head",    [],
4),
        ("layer4", ["layer4"],
4),
        ("layer3", ["layer4","layer3"],
4),
        ("layer2", ["layer4","layer3","layer2"],
4),
        ("layer1", ["layer4","layer3","layer2","layer1"],
1.5e-4),
        ("full",    ["conv1","bn1","layer1","layer2","layer3","layer4"],
4),
    ]},
    "attention_deeplabv3plus": {"stages": [
        ("head",    [],
4),
        ("layer4", ["layer4"],
3.5e-4),
        ("layer3", ["layer4","layer3"],
2.5e-4),
        ("layer2", ["layer4","layer3","layer2"],
1.8e-4),
        ("layer1", ["layer4","layer3","layer2","layer1"],
1.2e-4),

```

```

        ("full",      ["conv1", "bn1", "layer1", "layer2", "layer3", "layer4"],      26, 1.5e-5,
8e-5),
    ]},
    "attention_unetpp": {"stages": [
        ("head",      [],      14, 0.0,      4e-
4),
        ("layer4", ["layer4"],      5, 5e-5,      3e-
4),
        ("layer3", ["layer4", "layer3"],      5, 4e-5,      2e-
4),
        ("layer2", ["layer4", "layer3", "layer2"],      4, 2.5e-5,
1.5e-4),
        ("layer1", ["layer4", "layer3", "layer2", "layer1"],      4, 1.8e-5,
1e-4),
        ("full",      ["conv1", "bn1", "layer1", "layer2", "layer3", "layer4"],      28, 1.2e-5,
7e-5),
    ]},
}

```

— Training loop —————

```

def train_one_epoch(model, loader, criterion, optimizer, scaler) -> Dict[str, float]:
    model.train()
    running_loss, running_count = 0.0, 0
    sums = {"tp": 0.0, "fp": 0.0, "fn": 0.0, "tn": 0.0}
    pbar = tqdm(loader, desc="train", leave=False)
    for batch in pbar:
        x, y = batch["image"].to(DEVICE, non_blocking=True), batch["mask"].to(DEVICE,
non_blocking=True)
        optimizer.zero_grad(set_to_none=True)
        with torch.cuda.amp.autocast(enabled=USE_AMP):
            logits = model(x); loss = criterion(logits, y)
        scaler.scale(loss).backward()
        if GRAD_CLIP: scaler.unscale_(optimizer);
    torch.nn.utils.clip_grad_norm_(model.parameters(), GRAD_CLIP)
    scaler.step(optimizer); scaler.update()
    bs = x.size(0); running_loss += loss.item() * bs; running_count += bs
    sums = merge_metric_sums(sums, metric_sums(logits.detach(), y.detach()))
    m = finalize_metrics(sums)
    pbar.set_postfix(loss=running_loss / max(1, running_count), dice=m["dice"], iou=m["iou"])
    metrics = finalize_metrics(sums); metrics["loss"] = running_loss / max(1, running_count)
    return metrics

@torch.no_grad()
def evaluate_one_epoch(model, loader, criterion, desc: str = "val") -> Dict[str, float]:
    model.eval()
    running_loss, running_count = 0.0, 0
    sums = {"tp": 0.0, "fp": 0.0, "fn": 0.0, "tn": 0.0}
    for batch in tqdm(loader, desc=desc, leave=False):
        x, y = batch["image"].to(DEVICE, non_blocking=True), batch["mask"].to(DEVICE,
non_blocking=True)

```

```

        with torch.cuda.amp.autocast(enabled=USE_AMP):
            logits = model(x); loss = criterion(logits, y)
            bs = x.size(0); running_loss += loss.item() * bs; running_count += bs
            sums = merge_metric_sums(sums, metric_sums(logits, y))
        metrics = finalize_metrics(sums); metrics["loss"] = running_loss / max(1, running_count)
    return metrics

# — Visualisation —————

def plot_history(history: List[Dict], title: str, save_path: Optional[Path] = None) -> None:
    df = pd.DataFrame(history)
    if df.empty: return
    plt.figure(figsize=(18, 5))
    for i, (tc, vc, label) in enumerate([
        ("train_loss", "val_loss", "Loss"),
        ("train_dice", "val_dice", "Dice"),
        ("train_iou", "val_iou", "IoU")], start=1):
        plt.subplot(1, 3, i)
        plt.plot(df["epoch"], df[tc], label="Train")
        plt.plot(df["epoch"], df[vc], label="Val")
        plt.title(label); plt.xlabel("Epoch"); plt.grid(True, alpha=0.3); plt.legend()
    plt.suptitle(title); plt.tight_layout()
    if save_path:
        save_path.parent.mkdir(parents=True, exist_ok=True)
        plt.savefig(save_path, dpi=160, bbox_inches="tight")
    plt.show()

# — Experiment runner —————

def run_experiment(model_key: str) -> Dict[str, Any]:
    if CURRENT_DATASET_NAME is None:
        raise RuntimeError("Call set_current_dataset() first.")
    display_name = MODEL_DISPLAY_NAMES[model_key]
    out_dir = RUNS_ROOT / CURRENT_DATASET_NAME / model_key
    out_dir.mkdir(parents=True, exist_ok=True)
    ckpt_path = out_dir / "best_checkpoint.pth"
    model = build_model(model_key).to(DEVICE)
    criterion = DiceBCEtverskyLoss()
    scaler = torch.cuda.amp.GradScaler(enabled=USE_AMP)
    history = []
    best_val_dice, best_epoch, stale_epochs, global_epoch = -1.0, -1, 0, 0

    print("=" * 90)
    print(f"Dataset: {CURRENT_DATASET_NAME} | Model: {display_name}")
    print("=" * 90)

    for stage_name, blocks, num_epochs, lr_enc, lr_dec in
        MODEL_TRAIN_CONFIGS[model_key]["stages"]:
        set_trainable_stage(model, blocks)
        optimizer = make_optimizer(model, lr_enc, lr_dec)
        scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode="max", factor=0.5,

```

```

patience=4, min_lr=1e-7)

trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"Stage: {stage_name:>8} | epochs={num_epochs:02d} | "
      f"enc_lr={lr_enc:.2e} | dec_lr={lr_dec:.2e} | trainable={trainable:,}")

for _ in range(num_epochs):
    global_epoch += 1
    start = time.time()
    train_m = train_one_epoch(model, CURRENT_LOADERS["train"], criterion, optimizer,
scaler)

    val_m = evaluate_one_epoch(model, CURRENT_LOADERS["val"], criterion)
    scheduler.step(val_m["dice"])
    row = {"epoch": global_epoch, "stage": stage_name,
          **{f"train_{k}": v for k, v in train_m.items()},
          **{f"val_{k}": v for k, v in val_m.items()},
          "seconds": time.time() - start}
    history.append(row)
    print(f"Epoch {global_epoch:03d} [{stage_name}] "
          f"loss {row['train_loss']:.4f}/{row['val_loss']:.4f} | "
          f"dice {row['train_dice']:.4f}/{row['val_dice']:.4f} | "
          f"iou {row['train_iou']:.4f}/{row['val_iou']:.4f}")
    if val_m["dice"] > best_val_dice:
        best_val_dice, best_epoch, stale_epochs = val_m["dice"], global_epoch, 0
        torch.save({"model_key": model_key, "model_display_name": display_name,
                    "dataset_name": CURRENT_DATASET_NAME,
                    "image_size": IMG_SIZE, "batch_size": BATCH_SIZE,
                    "encoder_name": ENCODER_NAME, "threshold": THRESHOLD,
                    "model_state_dict": model.state_dict(),
                    "best_val_dice": best_val_dice, "best_epoch": best_epoch}, ckpt_path)
    else:
        stale_epochs += 1
    if stale_epochs >= EARLY_STOP_PATIENCE:
        print(f"Early stopping after {stale_epochs} stale epochs."); break
if stale_epochs >= EARLY_STOP_PATIENCE: break

pd.DataFrame(history).to_csv(out_dir / "history.csv", index=False)
model.load_state_dict(torch.load(ckpt_path, map_location=DEVICE)["model_state_dict"])
test_m = evaluate_one_epoch(model, CURRENT_LOADERS["test"], criterion, desc="test")

result = {"dataset": CURRENT_DATASET_NAME, "model_key": model_key,
          "model_display_name": display_name, "best_epoch": int(best_epoch),
          "best_val_dice": float(best_val_dice),
          "test_metrics": {k: float(v) for k, v in test_m.items()},
          "checkpoint": str(ckpt_path)}
with open(out_dir / "metrics.json", "w") as f: json.dump(result, f, indent=2)

summary_csv = RUNS_ROOT / "summary_metrics.csv"
summary_row = {"dataset": CURRENT_DATASET_NAME, "model": display_name, "model_key":
model_key,
               "best_epoch": best_epoch, "best_val_dice": best_val_dice,
               **{f"test_{k}": v for k, v in test_m.items()}, "checkpoint": str(ckpt_path)}
sdf = pd.concat([pd.read_csv(summary_csv).pipe(

```

```

        lambda d: d[~((d["dataset"] == CURRENT_DATASET_NAME) & (d["model_key"]
== model_key))])

        if summary_csv.exists() else pd.DataFrame(),
        pd.DataFrame([summary_row])), ignore_index=True)
    sdf.to_csv(summary_csv, index=False)

    plot_history(history, title=f"{display_name} - {CURRENT_DATASET_NAME}",
                  save_path=out_dir / "training_curves.png")
    from IPython.display import display as ipy_display
    ipy_display(pd.DataFrame([test_m]))
    del model; gc.collect()
    if DEVICE.type == "cuda": torch.cuda.empty_cache()
    return result

# — Run all experiments —————

CURRENT_LOADERS, CURRENT_SPLITS = set_current_dataset("full_original", full_original_records)

MODEL_KEYS = ["unet", "unetpp", "deeplabv3plus", "attention_deeplabv3plus", "attention_unetpp"]
ALL_RESULTS = {}
for key in MODEL_KEYS:
    ALL_RESULTS[key] = run_experiment(key)

summary_csv = RUNS_ROOT / "summary_metrics.csv"
if summary_csv.exists():
    from IPython.display import display as ipy_display
    ipy_display(pd.read_csv(summary_csv).sort_values(["dataset", "test_dice"], ascending=[True,
False]))

# classifier.py
import torch
import torch.nn as nn
from pathlib import Path

try:
    from efficientnet_pytorch import EfficientNet
except ImportError:
    raise ImportError("Install: pip install efficientnet-pytorch")

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class EfficientNetB4BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
        self.backbone = EfficientNet.from_pretrained("efficientnet-b4")
        self.backbone._fc = nn.Linear(self.backbone._fc.in_features, 2)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.backbone(x)

```

```

def load_classifier(checkpoint_path: str) -> nn.Module:
    path = Path(checkpoint_path)
    if not path.exists():
        raise FileNotFoundError(f"Checkpoint not found: {path}")
    model = EfficientNetB4BinaryClassifier().to(DEVICE)
    state = torch.load(path, map_location=DEVICE)
    fixed = {"backbone." + k: v for k, v in state.items()}
    missing, unexpected = model.load_state_dict(fixed, strict=False)
    print(f"Classifier loaded: {path} | missing={len(missing)} unexpected={len(unexpected)}")
    return model.eval()

@torch.no_grad()
def classify_image(model: nn.Module, tensor: torch.Tensor) -> tuple[int, float]:
    probs = torch.softmax(model(tensor), dim=1)[0]
    label = int(torch.argmax(probs).item())
    return label, probs[label].item()

# segmentation.py
import cv2
import numpy as np
import torch
import torch.nn as nn
import albumentations as A
from albumentations.pytorch import ToTensorV2
from pathlib import Path
from typing import Tuple
import segmentation_models_pytorch as smp

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
USE_AMP = DEVICE.type == "cuda"

IMG_SIZE = 224
THRESHOLD = 0.5
ENCODER_NAME = "resnet50"
ENCODER_WEIGHTS = "imagenet"
MASK_COLOR = (196, 78, 82)
MASK_ALPHA = 0.45

seg_preprocess = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2(),
])

class AttentionDeepLabV3Plus(nn.Module):
    def __init__(self, encoder_name=ENCODER_NAME, encoder_weights=ENCODER_WEIGHTS,
                  in_channels=3, classes=1):
        super().__init__()
        self.base = smp.DeepLabV3Plus(encoder_name=encoder_name, encoder_weights=encoder_weights,

```

```

        in_channels=in_channels, classes=classes, activation=None)
    self.gate      = nn.Sequential(nn.Conv2d(classes, classes, 7, padding=3, bias=True),
nn.Sigmoid())
    self.refine = nn.Sequential(nn.Conv2d(classes, 16, 3, padding=1, bias=False),
                                nn.BatchNorm2d(16), nn.ReLU(inplace=True),
                                nn.Conv2d(16, classes, 1))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        logits = self.base(x)
        return logits + self.gate(logits) * self.refine(logits)

def load_seg_model(checkpoint_path: str) -> nn.Module:
    path = Path(checkpoint_path)
    if not path.exists():
        raise FileNotFoundError(f"Checkpoint not found: {path}")
    model = AttentionDeepLabV3Plus().to(DEVICE)
    ckpt = torch.load(path, map_location=DEVICE, weights_only=False)
    missing, unexpected = model.load_state_dict(ckpt.get("model_state_dict", ckpt), strict=False)
    print(f"Seg model loaded: {path} | missing={len(missing)} unexpected={len(unexpected)}")
    return model.eval()

def apply_clahe(img_rgb: np.ndarray) -> np.ndarray:
    gray = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8)).apply(
        cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY))
    return cv2.cvtColor(gray, cv2.COLOR_GRAY2RGB)

@torch.no_grad()
def predict_mask(model: nn.Module, image_rgb: np.ndarray) -> np.ndarray:
    orig_h, orig_w = image_rgb.shape[:2]
    tensor = seg_preprocess(image=apply_clahe(image_rgb))["image"].unsqueeze(0).to(DEVICE)
    with torch.cuda.amp.autocast(enabled=USE_AMP):
        prob = torch.sigmoid(model(tensor))[0, 0].float().cpu().numpy()
    return cv2.resize(prob, (orig_w, orig_h), interpolation=cv2.INTER_LINEAR).astype(np.float32)

def overlay_mask(image_rgb: np.ndarray, prob_map: np.ndarray,
                 threshold: float = THRESHOLD,
                 color: Tuple[int, int, int] = MASK_COLOR,
                 alpha: float = MASK_ALPHA) -> np.ndarray:
    binary = (prob_map > threshold).astype(np.uint8)
    colored = np.zeros_like(image_rgb); colored[binary == 1] = color
    result = cv2.addWeighted(image_rgb, 1.0, colored, alpha, 0)
    contours, _ = cv2.findContours(binary * 255, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cv2.drawContours(result, contours, -1, color, thickness=2)
    return result

def run_segmentation(model: nn.Module, image_rgb: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    prob_map = predict_mask(model, image_rgb)
    return overlay_mask(image_rgb, prob_map), prob_map

```

```

# layercam.py
import cv2
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import albumentations as A
from albumentations.pytorch import ToTensorV2
from typing import Optional, Tuple

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

CAM_INPUT_SIZE      = 224
CAM_PERCENTILE       = 80
CAM_MARGIN           = 20
MIN_CROP_AREA_RATIO = 0.05
MAX_CROP_AREA_RATIO = 0.7

cam_preprocess = A.Compose([
    A.Resize(CAM_INPUT_SIZE, CAM_INPUT_SIZE),
    A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    ToTensorV2(),
])

def apply_clahe(img_rgb: np.ndarray) -> np.ndarray:
    gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
    gray_eq = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8)).apply(gray)
    return cv2.cvtColor(gray_eq, cv2.COLOR_GRAY2RGB)

class ManualLayerCAM:
    def __init__(self, model: nn.Module, target_layer: nn.Module):
        self.model = model.eval()
        self.activations = self.gradients = None
        self.fwd_handle = target_layer.register_forward_hook(
            lambda m, i, o: setattr(self, "activations", o.detach()))
        self.bwd_handle = target_layer.register_full_backward_hook(
            lambda m, gi, go: setattr(self, "gradients", go[0].detach()))

    def close(self):
        self.fwd_handle.remove(); self.bwd_handle.remove()

    def __call__(self, x: torch.Tensor, target_class: int = 1) -> np.ndarray:
        self.model.zero_grad(set_to_none=True)
        self.model(x)[0, target_class].backward()
        if self.activations is None or self.gradients is None:
            raise RuntimeError("LayerCAM hooks did not capture activations/gradients.")
        cam = F.relu((F.relu(self.gradients[0]) * self.activations[0]).sum(dim=0))
        cam_np = cv2.resize(cam.detach().cpu().numpy(), (x.shape[-1], x.shape[-2]),
                             interpolation=cv2.INTER_LINEAR)
        cam_np -= cam_np.min(); cam_np /= (cam_np.max() + 1e-8)
        return cam_np.astype(np.float32)

```

```

def get_target_layer(model: nn.Module) -> nn.Module:
    return model.backbone._blocks[-1]

def cam_to_bbox(cam: np.ndarray, orig_w: int, orig_h: int,
               margin: int = CAM_MARGIN) -> Optional[Tuple[int, int, int, int]]:
    cam_b = cv2.GaussianBlur(cv2.resize(cam, (orig_w, orig_h), interpolation=cv2.INTER_LINEAR),
                             (15, 15), 0)
    binary = (cam_b >= np.percentile(cam_b, CAM_PERCENTILE)).astype(np.uint8)
    kernel = np.ones((7, 7), np.uint8)
    binary = cv2.dilate(cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel), kernel)
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if not contours: return None
    x, y, w, h = cv2.boundingRect(max(contours, key=cv2.contourArea))
    if not (MIN_CROP_AREA_RATIO <= (w * h) / (orig_w * orig_h) <= MAX_CROP_AREA_RATIO): return
    None
    return max(0, x - margin), max(0, y - margin), min(orig_w, x + w + margin), min(orig_h, y + h
    + margin)

def generate_layercam(
    model: nn.Module, image_rgb: np.ndarray,
) -> Tuple[np.ndarray, np.ndarray, Optional[Tuple], Optional[np.ndarray]]:
    orig_h, orig_w = image_rgb.shape[:2]
    tensor = cam_preprocess(image=apply_clahe(image_rgb))["image"].unsqueeze(0).to(DEVICE)
    cam_engine = ManualLayerCAM(model, get_target_layer(model))
    cam_map = cam_engine(tensor, target_class=1)
    cam_engine.close()

    cam_full = cv2.resize(cam_map, (orig_w, orig_h), interpolation=cv2.INTER_LINEAR)
    heatmap_rgb = cv2.cvtColor(cv2.applyColorMap((cam_full * 255).astype(np.uint8),
                                                  cv2.COLORMAP_JET), cv2.COLOR_BGR2RGB)
    heatmap_overlay = cv2.addWeighted(image_rgb, 0.5, heatmap_rgb, 0.5, 0)

    bbox = cam_to_bbox(cam_map, orig_w, orig_h)
    crop = image_rgb[bbox[1]:bbox[3], bbox[0]:bbox[2]].copy() if bbox is not None else None
    return heatmap_overlay, cam_full, bbox, crop

# pipeline.py
import cv2
import numpy as np
import base64
from typing import Optional, Dict, Any

from classifier import classify_image, load_classifier
from layercam import generate_layercam, apply_clahe
from segmentation import run_segmentation, load_seg_model

def numpy_to_base64(image_rgb: np.ndarray) -> str:
    _, buffer = cv2.imencode(".jpg", cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR),

```

```

        [cv2.IMWRITE_JPEG_QUALITY, 92])
    return base64.b64encode(buffer).decode("utf-8")

def read_image_rgb(file_bytes: bytes) -> np.ndarray:
    img_bgr = cv2.imdecode(np.frombuffer(file_bytes, np.uint8), cv2.IMREAD_COLOR)
    if img_bgr is None:
        raise ValueError("Cannot decode image. Make sure it is a valid JPEG/PNG.")
    return cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

class ModelRegistry:
    def __init__(self, classifier_path: str, seg_full_path: str,
                  seg_layercam_path: str, seg_bbox_path: str):
        print("Loading classifier...")
        self.classifier = load_classifier(classifier_path)
        print("Loading segmentation models...")
        self.seg_full = load_seg_model(seg_full_path)
        self.seg_layercam = load_seg_model(seg_layercam_path)
        self.seg_bbox = load_seg_model(seg_bbox_path)
        print("All models loaded.")

    def get_seg_model(self, mode: str):
        models = {"full": self.seg_full, "layercam": self.seg_layercam, "bbox": self.seg_bbox}
        if mode not in models:
            raise ValueError(f"Unknown segmentation mode: {mode}")
        return models[mode]

def run_classify_pipeline(registry: ModelRegistry, file_bytes: bytes) -> Dict[str, Any]:
    import alumentations as A
    from alumentations.pytorch import ToTensorV2
    import torch

    image_rgb = read_image_rgb(file_bytes)
    orig_h, orig_w = image_rgb.shape[:2]

    cls_transform = A.Compose([
        A.Resize(224, 224),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ])

    tensor = cls_transform(image=apply_clahe(image_rgb))["image"].unsqueeze(0)
    tensor = tensor.to(next(registry.classifier.parameters()).device)
    label, confidence = classify_image(registry.classifier, tensor)

    heatmap_overlay, cam_map, bbox, crop = generate_layercam(registry.classifier, image_rgb)

    result: Dict[str, Any] = {
        "label": label,
        "confidence": round(confidence, 4),
        "original_image": numpy_to_base64(image_rgb),
        "layercam_image": numpy_to_base64(heatmap_overlay),
    }

```

```

        "image_width":    orig_w,
        "image_height":   orig_h,
        "layercam_crop":  None,
        "crop_bbox":      None,
    }
    if crop is not None and crop.size > 0:
        result["layercam_crop"] = numpy_to_base64(crop)
        result["crop_bbox"] = {"x": bbox[0], "y": bbox[1],
                               "w": bbox[2] - bbox[0], "h": bbox[3] - bbox[1]}
    return result

def run_segment_pipeline(registry: ModelRegistry, file_bytes: bytes,
                        mode: str, bbox_coords: Optional[Dict[str, int]] = None) -> Dict[str,
Any]:
    image_rgb = read_image_rgb(file_bytes)

    if mode == "full":
        input_image = image_rgb
    elif mode == "layercam":
        _, _, bbox, crop = generate_layercam(registry.classifier, image_rgb)
        input_image = crop if (crop is not None and crop.size > 0) else image_rgb
    elif mode == "bbox":
        if bbox_coords is None:
            raise ValueError("bbox_coords required for mode='bbox'")
        x, y, w, h = bbox_coords["x"], bbox_coords["y"], bbox_coords["w"], bbox_coords["h"]
        orig_h, orig_w = image_rgb.shape[:2]
        x1, y1 = max(0, x), max(0, y)
        x2, y2 = min(orig_w, x + w), min(orig_h, y + h)
        if x2 <= x1 or y2 <= y1:
            raise ValueError("Invalid bbox coordinates.")
        input_image = image_rgb[y1:y2, x1:x2]
    else:
        raise ValueError(f"Unknown mode: {mode}")

    overlay, prob_map = run_segmentation(registry.get_seg_model(mode), input_image)
    return {"mode": mode, "mask_image": numpy_to_base64(overlay),
            "input_image": numpy_to_base64(input_image)}

# main.py
import sys
from pathlib import Path
sys.path.insert(0, str(Path(__file__).parent))

from fastapi import FastAPI, File, UploadFile, Form, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse
import json
from typing import Optional

from pipeline import ModelRegistry, run_classify_pipeline, run_segment_pipeline

```

```

BASE_DIR          = Path(__file__).parent.parent
CLASSIFIER_PATH   = BASE_DIR / "best_model_classifier_91_acc.pth"
SEG_FULL_PATH     = BASE_DIR / "full_original" / "attention_deeplabv3plus" /
"best_checkpoint.pth"
SEG_LAYERCAM_PATH = BASE_DIR / "layercam_crops" / "attention_deeplabv3plus" /
"best_checkpoint.pth"
SEG_BBOX_PATH     = BASE_DIR / "paper_crops" / "attention_deeplabv3plus" /
"best_checkpoint.pth"
FRONTEND_DIR      = BASE_DIR / "frontend"

app = FastAPI(title="Fracture Diagnosis API", version="1.0.0")
app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_methods=["*"], allow_headers=["*"])
app.mount("/static", StaticFiles(directory=str(FRONTEND_DIR)), name="static")

registry: Optional[ModelRegistry] = None

@app.on_event("startup")
async def startup_event():
    global registry
    print("Loading models...")
    registry = ModelRegistry(
        classifier_path=str(CLASSIFIER_PATH),
        seg_full_path=str(SEG_FULL_PATH),
        seg_layercam_path=str(SEG_LAYERCAM_PATH),
        seg_bbox_path=str(SEG_BBOX_PATH),
    )
    print("All models ready.")

@app.get("/")
async def root():
    index_path = FRONTEND_DIR / "index.html"
    return FileResponse(str(index_path)) if index_path.exists() else {"message": "API is
running."}

@app.get("/health")
async def health():
    return {"status": "ok", "models_loaded": registry is not None}

@app.post("/api/classify")
async def classify(file: UploadFile = File(...)):
    if registry is None:
        raise HTTPException(503, "Models are not loaded yet.")
    if file.content_type not in {"image/jpeg", "image/png", "image/jpg"}:
        raise HTTPException(400, f"Unsupported file type: {file.content_type}. Use JPEG or PNG.")
    file_bytes = await file.read()
    if not file_bytes:
        raise HTTPException(400, "Empty file.")
    try:
        return run_classify_pipeline(registry, file_bytes)

```

```

except Exception as e:
    raise HTTPException(500, str(e))

@app.post("/api/segment")
async def segment(
    file: UploadFile = File(...),
    mode: str = Form(...),
    bbox_coords: Optional[str] = Form(None),
):
    if registry is None:
        raise HTTPException(503, "Models are not loaded yet.")
    if mode not in {"layercam", "full", "bbox"}:
        raise HTTPException(400, f"Invalid mode: {mode}. Use: layercam | full | bbox")

    parsed_bbox = None
    if mode == "bbox":
        if bbox_coords is None:
            raise HTTPException(400, "bbox_coords is required for mode='bbox'")
        try:
            parsed_bbox = json.loads(bbox_coords)
            assert all(k in parsed_bbox for k in ("x", "y", "w", "h"))
        except Exception:
            raise HTTPException(400, 'bbox_coords must be JSON: {"x": int, "y": int, "w": int, "h": int}')

    file_bytes = await file.read()
    if not file_bytes:
        raise HTTPException(400, "Empty file.")
    try:
        return run_segment_pipeline(registry, file_bytes, mode, parsed_bbox)
    except Exception as e:
        raise HTTPException(500, str(e))

```

ДОДАТОК В ГРАФІЧНІ МАТЕРІАЛИ

Система класифікації та сегментації медичних зображень для діагностування переломів

Виконав: студент Панченко Дмитро Олександрович, група КА-22

Науковий керівник: проф., д.т.н., доцент Недашківська Надія Іванівна

Київ — 2026

ІПСА · КПІ ІМ. ІГОРЯ СІКОРСЬКОГО

Актуальність

2

Переломи кісток — одна з найпоширеніших травм у світі з наростаючою тенденцією

Щороку у світі реєструється близько 178 мільйонів нових переломів кісток (за даними Global Burden of Disease)

За статистикою, від 1,1 до 3,7% переломів пропускається лікарями вже при першому зверненні пацієнта.

Особливо критичною є ситуація у вечірньо-нічний час — саме тоді фіксується найбільше усіх діагностичних помилок, пов'язаних із переломами



Мета, об'єкт та предмет дослідження

3

Мета

Розв'язання задачі автоматичного виділення зон переломів на рентгенограмах шляхом побудови двокрокової системи «класифікація → сегментація».

Об'єкт

Процес автоматизованої діагностики переломів кісток за рентгенівськими знімками з використанням методів комп'ютерного зору та глибокого навчання

Предмет

Методи глибокого навчання для класифікації та сегментації медичних рентгенівських зображень, зокрема архітектури згорткових нейронних мереж та механізми візуальної інтерпретації результатів .

Постановка задачі

4

1. Провести огляд сучасних методів автоматизованої діагностики переломів кісток за рентгенівськими знімками;
2. Проаналізувати архітектури нейронних мереж для класифікації та сегментації медичних зображень;
3. Виконати підготовку та аугментацію набору даних;

4. Підзадача 1 — Бінарна класифікація

Визначення факту наявності перелому: клас «норма» або «перелом». При виявленні — генерація теплових карт для грубої локалізації зони ушкодження у вигляді bounding box.

5. Підзадача 2 — Піксельна сегментація

Точне виділення зони перелому у вигляді бінарної маски — кожному пікселю присвоюється мітка «перелом / фон». Розглядаються три стратегії подачі вхідних даних: повний знімок, фрагмент за CAM-рамкою, фрагмент за еталонною рамкою.

6. Результат - Єдина діагностична послідовність (від класифікації до сегментації)

Обидві підзадачі об'єднуються в єдину автоматизовану систему, яка приймає рентгенівський знімок на вхід і на виході надає точну локалізацію зони перелому.

Математична постановка задачі

Введемо простір вхідних даних $\mathcal{X} \subset \mathbb{R}^{H \times W \times C}$, елементами якого є рентгенівські знімки зображення розміру $H \times W$ пікселів з C каналами. Після застосування препроцесингу CLAHE Grayscale маємо $C = 1, H = W = 224$

Кожному зображенню $x \in \mathcal{X}$ відповідає трійка анотацій (y, M, B) , де:

- 1) $y \in \{0, 1\}$ – глобальна мітка класу (0 – норма, 1 – перелом);
- 2) $M \in \{0, 1\}^{H \times W}$ – бінарна маска сегментації, де $M_{ij} = 1$ означає належність пікселя $\{i, j\}$ до зони перелому;
- 3) $B = (x_{min}, y_{min}, x_{max}, y_{max}) \in \mathbb{R}^4$ – обмежувальна рамка переломної зони.

Математична постановка задачі (класифікація)

Класифікатор визначає ймовірність наявності перелому для кожного знімка:

$$\hat{y}_i = f_{cls}(x_i, \theta_{cls}),$$

де θ_{cls} – вектор параметрів(ваг) згорткової нейронної мережі.

Метою оптимізації є знаходження таких параметрів θ_{cls}^* , при яких функція втрат досягає мінімуму.

$$\theta_{cls}^* = \arg \min_{\theta_{cls}} \frac{1}{|\mathcal{D}_{train}|} \sum_{(x_i, y_i) \in \mathcal{D}_{train}} H_{cls}(f_{cls}(x_i, \theta_{cls}), y_i)$$

Вихідне передбачення класу визначається пороговуванням:

$$C_i = 1[f_{cls}(x_i, \theta_{cls}) \geq \tau], \quad \tau \in [0, 1]$$

При $C_i = 0$ обробка завершується; при $C_i = 1$ додатково генерується карта просторової уваги $\mathcal{A} \in \mathbb{R}^{H' \times W'}$

Математична постановка задачі (сегментація) ⁷

Задача піксельної сегментації полягає у знаходженні оптимальних параметрів Φ^*_{seg} моделі g_Φ , що мінімізують функцію втрат:

$$\Phi^*_{seg} = \underset{\Phi}{\operatorname{argmin}} \frac{1}{|\mathcal{D}_{train}|} \sum_{(x'_i, M_i) \in \mathcal{D}_{train}} H_{seg}(g_\Phi(x'_i), M_i),$$

де x'_i — вхідний параметр згідно з обраною стратегією локалізації, H_{seg} — функція втрат для сегментації.

Вхідний простір $\mathcal{X}'$ визначається однією з трьох стратегій:

$$x' = \begin{cases} x & \text{— повний знімок} \\ x|_{B_{CAM}} & \text{— фрагмент за CAM рамкою} \\ x|_{B_{GT}} & \text{— фрагмент за еталонною рамкою} \end{cases}$$

Передбачена маска отримується пороговуванням виходу моделі:

$$M'_{ij} = 1[g_\Phi(x')_{ij} \geq 0,5], \quad i \in [1, H], j \in [1, W]$$

Метрики оцінювання моделей ⁸

Класифікація

1. **Accuracy** — частка правильно класифікованих знімків
2. **Precision** — частка правильно визначених переломів серед усіх передбачених
3. **Recall** — частка виявлених переломів серед усіх реальних
4. **F1-score** — гармонічне середнє Precision та Recall

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

Сегментація

1. **Dice Score** — міра збігу передбаченої маски $\hat{M}$ з еталонною розміткою M
2. **IoU (Intersection over Union)** — відношення перетину до об'єднання двох масок

$$IoU = \frac{|M \cap \hat{M}|}{|M \cup \hat{M}|} = \frac{TP}{TP + FP + FN}$$

$$Dice = \frac{2|M \cap \hat{M}|}{|M| + |\hat{M}|} = \frac{2 * TP}{2 * TP + FP + FN}$$

Набір даних та препроцесинг

9

Набір даних містить 4 083 рентгенограми з трьох лікарень. Лише 717 із 4 083 знімків містять переломи. Охоплює дві анатомічні ділянки: рука, нога.

Етап підготовки даних

1. Нормалізація розміру

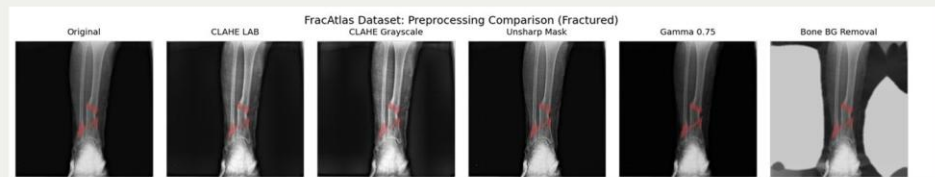
224×224 пікс. + нормалізація та стандартизація.

2. CLAHE Grayscale ★

Найкращий з 5 методів: виявляє переломні лінії без артефактів і без втрати деталей.

3. Аугментація

Слабкі перетворення — щоб не спотворювати тонкі структури.



Моделі класифікації переломів

10

Transfer learning · Поетапне розморожування

ResNet18

Базова модель

18 шарів. Легка та швидка у навчанні архітектура з залишковими з'єднаннями.

ResNet50

Глибша мережа

50 шарів. Значно глибша архітектура, здатна виявляти складніші патерни на рентгенівських знімках.

ConvNeXtV2-Tiny ★

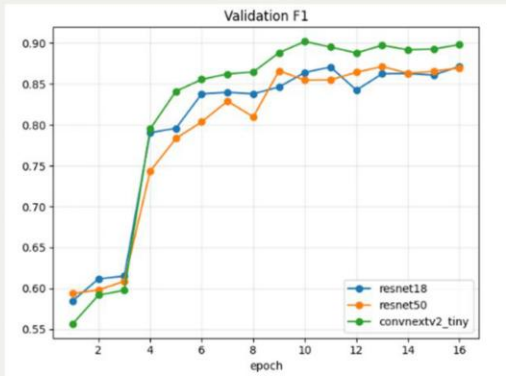
Сучасна архітектура.

Сучасна згортова архітектура, побудована на принципах трансформерів. Поєднує переваги класичних CNN із підходами Vision Transformer.

Результати класифікації

11

Модель	Accuracy	Precision	Recall	F1	AUC
ResNet18	92,3 %	88,8 %	85 %	86,9 %	96 %
ResNet50	92,8 %	89,8 %	86,1 %	87,8 %	95,7 %
ConvNeXtV2-Tiny	93,5 %	90,2 %	87,6 %	88,8 %	95,5 %

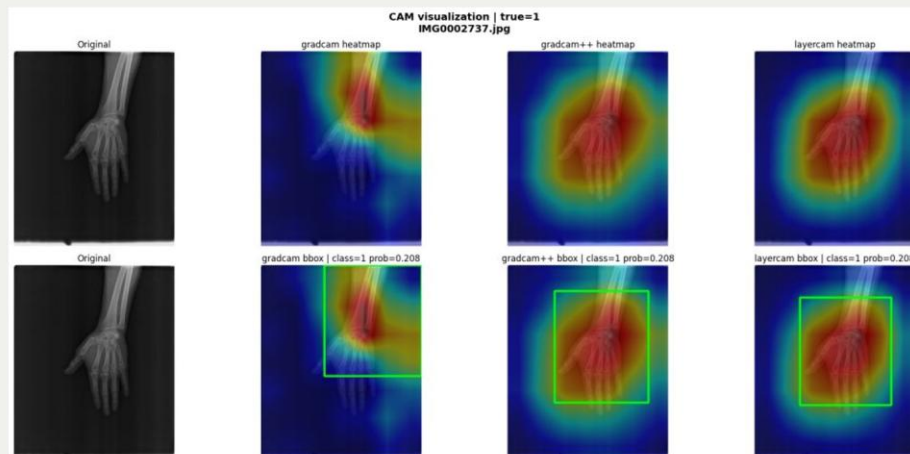


У клінічному контексті пріоритет — Recall: пропуск перелому небезпечніший, ніж хибне спрацювання.

Візуальна інтерпретація моделі

12

Ключова новизна підходу: Метод LayerCAM використовується не лише для інтерпретації — а як автоматичний проміжний крок системи: теплова карта → bounding box → вхід сегментатора. Це усуває потребу в ручній розмітці зони перелому.



Моделі сегментації переломів

13

5 архітектур · backbone ResNet50 pretrained

U-Net

Класична архітектура із пропускними з'єднаннями. Слугує базовою лінією завдяки простоті

U-Net++

Удосконалена версія U-Net. Вкладені skip-з'єднання для кращої передачі ознак різних масштабів.

DeepLabV3+

Аналізує зображення одночасно в кількох масштабах для точнішого виділення меж зони перелому.

Att. DeepLabV3+ ★

DeepLabV3+ із механізмом уваги — визначає *де* і *які* ознаки найважливіші.

Att. U-Net++

U-Net++ із увагою — автоматично відфільтровує нерелевантні ділянки.

Три стратегії вхідних даних

14

Якість локалізації — визначальний фактор точності сегментації!

1. Повний знімок

Базова лінія

Повний рентгенівський знімок без локалізації. Модель сама шукає перелом на тлі глибокого дисбалансу пікселів.

2. LayerCAM Crops

★ *Реальний сценарій*

Фрагмент вирізаний автоматично за bounding box від теплової карти.

3. GT Crops

Верхня межа точності

Фрагмент за еталонними анотованими рамками. Верхня межа точності

Результати сегментації

15

Модель	Dice Full	IoU Full	Dice LayerCam Crops	IoU LayerCam Crops	Dice Annotated Crops	IoU Annotated Crops
UNET	0,518	0,349	0,618	0,447	0,787	0,649
UNET++	0,536	0,366	0,596	0,425	0,773	0,630
DeepLabV3+	0,534	0,364	0,735	0,581	0,817	0,691
Attention DeepLabV3+	0,534	0,364	0,725	0,569	0,816	0,689
Attention UNET++	0,507	0,339	0,691	0,527	0,804	0,672

Ключові спостереження:

1. Усі моделі значно покращують результат при подачі обмежених зображень замість повного знімка
2. Найкращі результати на еталонних рамках — **DeepLabV3+** (Dice 0.817, IoU 0.691) та **Att. DeepLabV3+** (Dice 0.816, IoU 0.689)

Обрана модель для системи — Att. DeepLabV3+

За результатами повторних експериментів Att. DeepLabV3+ продемонстрував стабільнішу та вищу якість сегментації.

Механізм уваги забезпечує детальніший аналіз знімка.

Візуалізація результатів сегментації

16

Сегментація на оригінальних(повних) зображеннях



Сегментація на обрізаних зображеннях за допомогою теплової карти



Сегментація на обрізаних зображеннях через еталонну розмітку

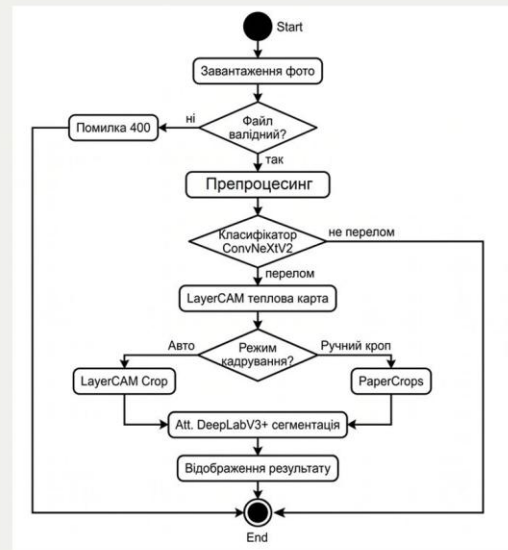


Програмна реалізація системи

17

Ключові етапи роботи системи:

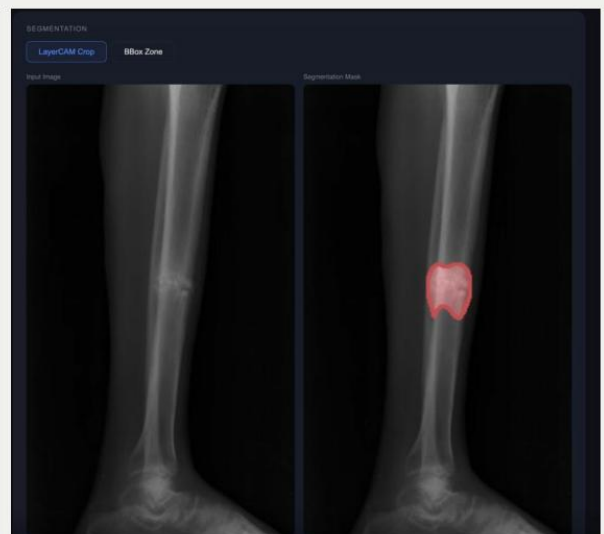
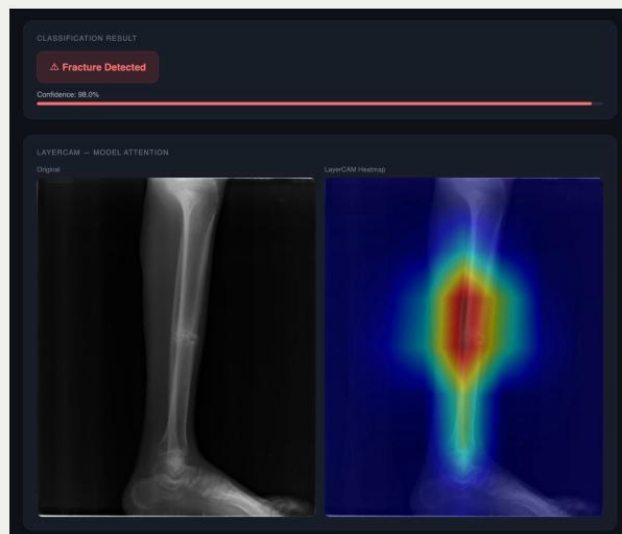
1. Завантаження рентгенівського знімка та валідація формату
2. Препроцесинг — CLAHE, resize, нормалізація
3. Класифікація — визначення наявності перелому
4. При виявленні перелому — генерація теплової карти LayerCAM та побудова bounding box
5. Сегментація зони перелому з виведенням маски на екран



Функціональна схема створеної системи

Демонстрація розробленої системи

18



Обмеження

19

1. Датасет містить лише 717 знімків із переломами — недостатньо для повноцінного навчання глибоких моделей
2. Глибокий дисбаланс класів на рівні пікселів (~1:20) ускладнює навчання сегментаційних моделей, що означає потребу в хорошій локалізації перелому
3. Через обмежені апаратні ресурси використовувались спрощені архітектури — потужніші SOTA-моделі типу SAM, DINO або ViT-based сегментатори не розглядались
4. Обмежені обчислювальні ресурси змусили скоротити кількість епох навчання, зменшити розмір вхідних зображень до 224×224 — що могло негативно вплинути на фінальну якість моделей

НАПРЯМИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

20

1. **Розширення датасету** Критичним кроком є збільшення обсягу навчальних даних — збір нових рентгенограм із різних клінік та географічних регіонів, робота над балансуванням класів і збагачення варіативності знімків.
2. **Багатокласова класифікація** Розширення від бінарної задачі до розпізнавання типу перелому, ступеня зміщення та анатомічної локалізації — що значно підвищить клінічну цінність системи для практикуючих лікарів.
3. **Клінічна валідація** Тестування системи на реальних медичних даних у співпраці з лікарнями, проспективне дослідження якості діагностики в порівнянні з роботою радіологів — як необхідний крок перед впровадженням.

Висновки

21

1. У роботі підтверджено актуальність автоматизованої діагностики переломів — переломи кісток є одним із найпоширеніших видів травм, а ручна інтерпретація рентгенограм залишається недосконалою через людський фактор.
2. За результатами порівняння методів препроцесингу обрано CLAHE Grayscale як найефективніший для виявлення ліній переломів.
3. Сучасна архітектура класифікатора перевершила класичні аналоги: Accuracy 93,5%, AUC 95,5%, F1 88,8%.
4. Ключовою новизною роботи є використання LayerCAM(теплова карта) як автоматичного проміжного кроку між класифікацією та сегментацією — без жодної ручної розмітки зони перелому.
5. Використання теплової карти суттєво підвищило якість сегментації — Dice Score зріс з 0,52 до 0,735.
6. Побудовано повністю діагностичну систему з веб-інтерфейсом — від завантаження рентгенівського знімка до отримання маски перелому

Дякую за увагу

Презентував: Панченко Дмитро Олександрович · КА-22

Науковий керівник: проф., д.т.н., доцент Недашківська Надія Іванівна

Київ — 2026

ІПСА · КЛІ ІМ. ІГОРЯ СІКОРСЬКОГО