Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ВИЗНАЧЕННЯ ВИПАДКІВ ПНЕВМОНІЇ
RESEARCH AND SOFTWARE IMPLEMENTATION OF PNEUMONIA DETECTION METHODS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м

Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

КАМІНСЬКИЙ Д.О.
(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.
(ПРІЗВИЩЕ та ініціали)

Рецензент ЗЕЛІК О.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
_____ Сергій СУББОТІН
"_____" _____2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КАМІНСЬКОГО Дениса Олексійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів визначення випадків пневмонії.
Research and Software Implementation of Pneumonia Detection Methods
керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 30 " вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року
3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблемної області. 2. Матеріали і методи. 3. Проєктування програми. 4. Розробка програми. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів	4-5 тижні	Розділ 2
	вирішення задачі.		
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження	9 тиждень	Розділ 5
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів до неї.	10-11тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Денис КАМІНСЬКИЙ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Валерій ЛЬОВКІН
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 135 с., 17 табл., 75 рис., 3 дод., 37 джерел.

ПНЕВМОНІЯ, ДІАГНОСТУВАННЯ, РЕНТГЕНІВСЬКІ ЗНІМКИ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, PYTHON, TYPESCRIPT, TENSORFLOW, DJANGO, SVELTE.

Об’єкт дослідження – процес побудови моделей діагностування пневмонії за медичними зображеннями.

Предмет дослідження – методи класифікації рентгенівських знімків грудної клітини.

Мета роботи – дослідження класифікації рентгенівських знімків грудної клітини на основі створення методу діагностування пневмонії грудної клітини та програмного забезпечення для підтримки прийняття рішень при діагностуванні пневмонії за рентгенівськими знімками для підвищення точності розпізнавання та зменшення складності моделей.

Матеріали, методи та технічні засоби: середовище розробки Visual Studio Code, мови програмування Python та Typescript, бібліотеки TensorFlow, Keras, фреймворки Django для серверної частини та Svelte для клієнтської, моделі на основі згорткових нейронних мереж, персональний комп’ютер з процесором Intel Core i7 та відеокартою NVIDIA з підтримкою CUDA під управлінням операційної системи Linux або або Windows 10 (з використанням Windows Subsystem for Linux).

Результати. Проведено аналіз предметної області, встановлено вимоги до програмного забезпечення. Досліджено та порівняно ефективність різних архітектур згорткових нейронних мереж, розроблено клієнт-серверний вебзастосунок з трьома рівнями доступу (пацієнт, лікар, адміністратор) та відповідною функціональністю.

Висновки. Розроблене програмне забезпечення дозволяє лікарям керувати медичними випадками, завантажувати рентгенівські знімки та отримувати класифікацію від ансамблю згорткових нейронних мереж як допоміжний інструмент. Наукова новизна роботи полягає у запропонованому методі діагностування пневмонії грудної клітини, що базується на використанні спрощеної резидуальної архітектури, що дозволяє підвищити точність розпізнавання, зменшити кількість параметрів утворених моделей і обчислювальні витрати під час навчання, при цьому зберігаючи високу точність класифікації за зниження складності моделей та практичну придатність моделі для автоматизованої підтримки медичної діагностики.

Галузь використання – медичне діагностування. Розроблене програмне забезпечення призначене для використання як система підтримки прийняття рішень для лікарів-радіологів та терапевтів.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 135 pages, 17 tables, 75 figures, 3 appendixes, 37 sources.

PNEUMONIA, DIAGNOSTICS, X-RAY IMAGES, CONVOLUTIONAL NEURAL NETWORKS, PYTHON, TYPESCRIPT, TENSORFLOW, DJANGO, SVELTE.

The object of study is the process of building models for diagnosing pneumonia based on medical images.

The subject of the study is the methods for classifying chest X-ray images.

The purpose of the work is to investigate the classification of chest X-ray images by developing a method for diagnosing pneumonia and software for decision support in pneumonia diagnosis based on X-ray images, with the aim of improving recognition accuracy and reducing model complexity.

Materials, methods, and tools: Visual Studio Code development environment, Python and TypeScript programming languages, TensorFlow and Keras libraries, Django framework for the server side and Svelte for the client side, models based on convolutional neural networks, IBM-compatible personal compute with an Intel Core i7 processor and an NVIDIA GPU with CUDA support under the control of the Linux or Windows 10 (using Windows Subsystem for Linux).

Results. The subject area was analyzed and software requirements were defined. The efficiency of various convolutional neural network architectures was examined and compared. A client–server web application was developed with three access levels (patient, doctor, administrator) and corresponding functionality.

Conclusions. The developed software allows doctors to manage medical cases, upload X-ray images, and obtain classifications from an ensemble of convolutional neural networks. The scientific novelty lies in the proposed pneumonia-diagnosis method based on a simplified residual architecture, which improves recognition accuracy, reduces the number of parameters and training

computational costs, while maintaining high classification accuracy, lower model complexity, and practical applicability for automated medical-diagnosis support.

The field of use is medical diagnostics. The developed software is intended to be used as a decision-support system for radiologists and physicians.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	– Convolutional Neural Network;
CRUD	– Create Read Update Delete;
DenseNet	– Dense Convolutional Network;
DRF	– Django REST Framework;
IDE	– Integrated development kit;
ML	– Machine Learning;
ORM	– Object Relational Mapper;
ReLU	– Rectified Linear Units;
ResNet	– Residual Network;
REST API	– Representational State Transfer Application Programming Interface;
SPA	– Single Page Application;
SVM	– Support Vector Machine;
ViTs	– Vision Transformers;
WSL	– Windows Subsystem for Linux;
БД	– база даних;
БНМ	– багатошарова нейронна мережа;
ЗНМ	– згорткова нейронна мережа;
КТ	– комп’ютерна томографія;
РСВ	– респіраторно-синцитіальний вірус;
СКБД	– система керування базами даних;
ШНМ	– штучна нейронна мережа.

ВСТУП

Пневмонія є серйозною загрозою для здоров’я людини, залишаючись однією з провідних причин смертності від інфекційних хвороб. Згідно з оцінками Всесвітньої організації охорони здоров’я, у 2019 році від пневмонії померло приблизно 2,5 мільйона осіб по всьому світу, з яких 672 тисячі припадали на дітей [1]. Такі дані свідчать про необхідність удосконалення методів діагностики для своєчасного виявлення та ефективного лікування цього захворювання.

Клінічні прояви пневмонії можуть варіюватися від легких до критично важких, включаючи задишку, лихоманку, продуктивний кашель, біль у грудній клітці та загальну слабкість [2]. Ці симптоми не лише погіршують якість життя, але й можуть призводити до серйозних ускладнень, особливо у вразливих груп населення, включаючи дітей та людей похилого віку. У зв’язку з цим точна та швидка діагностика пневмонії є життєво важливою.

Незважаючи на високий рівень підготовки медичних працівників, людський фактор, такий як втома чи суб’єктивність у оцінці медичних зображень, може призводити до діагностичних помилок. Аналіз рентгенівських знімків часто є складним завданням через неоднозначність візуальних даних. Це підкреслює потребу в інноваційних технологічних рішеннях, які можуть підтримувати лікарів у їхній роботі.

Використання штучного інтелекту, зокрема згорткових нейронних мереж, або Convolutional Neural Network (CNN), відкриває нові горизонти для автоматизації аналізу медичних зображень. Такі технології дозволяють швидко обробляти великі обсяги даних, виявляти характерні ознаки пневмонії та надавати лікарям цінні рекомендації. Подібні системи не покликані замінити медичних фахівців, а радше доповнюють їхній досвід, підвищуючи точність діагностики та ефективність лікувального процесу.

Метою даної роботи є дослідження класифікації рентгенівських знімків грудної клітини на основі створення методу діагностування пневмонії грудної

клітини та програмного забезпечення для підтримки прийняття рішень при діагностуванні пневмонії за рентгенівськими знімками для підвищення точності розпізнавання та зменшення складності моделей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Перший розділ присвячений детальному аналізу предметної області, пов’язаної з проблемою діагностування пневмонії. У розділі розглянуто основні поняття, класифікацію пневмонії, методи її діагностування. Також буде розглянуто існуючі методи машинного навчання для класифікації зображень.

1.1 Діагностування пневмонії як проблема класифікації

Пневмонія – це інфекція одного або обох легенів. Інфекцію можуть викликати віруси, бактерії або грибки. Коли людина заражається пневмонією, її альвеоли, або повітряні мішечки в легенях, заповнюються рідиною або гноєм. Це може призвести до таких симптомів, як кашель, лихоманка або озноб, які можуть мати різний ступінь тяжкості [3].

Стан здоров'я та вік людини, а також джерело інфекції можуть впливати на тяжкість симптомів. Більшість людей одужують за 2–4 тижні, але немовлята, люди похилого віку та люди з серцевими або легеневими захворюваннями піддаються ризику важкого перебігу хвороби і можуть потребувати лікування в лікарні [4].

1.1.1 Різновиди пневмонії

Залежно від етіології, пневмонію класифікують на бактеріальну, вірусну, грибкову та інші форми [3].

Бактеріальна пневмонія – це пневмонія, яка виникає внаслідок надмірного розмноження бактерій у легенях. Зазвичай вона виникає, коли організм ослаблений через хворобу, погане харчування, старість або порушення імунітету, і бактерії можуть проникнути в легені. Бактеріальна пневмонія може вражати людей будь-якого віку, але ви піддаєтеся більшому

ризику, якщо зловживаєте алкоголем, курите сигарети, ослаблені, нещодавно перенесли операцію, маєте респіраторне захворювання або вірусну інфекцію, або ослаблений імунітет [5].

Існує чотири типи бактеріальної пневмонії:

– позалікарняна пневмонія, або Community-acquired pneumonia (CAP): інфекція, що виникає після контакту з бактеріями під час повсякденного життя поза лікарнею або медичним закладом;

– лікарняна пневмонія, або Hospital-acquired pneumonia: пневмонія, що виникає через 48 годин після госпіталізації пацієнта. Пацієнт не потребує респіраторної підтримки під час перебування в лікарні;

– пневмонія, пов'язана з використанням апарату штучної вентиляції легенів, або Ventilator-associated pneumonia: пневмонія, що розвивається через 48 або більше годин після того, як пацієнт потребує респіраторної підтримки, наприклад, апарату штучної вентиляції легенів або інтубації;

– пневмонія, набута в медичних закладах, або Healthcare-acquired pneumonia: цей тип пневмонії виникає в інших типах медичних закладів, таких як амбулаторні клініки, будинки для літніх людей або центри діалізу, протягом трьох місяців після відвідування [3].

До поширених бактерій, що викликають бактеріальну пневмонію, належать, зокрема, Staphylococcus aureus, Streptococcus pneumoniae та Haemophilus influenzae. Коли ці поширені бактерії викликають пневмонію, вона називається типовою пневмонією. Менш поширені бактерії, такі як Mycoplasma pneumoniae, Legionella та Chlamydia psittaci, можуть викликати атипову пневмонію [3].

Вірусна пневмонія виникає, коли збудник вірусної природи уражає легеневу тканину. На її частку припадає приблизно третина всіх випадків пневмонії. Потрапляючи в дихальні шляхи, вірус спричиняє запалення та набряк легенів, що ускладнює надходження кисню. У більшості людей захворювання минає протягом кількох тижнів без ускладнень, але тяжкі форми становлять загрозу для життя [6].

Багато поширених вірусів можуть спричинити вірусну пневмонію, зокрема респіраторно-синцитіальний вірус (РСВ), риновірус (застуда), грип та коронавіруси, такі як COVID-19, серед інших [3].

Деякі групи людей більш схильні до вірусної пневмонії. Інфекція частіше зустрічається у літніх людей та маленьких дітей. Вагітні жінки також більш схильні до вірусної пневмонії, як і люди з ослабленою імунною системою внаслідок захворювань, таких як ВІЛ/СНІД, або через прийом певних ліків та лікування [3].

Грибкова пневмонія – це інфекція легенів, спричинена вдиханням грибкових спор, що призводить до запалення та пошкодження легеневої тканини. Вона в першу чергу вражає людей з ослабленою імунною системою, хоча іноді може виникати і в осіб зі здоровою імунною системою. Симптоми схожі на інші види пневмонії, такі як кашель, лихоманка та біль у грудях, і лікуються протигрибковими препаратами, хоча у важких випадках може знадобитися інше лікування або госпіталізація. Прикладом поширених джерел грибкової пневмонії є Coccidioides, Histoplasma та Blastomyces [3]-[7].

Іншим типом пневмонії є Аспіраційна пневмонія – це інфекція легенів, спричинена потраплянням їжі, рідини, вмістку шлунка або слини в дихальні шляхи або легені замість їх проковтування. Симптоми включають кашель, лихоманку, задишку та біль у грудях. Лікування зазвичай передбачає прийом антибіотиків і може включати кисневу терапію та інші підтримуючі заходи залежно від тяжкості захворювання [8].

1.1.2 Діагностика пневмонії

Діагностика пневмонії починається з клінічного обстеження, що включає аускультацію легень для виявлення хрипів, перкуторних змін та оцінку симптомів, таких як кашель, задишка, біль у грудях та лихоманка. Лабораторні тести, зокрема аналіз крові на лейкоцитоз, С-реактивний білок та

прокальцитонін, допомагають підтвердити запалення та диференціювати бактеріальну від вірусної етіології [9].

Інструментальні методи діагностики відіграють ключову роль: рентгенографія грудної клітини є стандартним методом для візуалізації інфільтратів, консолідації чи плеврального випоту. Комп'ютерна томографія (КТ) застосовується для більш детальної оцінки, особливо у складних випадках, тоді як ультразвук легень може виявляти плевральні зміни. Мікробіологічні дослідження, такі як посів мокротиння, ПЛР-тести на патогени чи бронхоскопія, використовуються для ідентифікації збудника [9].

1.1.3 Лікування та профілактика пневмонії

Лікування пневмонії залежить від етіології: антибіотики призначаються для бактеріальних форм (наприклад, амоксицилін чи макроліди), противірусні препарати – для вірусних (як озельтамівір при грипі), а антимікотики – для грибкових. Підтримуюча терапія включає оксигенотерапію, гідратацію та симптоматичне лікування [3].

Профілактика охоплює вакцинацію проти пневмококу та грипу, гігієну рук та уникнення факторів ризику. Дотримання здорового способу життя, такого як відмова від куріння, регулярні фізичні вправи та збалансоване харчування, також знижує ризик захворювання [3].

1.2 Огляд існуючих методів машинного навчання

Нижче наведено огляд декількох найпопулярніших методів машинного навчання для класифікації зображень.

Згорткові нейронні мережі (ЗНМ) – це різновид глибинних нейронних мереж, спеціально пристосованих для обробки зображень. Вони наслідують принципи роботи зорової кори: кожен штучний нейрон реагує лише на невелику ділянку зображення (рецептивне поле), а перекриття цих ділянок

дозволяє мережі поступово формувати повну картину. Завдяки згортковим фільтрам CNN автоматично виділяють характерні ознаки зображень і стали стандартом у задачах комп’ютерного зору [10].

Залишкові нейронні мережі або Residual Networks (ResNets) – це тип архітектури глибоких нейронних мереж, який вводить skip connections (пропускові з’єднання), що дозволяють «перескакувати» через шари. Завдяки цьому мережа навчається не прямій відповідності між вхідними та вихідними даними, а залишковій функції – різниці між ними. Такий підхід зменшує проблему зникаючого градієнта й дає змогу ефективно тренувати сотні та тисячі шарів, що суттєво покращує точність у задачах розпізнавання зображень [11].

Методи ансамблю в машинному навчанні поєднують кілька окремих моделей для створення єдиної, більш точної та надійної кінцевої моделі. Об'єднуючи сильні сторони різних моделей, методи ансамблю, такі як bagging (наприклад, Random Forest), boosting (наприклад, AdaBoost, Gradient Boosting) та stacking, можуть значно поліпшити результати прогнозування та зменшити кількість помилок порівняно з будь-якою окремою моделлю. Процес передбачає навчання різних базових моделей, а потім об'єднання їх прогнозів за допомогою таких технік, як голосування для класифікації або усереднення для регресії [12].

Трансформатори зору або Vision Transformers (ViTs) – це клас моделей які адаптують ідею трансформерів з обробки природної мови до комп’ютерного зору. Замість згорток вони розбивають зображення на фрагменти та аналізують їх за допомогою механізму самоуваги, що дає змогу виявляти глобальні залежності між частинами зображення. Цей підхід показав конкурентні результати у класифікації та інших задачах комп’ютерного зору, особливо при навчанні на великих наборах даних [13].

Машини опорних векторів або Support vector machine (SVM) застосовуються для класифікації та регресії, зокрема у сфері обробки зображень. Метод будує гіперплощину, яка розділяє класи з максимально

можливою відстанню до найближчих точок (опорних векторів). Завдяки цьому SVM ефективно працюють з багатовимірними ознаками та добре протидіють перенавчанню, залишаючись популярним класичним методом навіть у добу глибинного навчання [14].

1.3 Висновки до першого розділу

У даному розділі було розглянуто різновиди й процес діагностування пневмонії, що надає розуміння важливості даного проєкту в медичній сфері. Також було розглянуто різні методи машинного навчання, які використовуються для класифікації зображень. Далі буде детально розглянуто моделі на основі глибоких згорткових нейронних мереж щодо особливостей використання у задачі, що розглядається.

Пневмонія залишається однією з найпоширеніших хвороб у світі, а процес її діагностики має значні обмеження, що зумовлює потребу в удосконаленні методів виявлення. Для вирішення цієї задачі необхідно виконати наступні завдання:

– аналіз предметної області та формування технічного завдання;
– експериментальне дослідження різних архітектур CNN;
– експериментальне дослідження архітектури з найвищою точністю для її подальшого вдосконалення на основі запропонованого методу;
– створення вебінтерфейсу для використання створенного методу;
– проведення тестування програмного забезпечення для оцінки його ефективності та надійності.

2 МАТЕРІАЛИ І МЕТОДИ

В даному розділі має бути розглянуто існуючі методи інтелектуального аналізу даних та представлено задачу діагностування пневмонії. Основним результатом має стати опис власного методу діагностування.

2.1 Огляд методів інтелектуального аналізу даних для діагностування пневмонії

Згорткові нейронні мережі (ЗНМ) складається з вхідного, вихідного та кількох прихованих шарів. На відміну від класичної моделі багатошарової нейронної мережі (БНМ), приховані шари ЗНМ зазвичай мають спеціалізовану структуру, включаючи згорткові, агрегувальні, повнозв’язні шари та шари нормалізації [10].

На рис. 2.1 наведено схему ЗНМ.

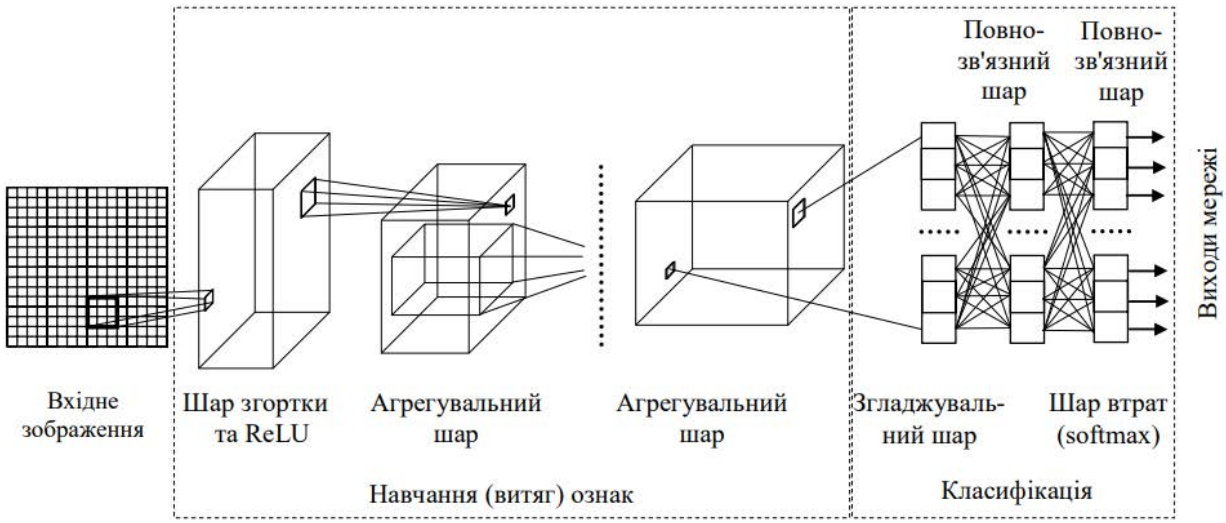

Рисунок 2.1 – Схема архітектури типової згорткової нейронної мережі [10]

Згортковий шар (convolutional layer) виконує операцію згортки над вхідними даними й передає результат далі. Ця операція моделює реакцію окремого нейрона на зоровий стимул. Кожен нейрон працює лише в межах свого рецептивного поля. Параметри шару – це набір фільтрів (ядер), які

охоплюють всю глибину вхідних даних, але мають невеликий розмір у просторі. Під час прямого проходу кожен фільтр ковзає по ширині та висоті, формуючи карту активацій. У такий спосіб мережа “вчиться”, які фільтри спрацьовують при виявленні певних ознак у конкретних областях зображення. Об’єднання карт активацій по глибині формує вихід згорткового шару, де кожен елемент можна інтерпретувати як реакцію нейрона на невелику ділянку входу [10].

Агрегувальний шар (pooling layer) виконує підзразкування даних: він зменшує просторовий розмір подання, поєднуючи групи виходів нейронів у більш компактне представлення. Це знижує кількість параметрів, обчислювальні витрати та зменшує ризик перенавчання. Зазвичай агрегувальні шари вставляють після кількох згорткових. Вони роблять мережу менш чутливою до дрібних зсувів об’єктів на зображенні. Операція виконується окремо для кожного каналу вхідних даних [10].

Максимізаційне агрегування (max pooling) бере найбільше значення з кожного фрагмента. Таким чином вхід ділиться на неперекривні прямокутники, з яких вибирається максимум [10].

Усереднювальне агрегування (average pooling) замінює кожен фрагмент його середнім значенням [10].

Шар Rectified Linear Unit (ReLU) застосовує просту нелінійну функцію активації, що не насичується. Це посилює здатність мережі відображати складні залежності, не змінюючи структури рецептивних полів. Окрім ReLU іноді використовують й інші функції, наприклад, насичувальні гіперболічний тангенс та сигмоїдна функція [10].

Згладжувальний шар (flatten layer) перетворює багатовимірні дані у вектор, щоб передати їх до повноз’єднаних шарів [10].

Повноз’єднаний шар (fully connected layer) з’єднує всі нейрони попереднього шару з усіма наступного, забезпечуючи високорівневу інтеграцію ознак. За своєю суттю він відповідає класичним шарам багатошарової мережі [10].

Шар втрат (loss layer) визначає функцію, за якою вимірюється різниця між передбаченими результатами та правильними мітками класів. Саме він задає критерій, за яким мережа “карається” за помилки й навчається [10].

Навчання ЗНМ відбувається за допомогою алгоритму зворотного поширення похибки [10].

Таким чином, після розгляду ключових складових і принципів роботи згорткових мереж можна переходити до аналізу конкретних архітектур, що застосовуються для задач обробки зображень загалом і задачі діагностування пневмонії зокрема.

2.1.1 Архітектура AlexNet

AlexNet – це згорткова нейронна мережа, опублікована у роботі [15] Алексом Крижевським у співпраці з Іллею Суцкевером та під керівництвом Джеффрі Хінтона. Широке визнання модель отримала після перемоги на конкурсі ILSVRC-2012, де її помилка top-5 становила лише 15,3%, що було майже на 11% нижче, ніж у найближчого конкурента [16].

Ключовим висновком було те, що велика глибина мережі значно покращує її продуктивність, проте потребує значних обчислювальних ресурсів. Застосування графічних процесорів під час навчання зробило можливим ефективне використання такої моделі. AlexNet включала п’ять згорткових шарів, об’єднаних із шарами об’єднання, а також кілька повноз’єднаних шарів наприкінці. Вона також стала однією з перших архітектур, де активно використовувалась функція активації ReLU для пришвидшення збіжності [16].

На рис. 2.2 наведено ілюстрацію архітектури AlexNet.

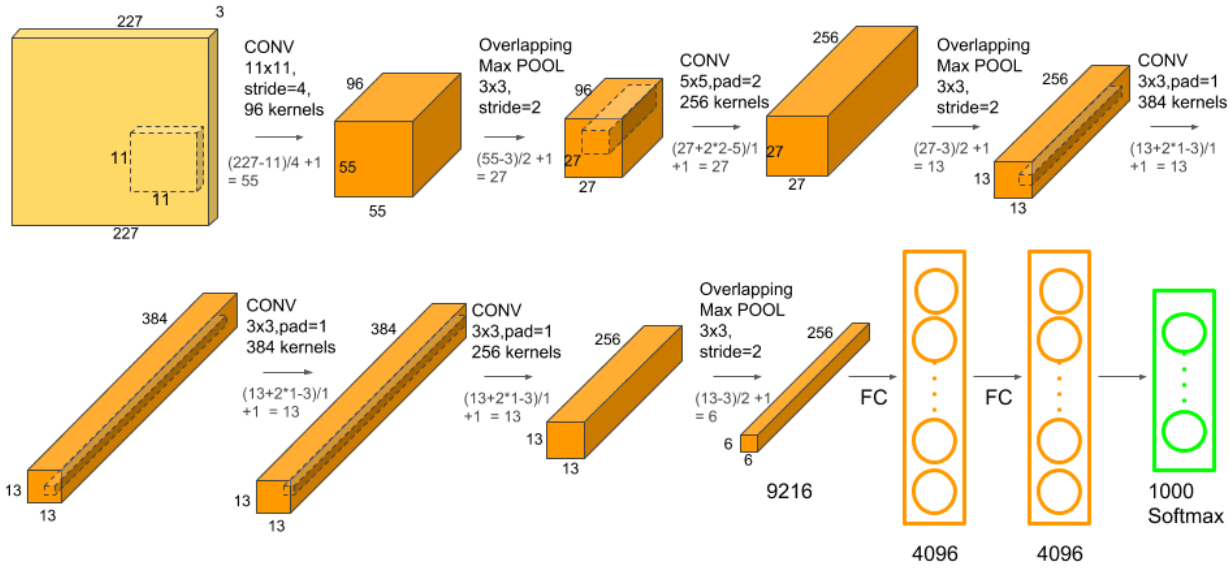

Рисунок 2.2 – Ілюстрація архітектури AlexNet [17]

2.1.2 Архітектура VGG16

VGG16 – це згорткова нейромережева модель, запропонована у роботі [18] групою дослідників з Оксфордського університету (Карен Сімонян та Ендрю Зіссерман). Модель була презентована у 2014 році на змаганні ILSVRC, де показала точність top-5 у 92,7% на тестовому наборі ImageNet. Її особливістю стала проста, але систематична ідея: використовувати послідовність згорткових фільтрів невеликого розміру (3×3), замість більших ядер, які застосовувались у попередніх моделях, включно з AlexNet. Це дозволило значно поглибити мережу, зберігаючи обчислювану ефективність. VGG16 включає 16 шарів із параметрами, серед яких 13 згорткових та 3 повноз’єднані. Навчання моделі потребувало кілька тижнів на сучасних на той час GPU NVIDIA Titan Black [19].

На рис. 2.3 наведено ілюстрацію архітектури VGG16.

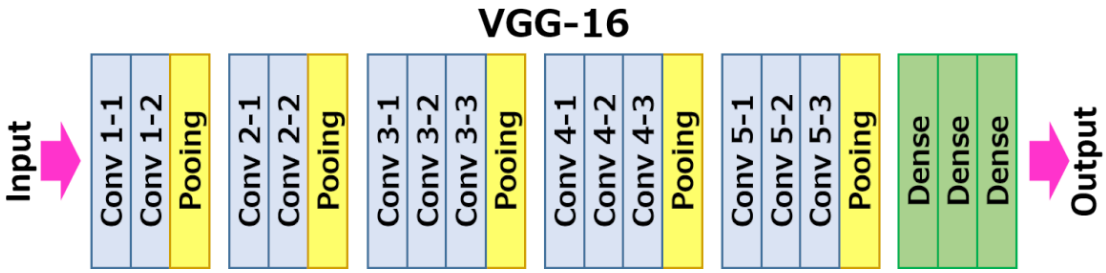

Рисунок 2.3 – Ілюстрація архітектури VGG16 [20]

2.1.3 Архітектура VGG19

VGG19 є подальшим розширенням попередньої моделі. Її архітектура аналогічна VGG16, проте містить 19 шарів з параметрами, що робить її ще більш глибокою. Вона включає додаткові згорткові шари та шари максимального об’єднання, завдяки чому здатна краще відображати складні структури ознак у зображеннях. Попри дещо кращі результати в окремих тестах, збільшення кількості шарів суттєво підвищує обчислювальні витрати та кількість параметрів, що робить VGG19 менш ефективною у використанні ресурсів порівняно з VGG16 [21].

На рис. 2.4 наведено ілюстрацію архітектури VGG19.

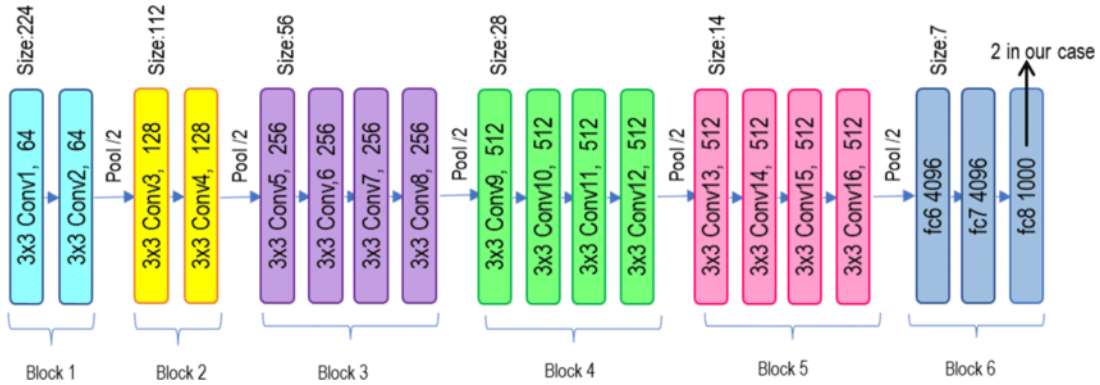

Рисунок 2.4 – Ілюстрація архітектури VGG19 [22]

2.1.4 Архітектура ResNet50

Residual Network (ResNet) розшифровується як "Залишкова мережа", була представлена у роботі [23] Kaiming He, Xiangyu Zhang, Shaoqing Ren та Jian Sun у 2015 році. Архітектура ґрунтується на використанні залишкових блоків із пропускними з'єднаннями (skip connections), які дають змогу ефективно тренувати глибокі нейронні мережі та мінімізувати проблему зникання градієнта [24].

Мережа ResNet-50 складається з 50 шарів (48 згорткових, одного шару MaxPool та одного шару глобального середнього об'єднання). Вона продемонструвала високу ефективність на ImageNet і заклала основу для подальших надглибоких моделей. Варто відзначити, що перші експерименти проводилися на ResNet-34, після чого архітектура була розширена [24].

На відміну від VGG, яка використовувала велику кількість параметрів, ResNet завдяки залишковим блокам досягала подібної або вищої точності при меншій складності моделі. Таким чином, вона стала однією з найбільш впливових архітектур у сучасному комп'ютерному баченні [24].

На рис. 2.5 наведено ілюстрацію архітектури ResNet50.

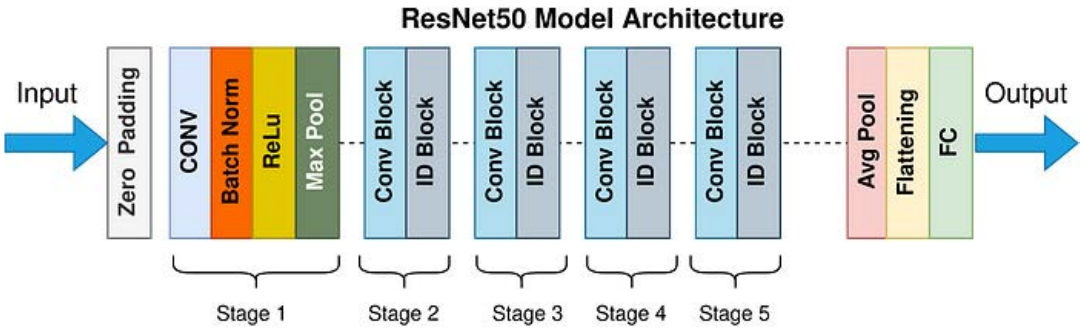

Рисунок 2.5 – Ілюстрація архітектури ResNet50 [25]

2.1.5 Архітектура DenseNet

Dense Convolutional Network (DenseNet) – це згорткова нейронна мережа, представлена Gao Huang, Zhuang Liu, Laurens van der Maaten і Kilian

Q. Weinberger у роботі [26], опублікованій у 2017 році. DenseNet революціонізувала сферу комп'ютерного зору, запропонувавши нову модель зв'язності в CNN, яка спрямована на вирішення таких проблем, як повторне використання ознак, зникаючі градієнти та ефективність параметрів. На відміну від традиційних архітектур CNN, де кожен шар пов'язаний тільки з наступними шарами, DenseNet встановлює прямі зв'язки між усіма шарами в блоці. Ця щільна зв'язність дозволяє кожному шару отримувати карти ознак від усіх попередніх шарів як вхідні дані, сприяючи широкому потоку інформації по всій мережі [27].

На рис. 2.6 наведено ілюстрацію архітектури DenseNet.

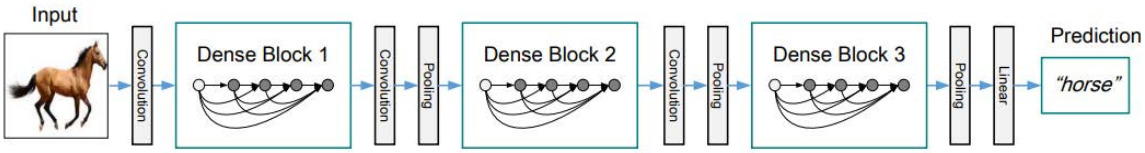

Рисунок 2.6 – Ілюстрація архітектури DenseNet [28]

2.1.6 Архітектура MobileNet

MobileNet — это семейство легких и эффективных архитектур сверточных нейронных сетей (CNN), разработанных Google. Вони розроблені для невеликих розмірів, низької затримки та низького енергоспоживання, що робить їх придатними для виведення на пристрої та периферійних обчислень на пристроях з обмеженими ресурсами, таких як мобільні телефони та вбудовані системи. Спочатку вони були розроблені для ефективної роботи на мобільних пристроях з TensorFlow Lite [29].

MobileNetV1 була опублікована у роботі [30] у квітні 2017 року. Серед ключових авторів були Andrew G. Howard, Menglong Zhu та ін.. Його основною архітектурною інновацією було включення роздільних по глибині згортків. Він був вперше розроблений Лораном Сіфре під час стажування в Google Brain

у 2013 році як архітектурна варіація AlexNet для поліпшення швидкості збіжності та розміру моделі [29].

На рис. 2.7 наведено ілюстрацію архітектури MobileNetV1.

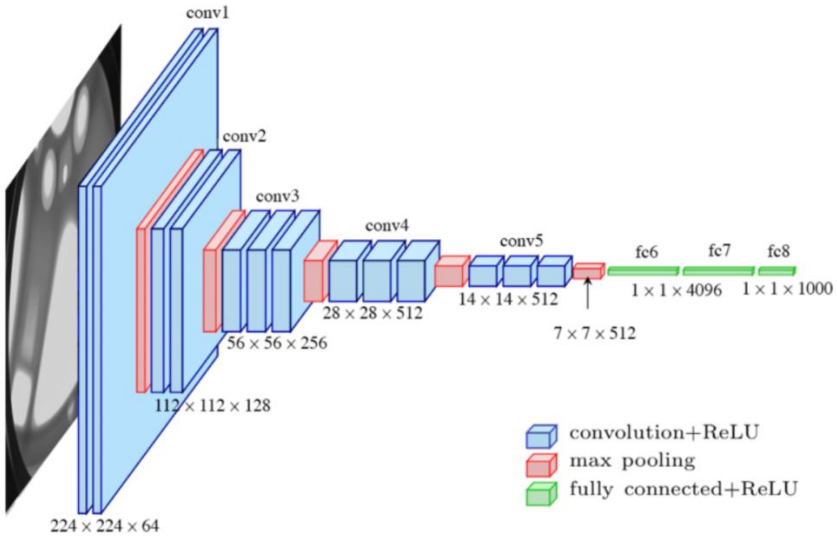

Рисунок 2.7 – Ілюстрація архітектури MobileNetV1 [31]

Для зручності аналізу ефективності згаданих архітектур у табл. 2.1 наведено їх порівняльні характеристики та результати на базі ImageNet-1K.

Таблиця 2.1 – Порівняння основних характеристик різних архітектур

Модель	Кіль-кість шарів	Кількість параметрів	Розмір вхідного зображення	ImageNet Top-5 Точність (%)
AlexNet	8	61.1 млн	227x227	N / A
VGG16	16	138.4 млн	224x224	~ 90.0
VGG19	19	143.7 млн	224x224	~ 90.1
ResNet50	50	25.6 млн	224x224	~ 92.1
DenseNet-121	121	8.1 млн	224x224	~ 92.3
MobileNetV1	55	4.3 млн	224x224	~ 70.8

2.2 Аналіз набору даних

Для тренування та тестування моделей в основі розв'язання задачі діагностування пневмонії в цій роботі використано відкритий набір даних Kaggle [32], що містить 5 863 рентгенівські зображення грудної клітини. Дані поділено на три підмножини:

– тренувальна вибірка: 5216 зображень (3875 із пневмонією, 1341 без);

– тестова вибірка: 624 зображення (390 із пневмонією, 234 без);

– валідаційна вибірка: 16 зображень (8 із пневмонією, 8 без).

Набір має суттєвий дисбаланс класів, тому застосовано техніки аугментації, описані в наступному розділі.

На рис. 2.8 подано розподіл даних між вибірками та класами оригінального датасету.

На рис. 2.9 наведено приклади зображень з набору даних.

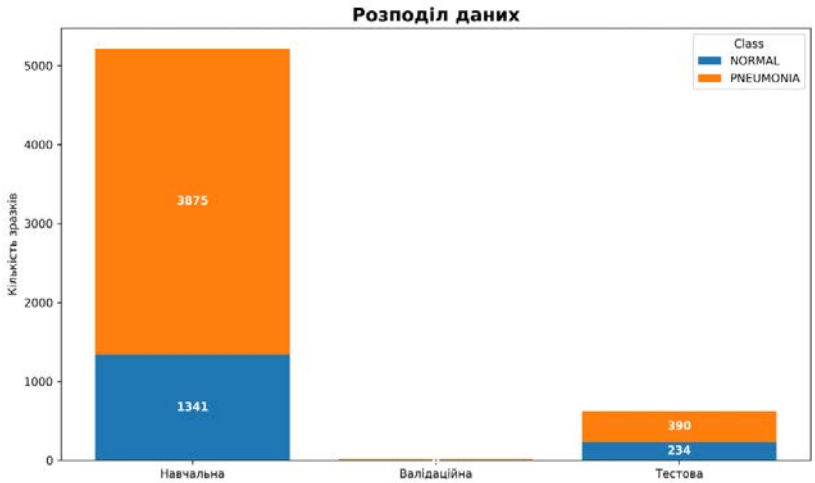

Рисунок 2.8 – Розподіл даних між вибірками та класами оригінального датасету

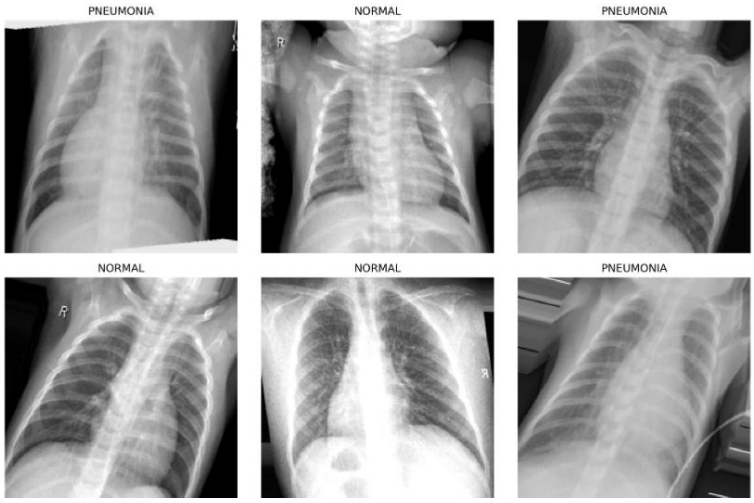

Рисунок 2.9 – Приклади зображень з набору даних

2.3 Метод діагностування пневмонії грудної клітини

Задача діагностування пневмонії полягає у класифікації рентгенівських знімків грудної клітини з метою виявлення пневмонії. Вхідними даними для цієї задачі, а відповідно і тими даними, які мають бути сформовані на етапі передобробки, є дані, які формуються шляхом приведення кожного до числового формату у вигляді матриці пікселів (значення у діапазоні 0–255). Далі при розв’язанні задачі діагностування пневмонії визначається ймовірність належності знімка до одного з двох класів:

– 0 – відсутність пневмонії;

– 1 – наявність пневмонії.

Це і є вихідними даними.

Для вирішення поставленої задачі у складі запропонованого в роботі методу діагностуваня пневмонії грудної клітини виділено наступні етапи:

– підготовка та аугментація даних;

– побудова архітектури згорткової нейронної мережі;

– навчання моделі;

– отримання і перевірка результатів розпізнавання моделі.

На першому етапі формується набір даних із рентгенівських знімків. Аугментація даних виконується окремим скриптом на стороні попередньої

обробки та застосовується для підвищення узагальнюючої здатності. Основні параметри аугментації:

– зміна масштабу (rescale);
– випадкове обертання (rotation_range);
– зміна яскравості (brightness_range);
– горизонтальні та вертикальні зсуви (width_shift_range, height_shift_range);
– зсув із деформацією (shear_range);
– горизонтальне відображення (horizontal_flip) .

Окремо варто зазначити, що валідаційний набір був сформований на ранньому етапі дослідження, з оригінальним обсягом даних. Цей набір було зафіксовано («заморожено») для забезпечення коректної порівнюваності результатів між різними архітектурами моделей та різними конфігураціями датасету – як у випадку роботи без аугментацій, так і після розширення даних. Подальше збільшення тренувальної вибірки за рахунок аугментованих варіантів не вплинуло на склад валідаційної вибірки, оскільки її зміна порушила б узгодженість експериментів. У результаті фінальний обсяг валідаційної вибірки залишився відносно невеликим (близько 900 зображень), що, попри підвищений рівень статистичного шуму на ранніх епохах, виявилося достатнім для подальшої стабільної оцінки якості моделей.

Завантаження зображень здійснюється у режимі «на льоту» через генератор даних (ImageDataGenerator), що використовується для оптимізації пам’яті під час тренування через обмеження апаратних ресурсів.

На другому етапі має бути сформовано архітектуру згорткової нейронної мережі з використанням резидуальних з’єднань.

Модель поєднує базові згорткові блоки з пропускними з’єднаннями, що дає змогу зберігати стабільний градієнтний потік і уникати деградації точності при збільшенні глибини. Архітектура передбачає послідовне зростання кількості фільтрів (32 – 64 – 128 – 256) із одночасним зменшенням просторової роздільності ознак. У кожному блоці застосовано Batch Normalization для

стабілізації навчання та функцію активації LeakyReLU, яка допомагає уникати «мертвих» нейронів. Після екстракції ознак виконується глобальне усереднення, повнозв’язний шар та вихідний шар для бінарної класифікації. Загалом модель містить близько 1,26 млн параметрів і є компактним, але достатньо глибоким рішенням для задачі класифікації зображень загалом і задачі діагностування пневмонії зокрема.

На рис. 2.10 наведена структура модифікованої моделі.

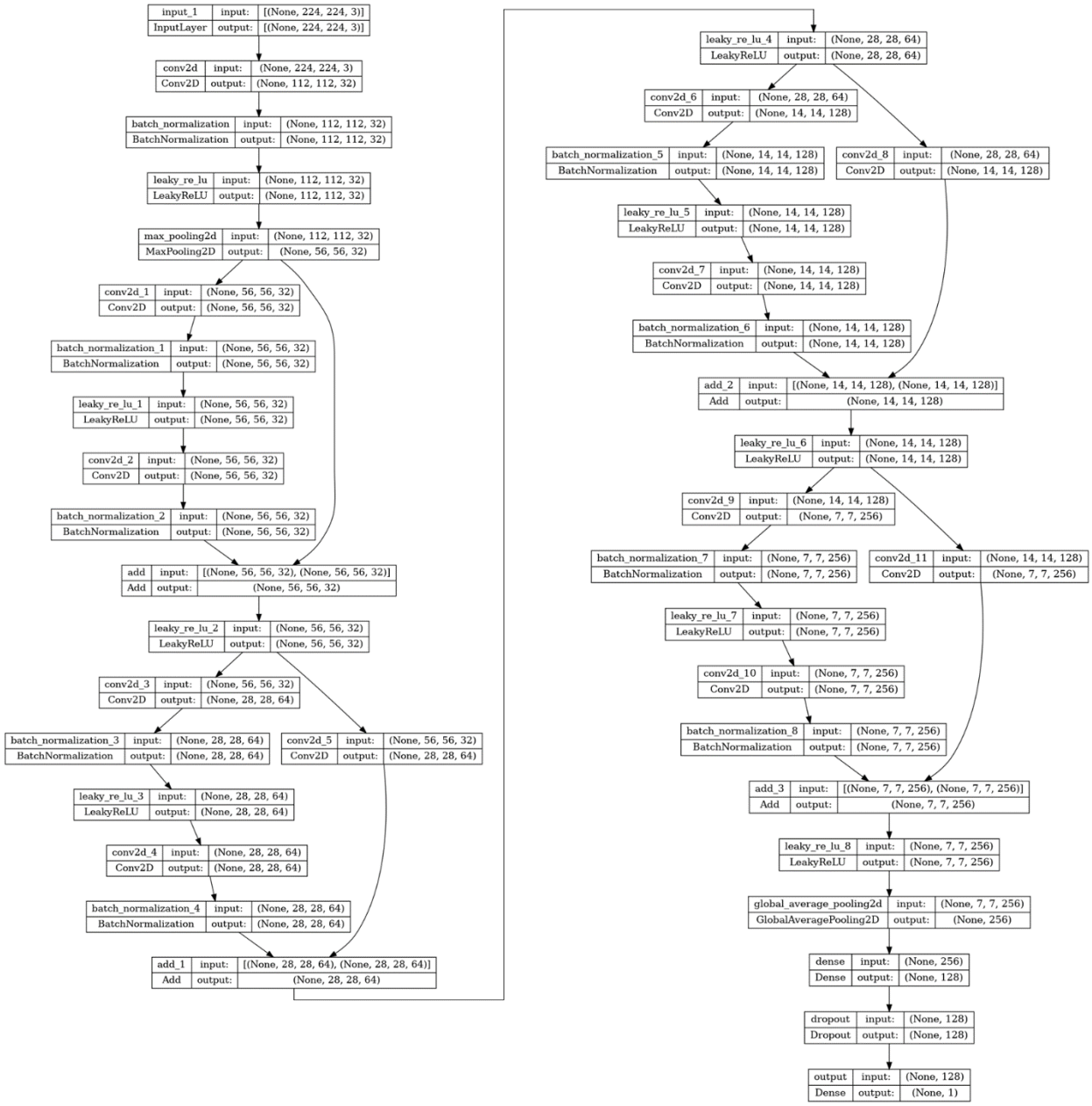

Рисунок 2.10 – Структура створеної м1оделі діагностування пневмонії

На третьому етапі модель навчається за допомогою оптимізатора Adam з використанням функції втрат binary crossentropy. У процесі навчання застосовуються зворотні виклики:

– EarlyStopping – для запобігання перенавчанню у випадку, якщо точність не зростає протягом кількох епох;

– ModelCheckpoint – для збереження найкращої версії моделі;

– ReduceLROnPlateau – для адаптивного зменшення швидкості навчання.

Після завершення навчання модель використовується для виконання класифікації на знімках грудної клітини, для яких модель, створена в основі методу, ще не застосовувалась. Під час експериментального дослідження такоє оцінюється на валідаційній вибірці, де обчислюються метрики для оцінки ефективності архітектури. Детальніше про результати тестування описано в підрозділі 5.4.

Відповідно для реалізації етапу отримання результатів розпізнавання моделі запропонованого методу в реальних умовах запропоновано алгоритм використання моделі для діагностування пневмонії, що виглядає наступним чином:

– лікар завантажує рентгенівський знімок грудної клітини у систему;

– зображення проходить попередню обробку;

– підготовлений масив подається на вхід мережі;

– модель видає прогноз у вигляді ймовірності приналежності випадку до класу наявності пневмонії;

– інтерфейс відображає висновок разом зі знімком для зручності користувача.

2.4 Висновки до другого розділу

У даному розділі було розглянуто традиційні архітектури згорткових нейронних мереж та проведено їх порівняння за ключовими характеристиками.

Для розробки програмного застосунку з діагностики пневмонії за рентгенівськими знімками було прийнято рішення створити власну архітектуру на основі принципів резидуальних мереж (ResNet). Запропонована структура передбачає використання блоків зі скороченими зв’язками, що сприяє кращому поширенню градієнтів та більш ефективному навчанню глибокої моделі. Додатково застосовано методи регуляризації та Dropout для зниження ризику перенавчання. Для тренування та тестування моделі використано відкритий набір даних рентгенівських знімків грудної клітини (Kaggle), що містить зображення здорових пацієнтів та пацієнтів із пневмонією.

Наукова новизна роботи полягає у запропонованому методі діагностування пневмонії грудної клітини, що базується на використанні спрощеної резидуальної архітектури, що дозволяє підвищити точність розпізнавання, зменшити кількість параметрів утворених моделей і обчислювальні витрати під час навчання, при цьому зберігаючи високу точність класифікації за зниження складності моделей та практичну придатність моделі для автоматизованої підтримки медичної діагностики.

3 ПРОЄКТУВАННЯ ПРОГРАМИ

В даному розділі має бути розглянуто основні особливості програмних засобів, за допомогою яких буде розроблятися програмне забезпечення.

3.1 Вибір мови програмування для створення бекенд-частини додатку

Для створення моделей машинного навчання найдоцільнішим рішенням є використання мови Python. Її перевага полягає у зрозумілому синтаксисі, потужній екосистемі фреймворків та широких можливостях застосування.

Завдяки простоті у написанні коду Python робить процес роботи з даними зручним і послідовним: від їх перевірки та очищення до аналітичної обробки. Такий підхід не лише спрощує індивідуальну розробку, а й полегшує командну співпрацю, оскільки код стає більш прозорим і доступним для розуміння іншими спеціалістами.

Великий вибір готових бібліотек звільняє розробника від потреби повторно реалізовувати стандартні алгоритми, що знижує ймовірність помилок і дозволяє зосередитися на реалізації унікальних функцій проєкту. Для роботи з нейронними мережами доступні бібліотеки TensorFlow, Keras, PyTorch, а для веброзробки – FastAPI та Django. Використання таких інструментів значно підвищує ефективність і швидкість створення програмних рішень.

Ще однією вагомою перевагою є універсальність: Python однаково добре працює в середовищах Windows, macOS та Linux. Додатковим аргументом на користь вибору виступає велика міжнародна спільнота розробників, що спрощує пошук навчальних матеріалів та рішень для технічних труднощів.

У табл. 3.1 наведено порівняння обраної мови програмування з іншими найпопулярнішими мовами програмування.

Таблиця 3.1 – Порівняння мов програмування

Критерій порівняння	Python	JavaScript	R	C#
Простота вивчення	Дуже проста	Середня	Середня	Складний
Швидкість виконання	Низька	Середня	Висока	Висока
Підтримка та спільноста	Дуже широка, глобальна	Дуже широка, орієнтована на веб	Висока, академічна	Висока, переважно enterprise
Бібліотеки для ML / Data Science	Висока (TensorFlow, PyTorch, scikit-learn)	Низький–середній (TensorFlow.js)	Високий (caret, randomForest, xgboost)	Середня (ML.NET)
Робота з даними	Дуже зручна (pandas, NumPy, matplotlib)	Обмежена	Дуже зручна (tidyverse, ggplot2, dplyr)	Обмежена
Підтримка вебфреймвор-ків	Висока (FastAPI, Django)	Висока (Express.js, Next.js)	Низька (Shiny)	Висока (ASP.NET)

3.2 Вибір бекенд фреймворку

У табл. 3.2 наведено порівняння трьох сучасних вебфреймворків на Python: Django, Flask та FastAPI.

Таблиця 3.2 – Порівняння вебфреймворків Django та Flask

Критерій порівняння	Django	Flask	FastAPI
1	2	3	4
Рівень складності	Високий	Середній	Середній
Масштабованість	Висока (структуровані великі проекти)	Середня	Висока (для мікросервісів та API)

Продовження таблиці 3.2

1	2	3	4
Швидкість розробки	Середня	Висока	Висока
Інтеграція з ORM	Django ORM	SQLAlchemy	Будь-яка (часто SQLAlchemy)
Вбудовані можливості	Адмін-панель, аутентифікація, міграції, шаблонізатор, middleware	Мінімум (додаються вручну за потреби)	Автоматична документація API, валідація даних
Ресурсоємність	Висока (більше модулів, складна структура)	Низька (мінімальний core)	Середня (асинхронність додає витрати)
Спільнота та підтримка	Дуже висока	Висока	Швидко зростає
Основне призначення	Повноцінні вебзастосунки з фронтендом і бекендом	Легкі API чи невеликі вебсервіси	Сучасні REST API / мікросервіси

З огляду на специфіку проєкту, було обрано саме Django. На відміну від Flask чи FastAPI, цей фреймворк краще підходить для побудови повноцінного вебдодатку із фронтенд-частиною, де важлива наявність стандартних механізмів: від адмін-панелі та аутентифікації до Object Relational Mapper (ORM) і готової структури. Django дозволяє швидко створювати не просто набір кінцевих точок, а цілі вебінтерфейси, що є ключовим у даному проєкті.

Для реалізації Representational State Transfer Application Programming Interface (REST API) використовується окрема бібліотека Django REST Framework (DRF). Вона працює як розширення Django й значно спрощує створення ендпойнтів для взаємодії з моделями машинного навчання. DRF дає можливість підключати фронтенд та інші клієнти до системи без зайвого писання обробників HTTP-запитів вручну.

3.3 Вибір мови програмування та фреймворку для створення фронтенд-частини додатку

Фронтенд сучасних вебзастосунків традиційно реалізується мовами JavaScript або TypeScript. JavaScript є стандартом для браузерних середовищ і фактично єдиною універсально підтримуваною мовою для фронтенд-розробки. Його широке поширення зумовлює наявність величезної кількості бібліотек і фреймворків.

Разом з тим, у великих і середніх проєктах часто перевага надається TypeScript – надбудові над JavaScript, що додає систему статичної типізації. Це дозволяє зменшити кількість помилок у коді, зробити його зрозумілішим для читання та полегшити підтримку в довгостроковій перспективі. Таким чином, TypeScript виступає оптимальним вибором для фронтенд-частини системи, де важлива стабільність і прогнозованість поведінки.

У табл. 3.3 наведено порівняння найбільш поширених фронтенд-фреймворків.

Таблиця 3.3 – Порівняння фронтенд-фреймворків

Критерій порівняння	React	Angular	Svelte 5
Рівень складності	Середній	Високий (багато концепцій)	Низький (простий синтаксис)
Розмір коду	Середній (JSX, багато boilerplate)	Високий (структуровані шаблони)	Дуже низький (мінімум зайвого коду)
Продуктивність виконання	Висока	Середня	Дуже висока (компілюється у JS)
Популярність та спільнота	Дуже висока	Висока	Середня (швидко зростає)
Основне призначення	Універсальні SPA, велика екосистема	Великі корпоративні застосунки	Легкі та середні вебзастосунки

На відміну від React чи Angular, які іноді обтяжені значним обсягом шаблонного коду, Svelte 5 пропонує мінімалістичний та інтуїтивно зрозумілий підхід. Його ключова відмінність полягає у компіляції під час збірки: замість завантаження складного фреймворку у браузер, користувач отримує оптимізований JavaScript-код. Це забезпечує кращу продуктивність і зменшує навантаження на клієнтську частину.

З огляду на потреби проєкту було обрано саме Svelte 5 як основний фронтенд-фреймворк. Його синтаксис легко читається навіть новачками, код лишається компактним та прозорим, а розробка не перевантажується зайвими інструментами. Таким чином, Svelte 5 поєднує простоту, продуктивність і сучасний підхід до створення інтерфейсів, що робить його найбільш придатним рішенням у даному випадку.

3.4 Вибір середовища розробки

Visual Studio Code, що розроблений корпорацією Microsoft, являє собою багатофункціональний редактор вихідного коду з кросплатформною підтримкою для операційних систем Windows, Linux та macOS [33]. Серед його основних функцій – інтегрована підтримка налагодження, підсвічування синтаксису, інтелектуальне завершення коду (IntelliSense), можливості рефакторингу та глибока інтеграція з системою контролю версій Git.

Велика гнучкість редактора дозволяє користувачам адаптувати його під свої потреби через зміну тем, налаштування комбінацій клавіш і, що особливо важливо, встановлення великої кількості розширень, які суттєво розширюють базову функціональність. Завдяки своїй універсальності та функціональності, VSCode визнаний найпопулярнішим інструментом серед розробників: в опитуванні Stack Overflow 2024 року 73,6 % з 58 121 опитаних повідомили про його використання [33].

В контексті цієї роботи, особливою перевагою VSCode є його безшовна інтеграція з Windows Subsystem for Linux (WSL). Оскільки розробка та

тестування виконувались у середовищі Linux (через WSL) для максимальної сумісності та продуктивності, а також з огляду на переваги, які надає ця підсистема, легке підключення VSCode до віддалених середовищ стало вирішальним фактором. Крім того, на час розробки нові версії бібліотеки TensorFlow для машинного навчання мають обмежену підтримку тренування моделі з використанням графічного процесора на Windows, що також посилило необхідність використання середовища Linux (через WSL).

Jupyter notebook – це інструмент з відкритим вихідним кодом, який є важливою частиною науки про дані. Він створює інтерактивне середовище, яке значно спрощує дослідження даних, забезпечує відтворюваність експериментів та сприяє обміну знаннями. Його унікальність полягає у здатності об'єднувати в одному документі виконавчий код (у так званих "клітинках"), візуалізації, описовий текст, математичні формули та мультимедійні елементи. Це дозволяє створювати повні, легкодоступні звіти та "живі" посібники. Клітинкова архітектура дозволяє розробникам писати, виконувати фрагменти коду та документувати свої міркування в режимі реального часу, що є ідеальним для швидкого експериментування з різними алгоритмами машинного навчання та налаштуваннями гіперпараметрів [34].

У табл. 3.4 наведено порівняння ключових Integrated development kit (IDE), які часто використовуються для розробки на мові Python.

Таблиця 3.4 – Порівняння середовищ розробки

Критерій порівняння	VSCode	PyCharm	JupyterNotebook
Підтримка розробки на різних мовах програмування	Висока	Середня	Низька
Легкість налаштування	Висока	Середня	Висока
Підтримка та спільнота	Висока	Висока	Висока
Кросплатформність	Висока	Висока	Висока
Інтеграція з системами контролю версій	Висока	Висока	Низька
Підтримка розширень та плагінів	Висока	Висока	Низька

Виходячи з проведеного аналізу та вимог проєкту, VSCode було обрано як основне середовище для написання основного коду програми, її структури та інтеграції. Водночас, на етапі експериментів з нейронною мережею та оптимізації результатів буде застосовано Jupyter Notebook. Це рішення прийняте через його ключову перевагу: можливість послідовного виконання окремих блоків коду зі збереженням стану та результатів між запусками, що незамінне для швидкого ітеративного тестування моделей машинного навчання.

3.5 Вибір бази даних

У табл. 3.5 наведено порівняння SQLLite, MySQL та PostgreSQL.

Таблиця 3.5 – Порівняння баз даних

Критерій порівняння	SQLite	MySQL	PostgreSQL
Легкість налаштування	Дуже висока	Середня	Середня
Продуктивність	Низька для великих обсягів, добра для невеликих	Висока	Висока, особливо для складних запитів
Масштабованість	Низька (локальна, без паралельних користувачів)	Висока	Дуже висока
Гнучкість у зміні схеми	Середня	Середня	Висока
Вимоги до ресурсів	Дуже низькі	Середні	Середні
Підтримка ORM	Повна підтримка Django ORM	Повна підтримка Django ORM	Повна підтримка Django ORM

Для розробки програмного продукту було обрано SQLite як основну базу даних. Головними перевагами цього рішення є простота використання та відсутність потреби у додатковому серверному програмному забезпеченні, що дозволяє швидко розпочати процес розробки та тестування без зайвих витрат

на конфігурацію. SQLite інтегрована в стандартну бібліотеку Python, тому налаштування зводиться до мінімуму.

Використання Django ORM забезпечує незалежність від конкретної реалізації бази даних. Це означає, що при зростанні вимог до продуктивності або обсягів даних систему можна легко переналаштувати для роботи з більш потужною системою керування базою даних (СКБД), такими як PostgreSQL або MySQL, змінивши конфігураційні параметри у файлі settings.py. Таким чином, зберігається гнучкість і масштабованість проєкту, а також можливість поступового переходу до складнішої інфраструктури у майбутньому.

3.6 Проєктування структури бази даних

Для зберігання даних даного проєкту було спроєктовано базою даних (БД), що складається з наступних таблиць:

Users містить основну інформацію про всіх користувачів системи, успадковану від AbstractUser Django, з додаванням поля ролі (табл. 3.6).

Таблиця 3.6 – Сутність «Users»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
username	STRING	Унікальне ім'я користувача
password	STRING	Хешований пароль
email	STRING	Унікальна електронна пошта
first_name	STRING	Ім'я користувача
last_name	STRING	Прізвище користувача
is_superuser	BOOLEAN	Прапорець статусу суперкористувача
is_staff	BOOLEAN	Прапорець статусу персоналу
is_active	BOOLEAN	Прапорець, чи активний акаунт
date_joined	DATETIME	Дата та час реєстрації
last_login	DATETIME	Дата та час останнього входу
role	ENUM	Роль користувача ('admin', 'patient', 'doctor')

DoctorProfiles містить специфічну інформацію, що стосується лише лікарів (табл. 3.7).

Таблиця 3.7 – Сутність «DoctorProfiles»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
user_id	INT	Зовнішній ключ (1-to-1) до табл. Users
license_number	STRING	Номер ліцензії лікаря
specialization	STRING	Спеціалізація лікаря

PatientProfiles містить специфічну інформацію, що стосується лише пацієнтів (табл. 3.8).

Таблиця 3.8 – Сутність «PatientProfiles»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
user_id	INT	Зовнішній ключ (OneToOne) до таблиці Users
dob	DATETIME	Дата народження пацієнта
sex	STRING	Стать пацієнта
medical_record_number	STRING	Номер медичної картки

MedicalCases – центральна таблиця, що описує медичні випадки та пов'язує пацієнтів і лікарів (табл. 3.9).

Таблиця 3.9 – Сутність «MedicalCases»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
patient_id	INT	Зовнішній ключ до таблиці Users (пацієнт)
primary_doctor_id	INT	Зовнішній ключ до таблиці Users (лікар)
title	STRING	Назва/заголовок медичного випадку
description	STRING	Детальний опис медичного випадку
status	ENUM	Статус випадку ('open', 'closed', 'archived')
diagnosis_summary	STRING	Підсумковий діагноз або резюме випадку
created_at	DATETIME	Дата та час створення запису
updated_at	DATETIME	Дата та час останнього оновлення запису

ChestScans містить інформацію про завантажені знімки грудної клітини, пов'язані з медичними випадками (табл. 3.10).

Таблиця 3.10 – Сутність «ChestScans»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
case_id	INT	Зовнішній ключ до таблиці MedicalCases
image_path	STRING	Шлях до файлу зображення знімка
final_label	STRING	Остаточний діагноз, встановлений лікарем
final_label_set_at	DATETIME	Дата та час встановлення остаточного діагнозу
uploaded_at	DATETIME	Дата та час завантаження знімка

ModelVersions забезпечує управління версіями моделей штучних нейронних мереж (ШНМ), які використовуються для аналізу (табл. 3.11).

Таблиця 3.11 – Сутність «ModelVersions»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
uploaded_by_admin_id	INT	Зовнішній ключ до таблиці Users (адміністратор)
model_name	STRING	Назва моделі (наприклад, 'VGG16')
storage_uri	STRING	Шлях до файла чекпоінту моделі
description	STRING	Опис версії моделі та її особливостей
performance_metrics	JSON	Показники продуктивності моделі
is_active	BOOLEAN	Прапорець, чи активна модель для прогнозів
created_at	DATETIME	Дата та час завантаження моделі в систему

AI_Analyses зберігає результати прогнозу (висновки) окремих моделей ШНМ для конкретних знімків (табл. 3.12).

Таблиця 3.12 – Сутність «AI_Analyses»

Поле	Тип даних	Опис
1	2	3
id	INT	Первинний ключ, автоінкремент

Продовження таблиці 3.12

Поле	Тип даних	Опис
1	2	3
scan_id	INT	Зовнішній ключ до таблиці ChestScans
model_version_id	INT	Зовнішній ключ до табл. ModelVersions
prediction_label	ENUM	Прогноз моделі ('pneumonia', 'normal')
confidence_score	FLOAT	Показник впевненості прогнозу
heatmap_path	STRING	Шлях до згенерованої теплової карти
heatmap_type	ENUM	Тип використаної теплової карти
generated_at	DATETIME	Дата та час генерації прогнозу

EnsembleResult зберігає фінальний результат ансамблевого методу класифікації для знімка (табл. 3.13).

Таблиця 3.13 – Сутність «EnsembleResult»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
scan_id	INT	Зовнішній ключ до таблиці ChestScans
method	ENUM	Метод ансамблю ('weighted', 'average', 'majority_vote')
combined_prediction_label	STRING	Комбінований (підсумковий) прогноз ансамблю
combined_confidence_sco-re	FLOAT	Комбінований показник впевненості
created_at	DATETIME	Дата та час створення результату ансамблю

DoctorAnnotations містить коментарі/нотатки/висновки, надані лікарями щодо знімків (табл. 3.14).

Таблиця 3.14 – Сутність «DoctorAnnotations»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
scan_id	INT	Зовнішній ключ до таблиці ChestScans
doctor_id	INT	Зовнішній ключ до таблиці Users (лікар)
notes	STRING	Текстові нотатки, коментарі лікаря
created_at	DATETIME	Дата та час створення нотатки

AuditLogs призначена для ведення журналів дій користувачів, помилок системи та інших важливих подій (табл. 3.15).

Таблиця 3.15 – Сутність «AuditLogs»

Поле	Тип даних	Опис
id	INT	Первинний ключ, автоінкремент
user_id	INT	Зовнішній ключ до таблиці Users
action	STRING	Опис виконаної дії (напр., CREATED_USER, 'SCAN_UPLOAD')
details	JSON	Додаткові деталі дії у форматі JSON
created_at	DATETIME	Дата та час запису в журнал

Зв’язки між таблицями:

– Users має зв'язок один до одного з таблицею DoctorProfiles через поле user_id;

– Users має зв'язок один до одного з таблицею PatientProfiles через поле user_id;

– Users (в ролі пацієнта) має зв'язок один до багатьох з таблицею MedicalCases через поле patient_id;

– Users (в ролі лікаря) має зв'язок один до багатьох з таблицею MedicalCases через поле primary_doctor_id;

– Users (в ролі лікаря) має зв'язок один до багатьох з таблицею DoctorAnnotations через поле doctor_id;

– Users (в ролі адміністратора) має зв'язок один до багатьох з таблицею ModelVersions через поле uploaded_by_admin_id;

– Users має зв'язок один до багатьох з таблицею AuditLogs через поле user_id;

– MedicalCases має зв'язок один до багатьох з таблицею ChestScans через поле case_id;

– ChestScans має зв'язок один до багатьох з таблицею AI_Analyses через поле scan_id;

– ChestScans має зв'язок один до багатьох з таблицею EnsembleResult через поле scan_id;

– ChestScans має зв'язок один до багатьох з таблицею DoctorAnnotations через поле scan_id;

– ModelVersions має зв'язок один до багатьох з таблицею AI_Analyses через поле model_version_id.

Схему бази даних та зв’язки між сутностями наведено на рис. 3.1.

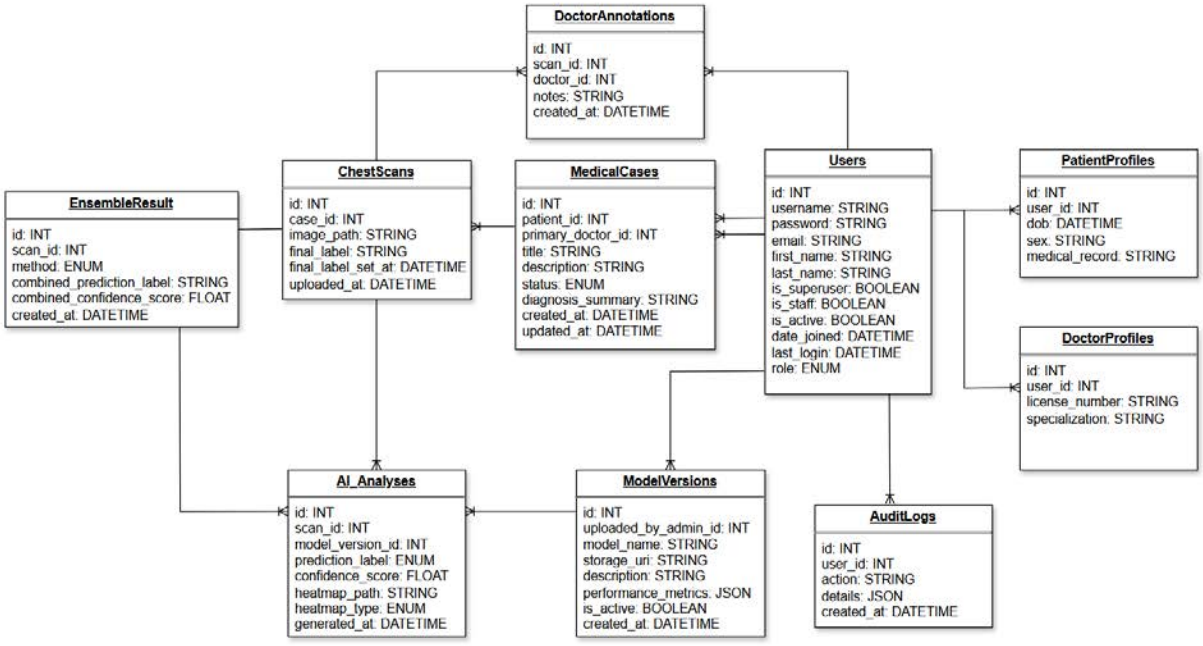

Рисунок 3.1 – Схема бази даних та зв’язки між сутностями

3.7 Архітектура програмного забезпечення

Для розробки програмного продукту у роботі використано архітектурний шаблон MVC у роз'єднаному вигляді. Такий підхід передбачає чіткий поділ системи на три незалежні компоненти: бекенд (Модель і Контролер), реалізований на Django REST Framework, та фронтенд (Подання), побудований на Svelte 5. Взаємодія між ними відбувається через REST API. Це забезпечує високу модульність, гнучкість у розробці та легкість масштабування.

Компоненти архітектури:

а) Model (Модель):

1) опис: Модель є рівнем даних та бізнес-логіки. Вона відповідає за структуру даних, їх зберігання, валідацію та операції маніпуляції;

2) реалізація: У даному проєкті цей компонент реалізовано за допомогою Django ORM, яка описує сутності бази даних (пацієнти, медичні випадки, знімки тощо). До цього ж рівня належать моделі машинного навчання, розроблені за допомогою TensorFlow, які виконують класифікацію рентгенівських знімків грудної клітини. Всі дані, включно зі шляхами до знімків та результатами прогнозів ШНМ, зберігаються у базі даних, керованій Django;

б) View (Подання):

1) опис: Подання відповідає за візуалізацію інформації та взаємодію з користувачем. У роз'єднаній архітектурі це повністю клієнтський застосунок, що отримує дані від Контролера;

2) реалізація: Подання являє собою односторінковий застосунок або Single Page Application (SPA), розроблений на фреймворку Svelte 5 зі стилізацією за допомогою Tailwind CSS. Він комунікує з бекендом через HTTP-запити до REST API, отримуючи дані у форматі JSON. Вебінтерфейс дозволяє лікарям завантажувати знімки, переглядати результати аналізу, вивчати прогнози ансамблю моделей та фіксувати остаточний діагноз;

в) Controller (Контролер):

1) опис: Контролер виступає як посередник між Поданням (клієнтом) та Моделлю (базою даних і бізнес-логікою). Він обробляє вхідні запити, виконує необхідні операції та повертає дані у стандартизованому форматі;

2) реалізація: Роль Контролера виконує Django REST Framework (DRF). Він надає набір API-ендпоінтів (кінцевих точок).

ViewSets (набори представлень) обробляють запити (GET, POST, PUT тощо), Serializers (серіалізори) валідують дані та перетворюють моделі Django у формат JSON (і навпаки). Таким чином, DRF керує всією логікою запитів-відповідей, забезпечуючи зв'язок між інтерфейсом Svelte та моделями даних.

На рис. 3.2 зображено архітектуру застосунку на основі шаблону проєктування MVC.

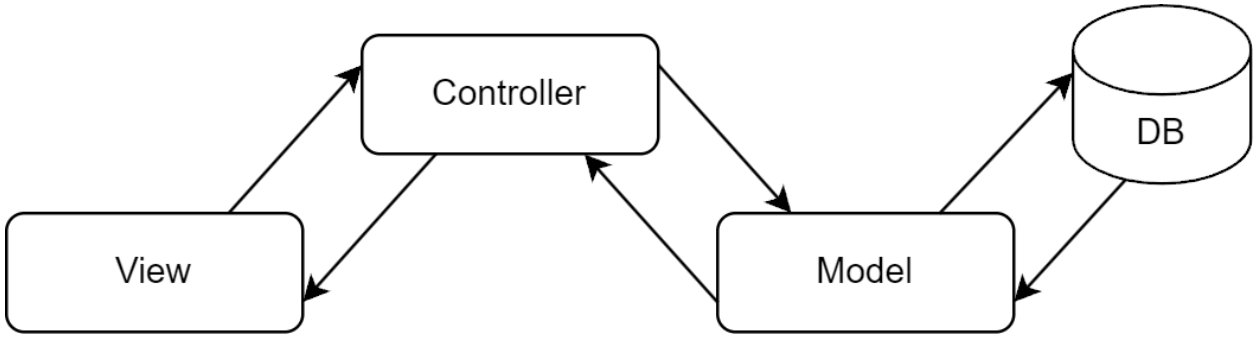

Рисунок 3.2 – Архітектура застосунку

3.8 Визначення функціональних вимог до застосунку

Для програмного забезпечення було визначено наступні функціональні вимоги.

Вимоги пацієнта:

– реєстрація в системі;
– авторизація в системі;
– перегляд списку медичних випадків та їх статусів;
– перегляд власних рентгенівських знімків у межах медичного випадку;
– перегляд остаточних діагнозів та висновків, наданих лікарем;
– вихід з особистого кабінету.

Вимоги лікаря:

– реєстрація в системі;
– авторизація в системі;

– перегляд списку медичних випадків;

– пошук медичних випадків за пацієнтом;

– створення нового медичного випадку для пацієнта;

– редагування інформації про медичний випадок;

– завантаження рентгенівських знімків до медичного випадку;

– перегляд завантажених знімків та результатів їх автоматичного аналізу (прогнозів ШНМ, результатів ансамблю, теплових карт);

– формування власних анотацій до окремих знімків;

– встановлення та редагування остаточного діагнозу для знімка;

– вихід з особистого кабінету.

Вимоги адміністратора:

– авторизація в системі;

– перегляд статистики всієї системи (кількість користувачів, моделей, помилок за останні 24 години);

– перегляд списку всіх користувачів системи;

– редагування даних профілів користувачів;

– ініціювати перенавчання моделей ШНМ;

– перегляд списку всіх збережених чекпоінтів моделей ШНМ;

– управління чекпоінтами моделей ШНМ (активація/деактивація для використання в аналізі, видалення);

– перегляд журналу аудиту;

– перегляд статистики порівняннь передбачень моделі з рішеннями лікарів;

– вихід з панелі адміністратора.

Діаграма прецедентів для пацієнта наведена на рис. 3.3.

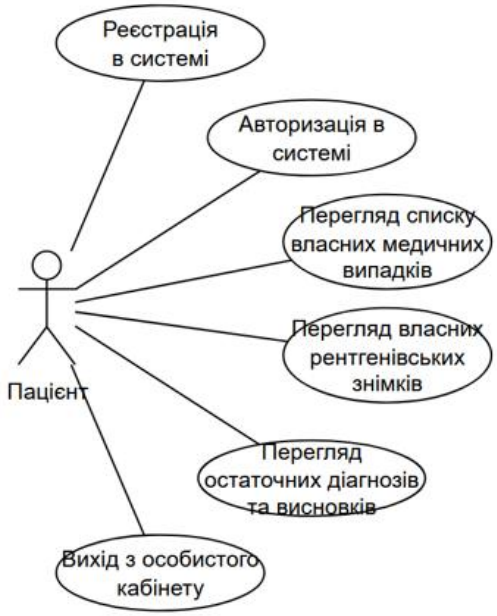

Рисунок 3.3 – Діаграма прецедентів для пацієнта

Діаграма прецедентів для лікаря наведена на рис. 3.4.

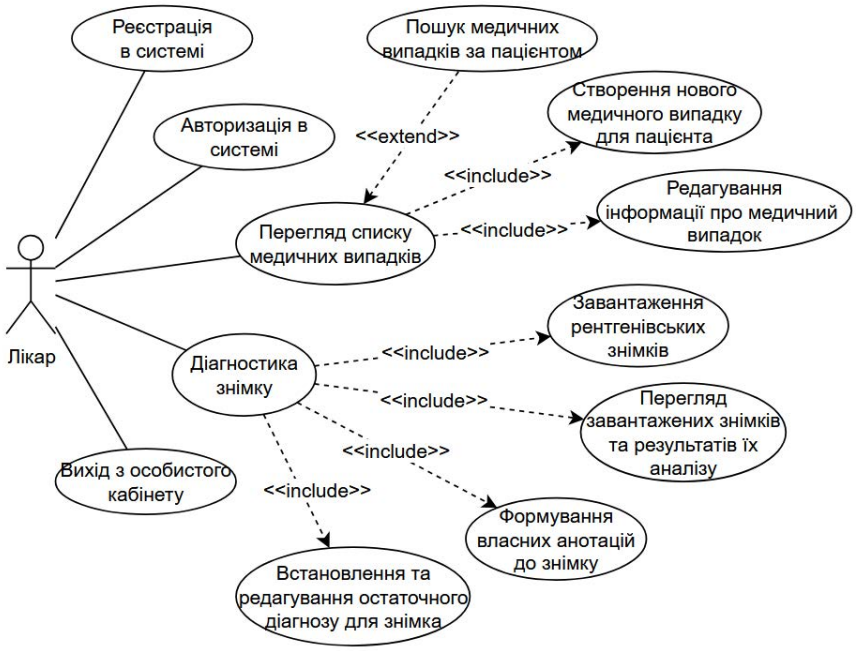

Рисунок 3.4 – Діаграма прецедентів для лікаря

Діаграма прецедентів для адміністратора наведена на рис. 3.5.

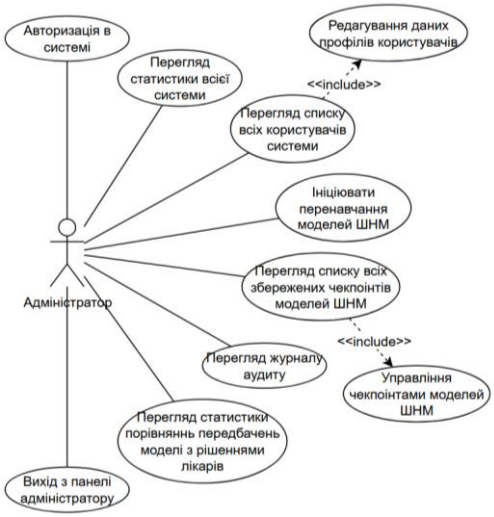

Рисунок 3.5 – Діаграма прецедентів для адміністратора

3.9 Висновки до третього розділу

У даному розділі було розглянуто обрані засоби, інструменти й методи для розробки програмного забезпечення, а саме:

– було обрано мову програмування Python та бібліотеки Django, DRF для реалізації backend-частини додатку;

– було обрано вебфрейморк Svelte5 для реалізації frontend-частини додатку;

– було обрано бібліотеки Tensorflow, Keras для реалізації нейромережних методів;

– було обрано середовище розробки VSCode для основної розробки програмного застосунку та JupyterNotebook для розробки нейромережних методів й експериментів для підвищення результатів;

– для роботи з базою даних було визначено СКБД SQLLite та Django ORM для збереження абстракції й полегшення подальшої міграції до іншої СКБД при необхідності.

Також було розглянуто структуру бази даних та визначено функціональні вимоги до застосунку.

4 РОЗРОБКА ПРОГРАМИ

4.1 Алгоритм функціонування застосунку

На рис. 4.1 наведено узагальнену схему функціонування вебзастосунку, розроблену на основі функціональних вимог, визначених у попередньому розділі.

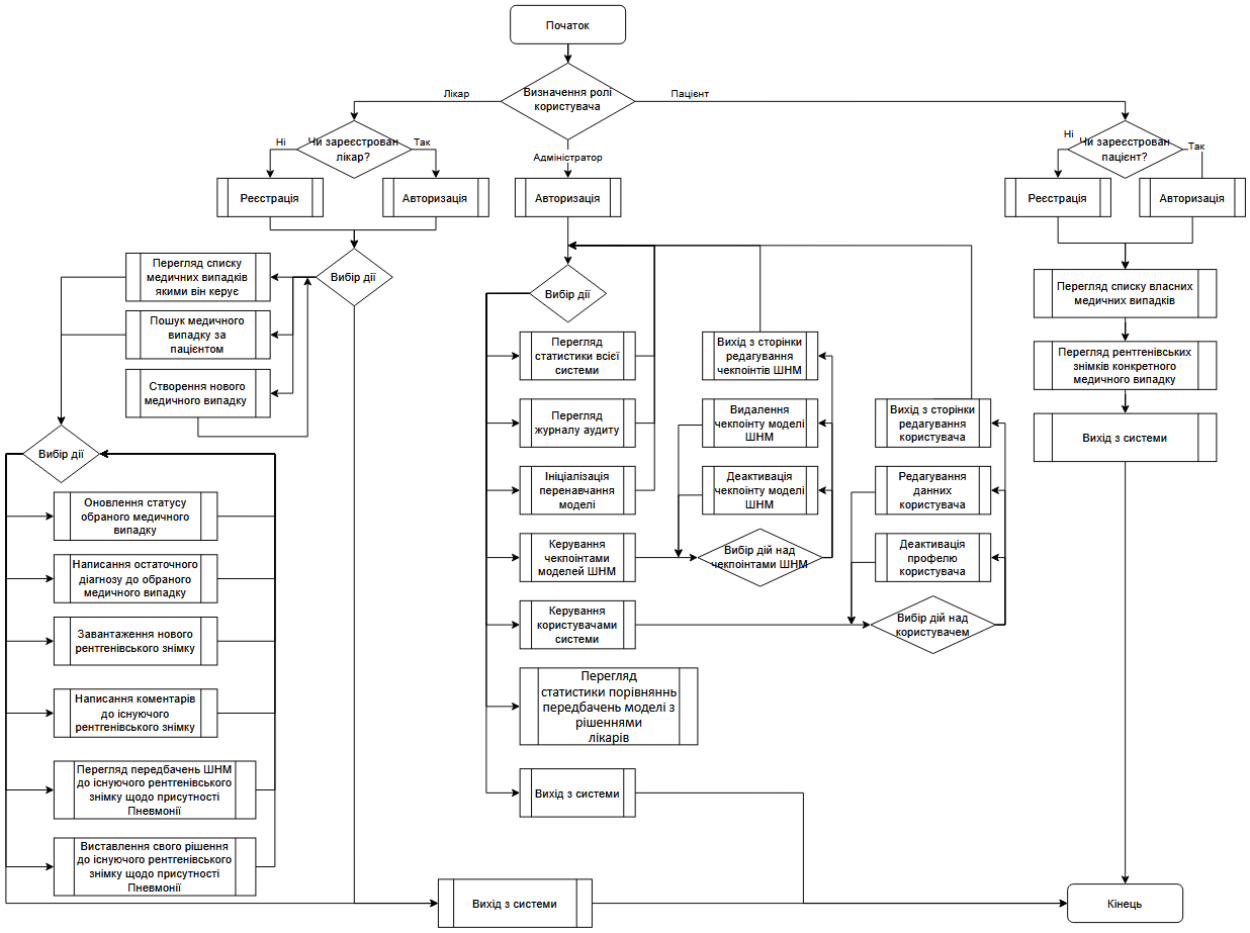

Рисунок 4.1 – Схему функціонування застосунку

Після запуску системи користувач, який уже має обліковий запис, переходить на сторінку авторизації та вводить свої облікові дані. Якщо користувач (пацієнт або лікар) ще не зареєстрований, йому необхідно пройти процедуру реєстрації для створення нового облікового запису.

Після успішної авторизації система визначає роль користувача – пацієнт, лікар або адміністратор – і на основі цього відкриває відповідний інтерфейс.

Пацієнт після входу потрапляє на сторінку власних медичних випадків. Він може переглядати список випадків, відкривати окремі з них і переглядати пов’язані рентгенівські знімки.

Лікар переходить на сторінку призначених йому медичних випадків, де може переглядати їх, шукати за ім’ям та прізвищем пацієнта, або створювати новий. Після обирання конкретного медичного випадку йому доступний такий вибір дій:

– лікар може змінювати статус випадку на “Open”, “Closed” чи “Archived”;

– додавати остаточний діагноз із описом перебігу захворювання, методів лікування та результатів;

– лікар може завантажувати нові рентгенівські знімки. Після завантаження знімка система автоматично виконує класифікацію за допомогою всіх активованих нейронних мереж, відображає окремі результати кожної з них та ансамблевий підсумковий результат, який також зберігається у базі даних;

– додавати коментарі до існуючих знімків;

– залишати власний висновок щодо знімка після перегляду результатів ШНМ.

Адміністратору доступний такий вибір дій:

– перегляд статистики всієї системи;

– перегляд статистики порівняннь передбачень моделі з рішеннями лікарів;

– перегляд журналу аудиту;

– керування користувачами системи (редагування імені, прізвища, електронної пошти, деактивація акаунта);

– ініціалізація перенавчання обранної нейронної мережі;

– керування чекпоінтами моделей ШНМ з можливістю їх деактивації або видалення.

Усі сторінки вебзастосунку логічно взаємопов’язані, що забезпечує зручну навігацію між основними розділами. Робота користувача з системою завершується після виходу з облікового запису.

4.2 Структура програмного застосунку

Коренева директорія проєкту наведена на рис. 4.2.

```
masters-thesis/
├── backend/
│   ├── classifier/             # TensorFlow CNN & data pipeline
│   │   ├── classifier.py       # Classifier definition, training loop
│   │   ├── progress_state.py   # Training progress tracking
│   │   ├── datasets/           # Training / validation datasets
│   │   └── outputs/            # Training metrics, charts, confusion matrix, saved checkpoints, gradcam heatmaps
│   ├── api/                    # App for main Django REST API
│   │   ├── management/commands/seed_db.py     # Custom Django management command (seeding)
│   │   ├── models.py           # Core database models
│   │   ├── serializers.py      # DRF serializers
│   │   ├── permissions.py      # Role-based access control
│   │   ├── views.py            # API endpoints
│   │   ├── audit_mixin.py      # Audit mixin for logging logic
│   │   ├── audit_utils.py      # Audit utils
│   │   └── urls.py             # URL routing
│   ├── users/                  # App for Authentication and roles  (Admin/Doctor/Patient)
│   │   ├── models.py           # User-related database models
│   │   ├── serializers.py      # DRF serializers
│   │   ├── permissions.py      # Role-based access control
│   │   ├── views.py            # API endpoints
│   │   └── urls.py             # URL routing
│   ├── pneumonia_diagnosis/    # Django project settings & entry points
│   │   ├── settings.py         # Project settings
│   │   └── urls.py             # URL routing
│   ├── frontend/               # App for serving frontend/static page requests via Django
│   ├── media/                  # Uploaded scans, heatmaps
│   ├── static/                 # Compiled frontend files (for production)
│   ├── db.sqlite3              # Development database
│   └── manage.py               # Django management CLI
│
├── frontend/
│   ├── src/
│   │   ├── lib/
│   │   │   ├── components/     # Modular Svelte components
│   │   │   │   ├── AdminDashboard/       # Admin UI components
│   │   │   │   ├── DoctorDashboard/      # Doctor UI components
│   │   │   │   ├── PatientDashboard/     # Patient UI components
│   │   │   │   └── Shared UI components  # Reusable UI blocks
│   │   │   ├── api.ts              # REST API client
│   │   │   ├── auth.ts             # Authentication helpers
│   │   │   └── types.ts            # Shared TypeScript types
│   │   ├── routes/             # SvelteKit routes (login, register, dashboard, etc.)
│   │   │   ├── +page.svelte        # Page component
│   │   │   ├── +layout.svelte      # Layout component
│   │   │   ├── load.ts             # Load function for route data
│   │   │   └── +layout.ts          # Layout-level load logic
│   │   └── app.html / app.css   # Root HTML template & global styles
│   ├── svelte.config.js         # SvelteKit configuration
│   ├── vite.config.ts           # Vite bundler configuration
│   └── package.json             # Frontend dependencies & scripts
│
├── pyproject.toml   # Python project configuration
├── .gitignore       # Git ignore rules
└── README.md        # Project overview and documentation
```

Рисунок 4.2 – Коренева директорія проєкту

Проєкт реалізовано за архітектурою з відокремленими клієнтською та серверною частинами, що знаходяться в одній кореневій директорії masters-thesis/:

– backend/: директорія, що містить серверну частину застосунку, розроблену з використанням фреймворку Django та Django Rest Framework (DRF);

– frontend/: директорія, що містить клієнтську частину застосунку, розроблену з використанням фреймворку Svelte 5;

– .gitignore: визначає навмисно невідстежувані файли, які Git має ігнорувати (наприклад, файли конфігурації середовища, кеш, медіафайли тощо);

– pyproject.toml: файли для керування залежностями Python.

4.2.1 Структура серверної частини (backend)

Директорія backend у складі проєкту реалізує API-застосунок на базі Django + DRF, що забезпечує роботу бази даних, аутентифікацію користувачів, управління моделями нейронних мереж та комунікацію з фронтендом через REST API.

а) pneumonia_diagnosis: основний пакунок проєкту Django:

1) settings.py: містить усі налаштування конфігурації Django, включаючи підключення до бази даних, CORS, параметри DRF, тощо;

2) urls.py: кореневий файл URL-маршрутизації, що підключає маршрути з окремих застосунків (users, api) та налаштовує роздачу медіа- та статичних файлів;

б) users: застосунок Django, що відповідає за логіку, пов'язану з користувачами (автентифікація, ролі):

1) models.py: визначає частину моделей даних для роботи з користувачами, наведеними у підрозділі 3.5;

2) serializers.py: містить DRF-серіалізатори для перетворення моделей користувачів у JSON та навпаки;

3) views.py: реалізує ViewSets (набори представлень) для обробки API-запитів, пов’язаних з користувачами;

4) urls.py: визначає маршрутизацію API для цього застосунку;

5) permissions.py: містить кастомні класи дозволів (permissions) для обмеження доступу до певних ендпоінтів залежно від ролі користувача (наприклад, IsDoctor, IsAdmin);

в) api: основний застосунок Django, що реалізує бізнес-логіку програмного забезпечення:

1) models.py: визначає частину моделей даних для роботи з медичними випадками, наведеними у підрозділі 3.5;

2) serializers.py: містить серіалізатори для моделей застосунку;

3) views.py: реалізує ViewSets (набори представлень) для обробки API-запитів, пов'язаних з медичними випадками, знімками, анотаціями та керуванням моделями;

4) urls.py: визначає маршрутизацію API для цього застосунку;

5) audit_mixins.py, audit_utils.py: допоміжні утиліти для логування операцій у система;

6) management/commands/seed_db.py: кастомна команда Django для автоматичного заповнення бази даних тестовими даними (seeding);

г) classifier: модуль, що інкапсулює всю логіку, пов'язану з машинним навчанням:

1) classifier.py: містить основний клас Classifier, що відповідає за завантаження моделей, попередню обробку зображень, класифікацію, тренування та генерацію Grad-CAM;

2) progress_state.py: містить логіку для збереження стану навчання ШНМ;

3) datasets: зберігає навчальні та тестові рентгенівські знімки, структуровані за класами (NORMAL, PNEUMONIA);

4) outputs: директорія для збереження результатів роботи ML-моделей. Вона містить: директорію checkpoints, що зберігає файли ваг навчених моделей (.hdf5), директорію gradcam, що зберігає згенеровані теплові карти Grad-CAM, директорію results, що зберігає графіки (матриці помилок, історія навчання);

д) frontend: застосунок Django, призначений виключно для обслуговування зібраного Svelte-застосунку. Його views.py містить одне "catch-all" представлення, яке рендерить index.html фронтенду, дозволяючи SvelteKit обробляти всю клієнтську маршрутизацію;

е) media: директорія, де Django зберігає всі файли, завантажені користувачами: scans для рентгенівських знімків; heatmaps для збережених результатів Grad-CAM;

ж) static: директорія, куди збираються всі статичні файли. Вона містить зібрані (зкомпільовані) файли Svelte-застосунку, які обслуговуються Django;

з) manage.py: утиліта командного рядка Django для виконання завдань адміністрування (запуск сервера, міграції, створення суперкористувача, запуск кастомної команди seed_d).

4.2.2 Структура клієнтської частини (frontend)

Клієнтська частина є односторінковим застосунком, або Single page application (SPA), що забезпечує користувацький інтерфейс:

а) src: коренева директорія вихідного коду Svelte-застосунку;

б) src/routes: містить структуру сторінок застосунку, використовуючи файлову маршрутизацію SvelteKit:

1) +page.svelte: файл, що визначає компонент сторінки (наприклад, головна сторінка, сторінка логіну);

2) +layout.svelte: визначає загальний макет (layout), який обгортає всі послідуючі вкладені сторінки (наприклад, містить Header та Footer);

3) +layout.ts / load.ts: файли для реалізації javascript/typescript логіки що можна імпортувати у компоненті, наприклад для завантаження даних, необхідних для сторінки, перед їх рендерингом;

в) src/lib: директорія для допоміжних модулів, компонентів та утиліт:

1) components: містить перевикористовувані Svelte-компоненти, згруповані за призначенням;

2) api.ts: модуль, що інкапсулює логіку HTTP-запитів до Django API з включеням обробки токенів автентифікації;

3) auth.ts: модуль для керування станом автентифікації користувача (збереження токенів, логін, логаут);

4) types.ts: відокремлене оголошення Typescript типів, які потім імпортуються в компонентах;

г) svelte.config.js та vite.config.ts: файли конфігурації для SvelteKit та збирача Vite, що визначають, як проєкт збирається та запускається в режимі розробки;

д) package.json: визначає залежності JavaScript (npm/bun) та скрипти для запуску, збірки та тестування фронтенду.

4.2.3 Структура API-ендпоінтів

Серверна частина надає наступні кінцеві точки (endpoints) REST API, згруповані за префіксом /api/:

Керування користувачами (застосунок users)

– /auth/register/ [POST]: реєстрація нового користувача (пацієнта або лікаря);

– /auth/token/ [POST]: отримання пари JWT-токенів (access та refresh) за логіном та паролем;

– /auth/token/refresh/ [POST]: оновлення access-токену за допомогою refresh-токену;

– /list/users/ [GET, POST]: отримання списку або створення користувача (для адміністраторів);

– /list/users/<id>/ [GET, PUT, PATCH, DELETE]: операції Create Read Update Delete (CRUD) над конкретним користувачем.

Основна логіка (застосунок api)

– /cases/ [GET, POST]: отримання списку медичних випадків або створення нового;

– /cases/<id>/ [GET, PUT, PATCH, DELETE]: операції CRUD над конкретним медичним випадком;

– /cases/<case_id>/scans/upload/ [POST]: завантаження нового знімка, пов'язаного з медичним випадком, й проведення передбачень, використовуючи усі активні моделі в системі;

– /scans/ [GET, POST]: отримання списку всіх знімків або створення знімка;

– /scans/<id>/ [GET, PUT, PATCH, DELETE]: операції CRUD над конкретним знімком;

– /annotations/ [GET, POST]: отримання або створення коментарів до рентгенівського знімку;

– /annotations/<id>/ [GET, PUT, PATCH, DELETE]: операції CRUD над конкретним коментарем.

Адміністрування:

– /stats/ [GET]: отримання статистики переважно про збіги діагнозів лікарів та моделей;

– /train/ [POST]: запуск процесу тренування нової моделі;

– /train/progress/ [GET]: отримання поточного прогресу тренування (опитування кожну секунду);

– /models/ [GET, POST]: отримання списку доступних CNN-моделей або реєстрація нової;

– /models/<id>/ [GET, PUT, PATCH, DELETE]: операції CRUD над конкретною моделлю;

– /models/stats/ [GET]: отримання статистики по моделях (загальна кількість, кількість активних);

– /auditlogs/ [GET]: отримання журналу аудиту дій в системі;

– /auditlogs/recent/ [GET]: отримання списку останніх 10 записів та 10 помилок;

– /auditlogs/recent_errors/ [GET]: отримання кількості помилок за останні 24 години.

4.3 Висновки до четвертого розділу

У даному розділі було визначено алгоритм функціонування розроблюваного програмного застосунку. Також було детально розглянуто структуру створеного програмного застосунку з описом створених у підсумку рішень.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис застосування програми

Розроблене програмне забезпечення є вебплатформою, призначеною для медичних закладів та фахівців у галузі радіології та пульмонології. Воно пропонує інструмент автоматизованої підтримки прийняття рішень лікарям, надаючи допомогу в класифікації рентгенівських знімків грудної клітини для виявлення ознак пневмонії.

5.2 Умови виконання програми

Для розгортання програмного забезпечення необхідно мати наявними такі компоненти:

– процессор Intel Core i7 або еквівалент;

– мінімум 8 ГБ, рекомендовано 16 ГБ або більше (особливо для тренування моделей);

– відеокарта: рекомендовано відеокарту NVIDIA з підтримкою CUDA для прискорення обчислень нейронної мережі;

– операційна система: Linux (наприклад, Ubuntu 20.04 LTS / 22.04 LTS) або WSL 2. Нативна робота в середовищі Windows не підтримується через специфічні вимоги tensorflow[and-cuda];

– інтерпретатор Python версії 3.10 або вище;

– встановлені фреймворки Django, djangorestframework та бібліотеки tensorflow[and-cuda]==2.15.1, opencv-python.

Для коректної роботи програми на стороні клієнта (лікарів та пацієнтів) потрібно мати:

– процесор від 1.0 ГГц;

– обсяг оперативної пам’яті від 4 ГБ;

– веббраузер (Google Chrome, Mozilla Firefox, OperaGX тощо).

5.3 Інструкція по експлуатації програми

При першому відвідуванні вебсайту користувач потрапляє на сторінку автентифікації (рис. 5.1). Це початковий екран, призначений для введення існуючих облікових даних (логіна та пароля) з метою отримання доступу до системи.

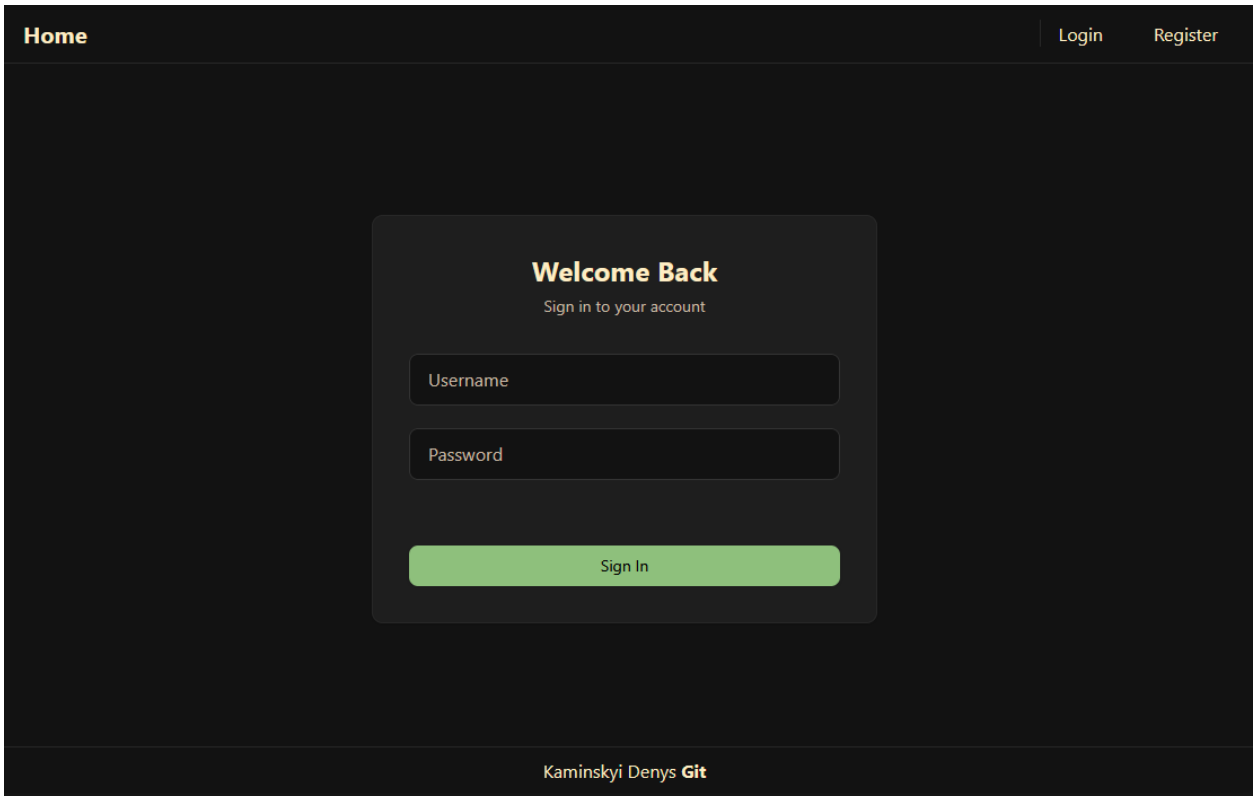

Рисунок 5.1 – Сторінка автентифікації

Якщо користувач ще не зареєстрований у системі, у верхній правій частині навігаційної панелі розміщено посилання «Register». При переході за цим посиланням, користувач перенаправляється на сторінку реєстрації (рис. 5.2). Ця форма призначена для збору необхідних даних, що дозволить створити новий унікальний обліковий запис.

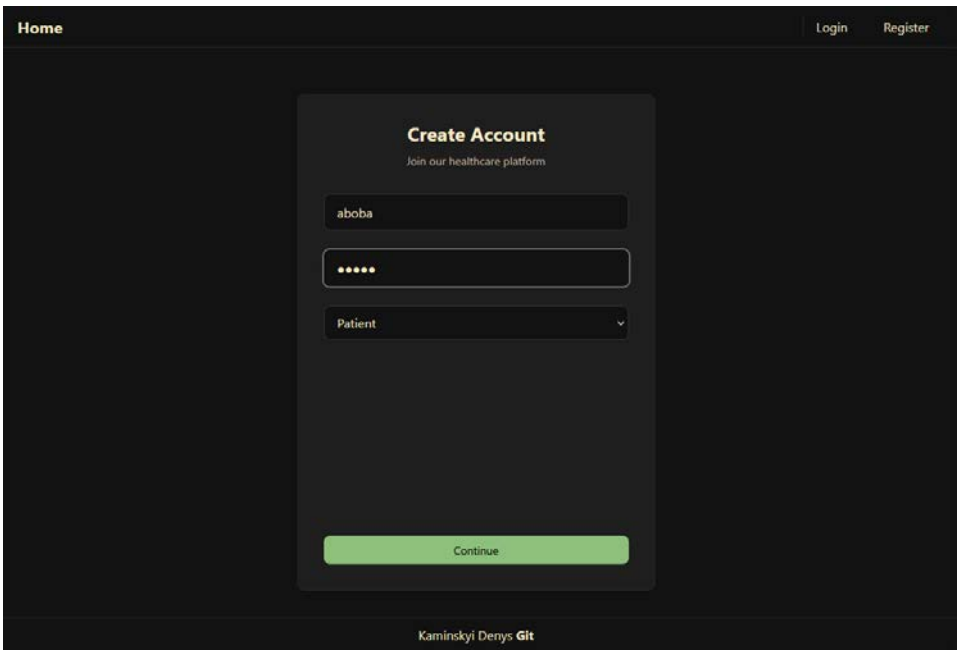

Рисунок 5.2 – Сторінка реєстрації

Система підтримує два типи користувачів з різними ролями. Тому для пацієнта при натисканні кнопки «Continue» відображаються специфічні додаткові поля, необхідні для профілю пацієнта (рис. 5.3). Аналогічно, для лікаря система надає окремі додаткові поля, що відповідають його ролі (рис. 5.4). Це забезпечує належний збір даних відповідно до типу користувача.

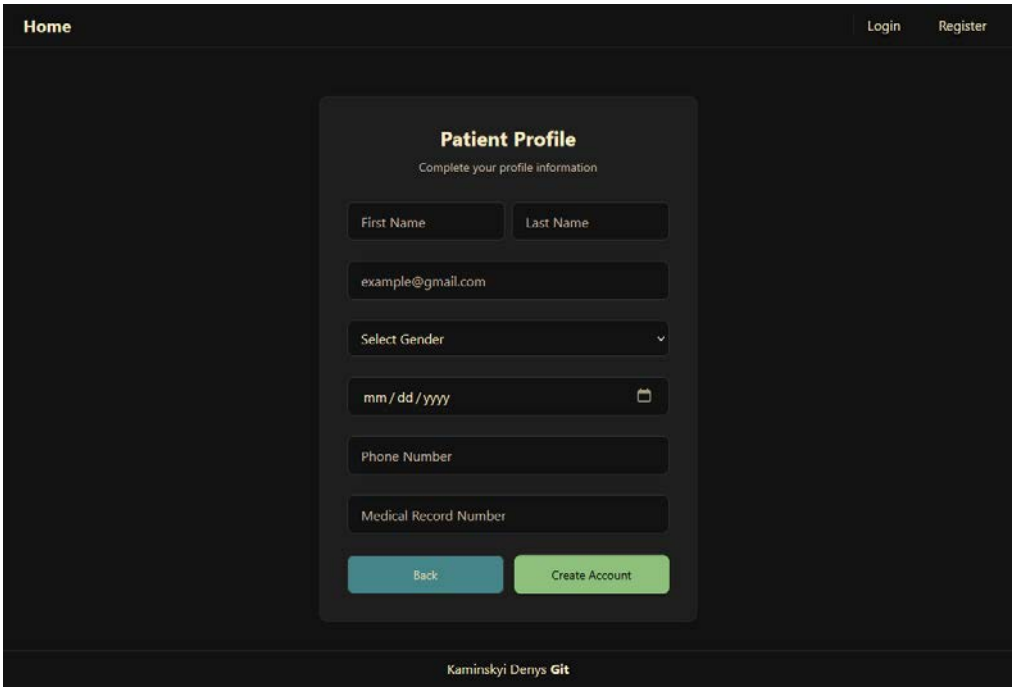

Рисунок 5.3 – Додаткові поля пацієнта на сторінці реєстрації

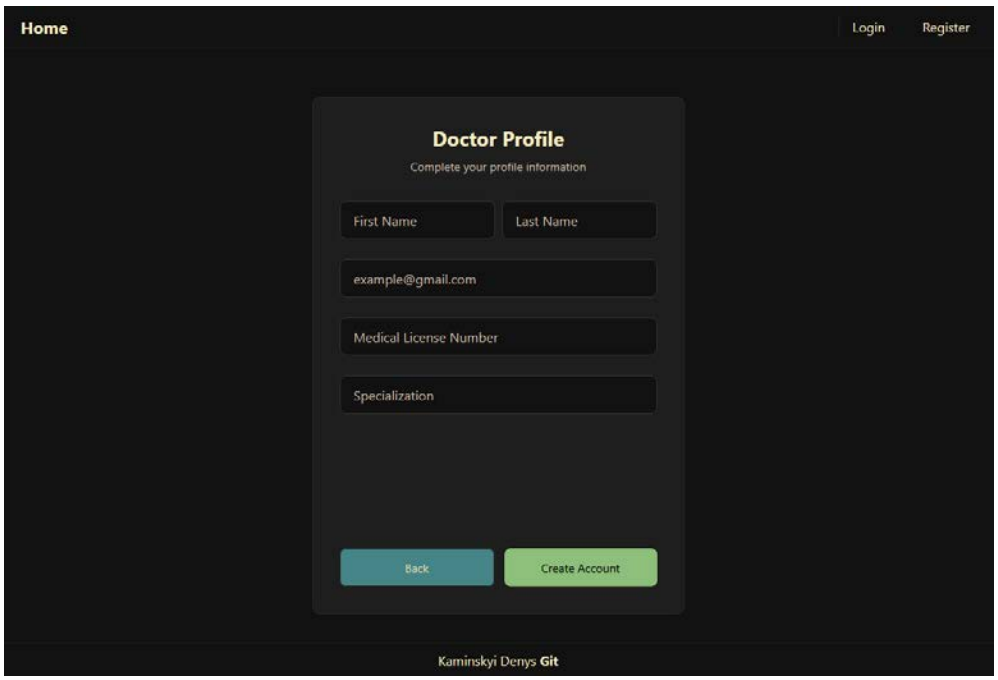

Рисунок 5.4 – Додаткові поля лікаря на сторінці реєстрації

Після успішного заповнення та надсилання реєстраційної форми, програма створює новий обліковий запис та автоматично автентифікує користувача (виконує вхід) на основі щойно введених даних.

5.3.1 Інструкція з експлуатації програми для пацієнта

Після успішної автентифікації пацієнт отримує доступ до обмеженої функціональності, що включає лише перегляд історії медичних випадків та вихід з облікового запису (рис. 5.5). На головній сторінці пацієнта відображаються усі пов'язані з ним медичні випадки, відсортовані за датою у вигляді інформативних карток. Кожна картка містить ключову інформацію: назву, опис, ім’я та прізвище пацієнта й лікаря, дату створення та поточний статус. Для отримання більш детальної інформації, пацієнт може обрати медичний випадок та натиснути на відповідну картку для переходу на його сторінку.

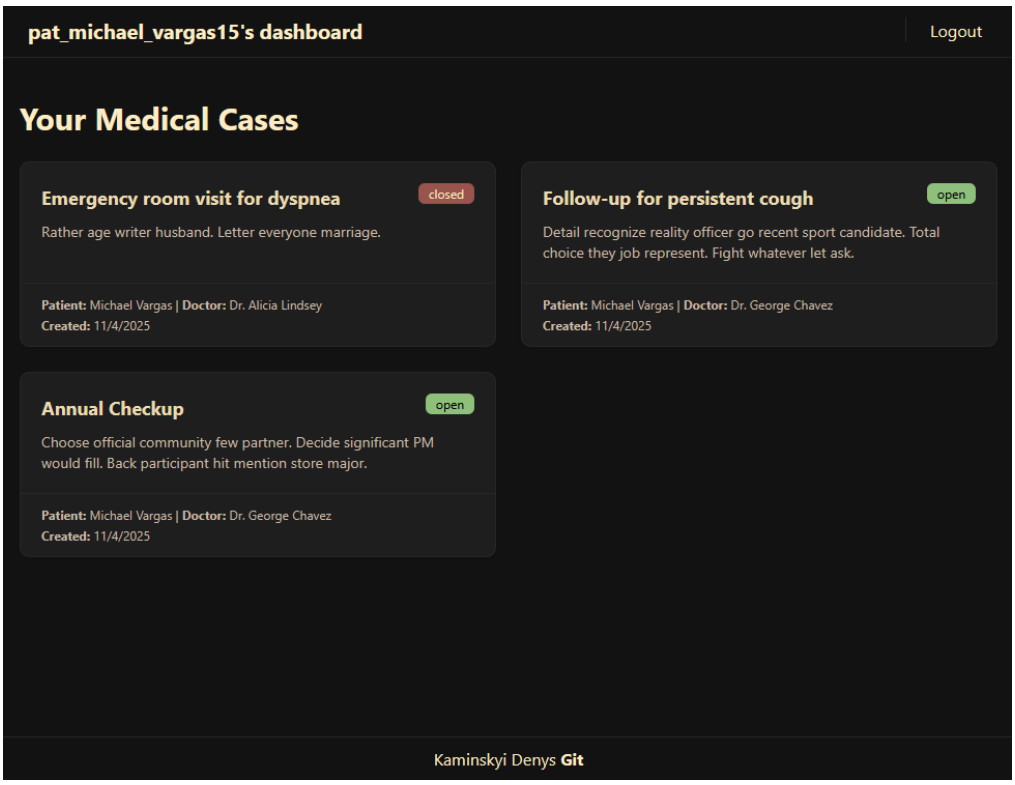

Рисунок 5.5 – Сторінка медичних випадків зі сторони пацієнта

На сторінці з детальним описом конкретного медичного випадку (рис. 5.6) пацієнт може переглянути зведену інформацію: назву, опис, ім’я та прізвище пацієнта й лікаря, статус та ідентифікатор (id) випадку. Також пацієнту відображається поле з встановленим діагнозом. Коли лікар вважає медичний випадок завершеним (наприклад, на останньому знімку пневмонії не виявлено), він заповнює підсумковий опис, що містить початковий стан, деталі процесу лікування та кінцевий результат.

Нижче розташовано блок з рентгенівськими знімками. Ліворуч знаходиться панель зі списком усіх завантажених знімків, впорядкованих за датою й часом завантаження. Кожен елемент списку функціонує як кнопка, що дозволяє обрати та відобразити деталі конкретного рентгенівського знімку в основній області праворуч. У блоці деталей знімку пацієнт бачить його назву (у вигляді дати й часу завантаження), саме зображення, а також остаточне рішення лікаря (щодо наявності чи відсутності пневмонії на даному знімку). Додатково відображаються дата й час встановлення цього рішення та коментарі лікаря, якщо вони були надані.

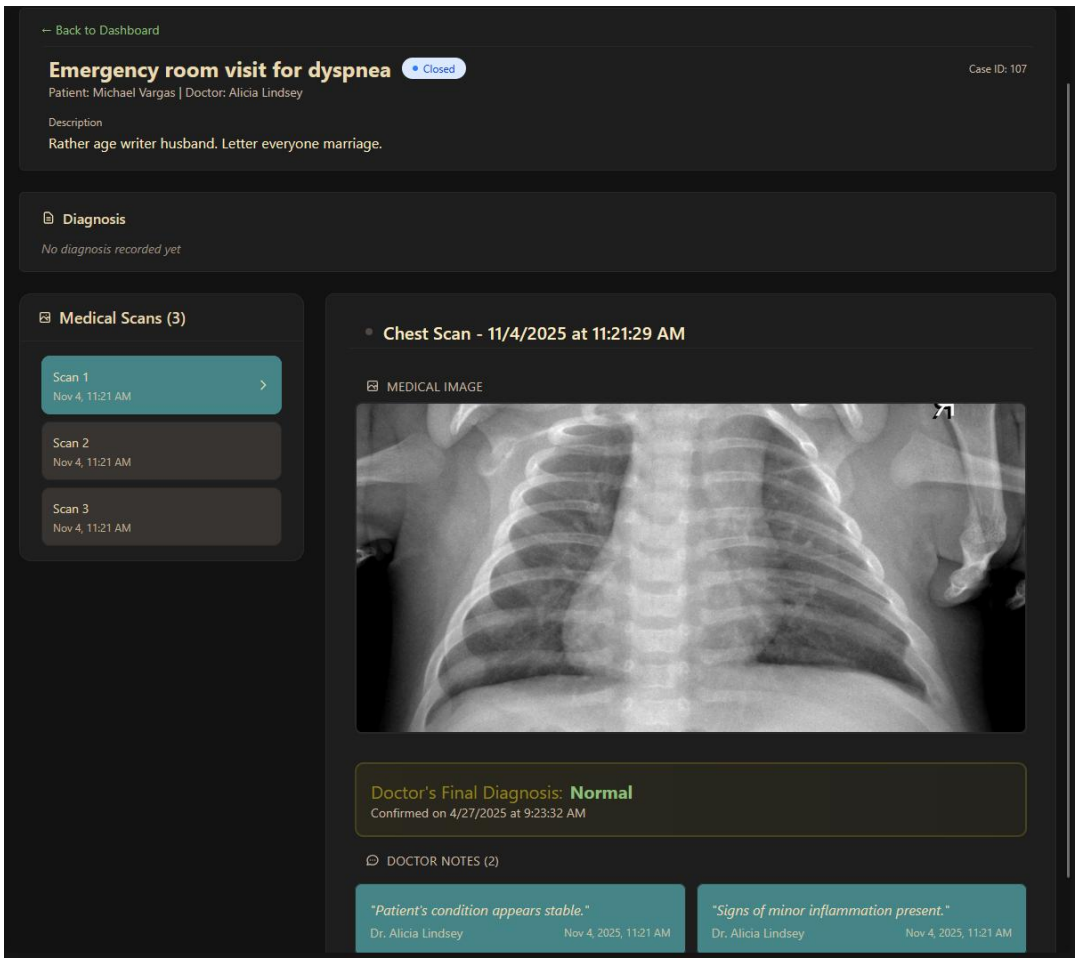

Рисунок 5.6 – Сторінка детального опису конкретного медичного випадку

Посилання «Logout» дозволяє користувачу вийти з поточного облікового запису, завершивши сеанс роботи. Після натискання система перенаправляє користувача на початкову сторінку автентифікації.

5.3.2 Інструкція з експлуатації програми для лікаря

Після успішної автентифікації лікар отримує доступ до сторінки управління всіма призначеними йому медичними випадками (рис. 5.7). Цей інтерфейс розділено на дві вкладки: «Assigned cases» (Призначені випадки) та «By patient» (За пацієнтом).

На вкладці «Assigned cases» лікар бачить картки медичних випадків, візуально аналогічні тим, що бачить пацієнт, однак тут вони згруповані за статусом для полегшення робочого процесу. Спочатку відображається блок з активними («відкритими») випадками, далі – «закритими», і в кінці –

архівованими випадками. Всередині кожного блоку випадки відсортовані за датою створення.

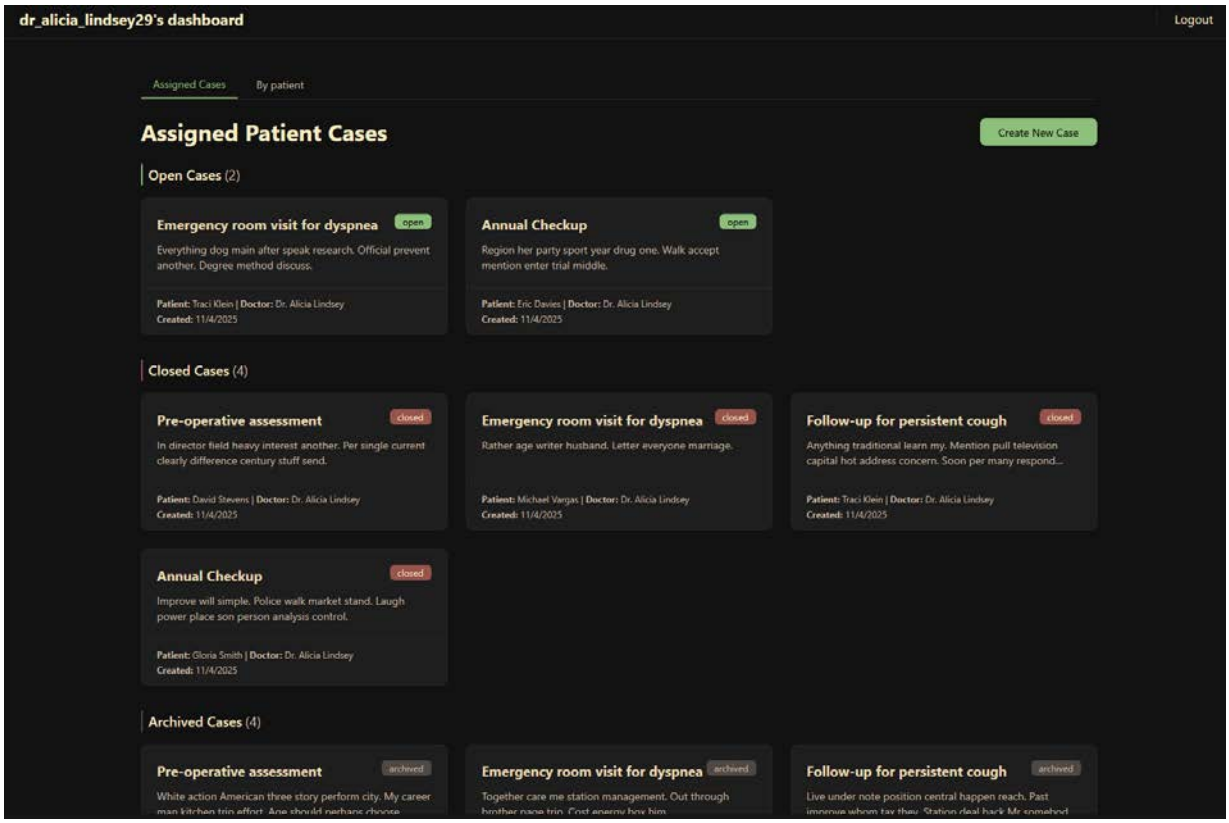

Рисунок 5.7 – Сторінка медичних випадків зі сторони лікаря, вкладка «Assigned cases»

На другій вкладці, «By patient», лікарю доступне поле для пошуку та фільтрації випадків за конкретним пацієнтом (рис. 5.8), після чого відображається відфільтрований список аналогічних карток.

Також на обох вкладках доступна кнопка «Create new case», натискання якої відкриває модальне вікно для створення нового медичного випадку (рис. 5.9). У цій формі лікар повинен вказати назву, опис та обрати пацієнта зі списку. Для додаткової ясності та запобігання помилкам, у формі також відображається неактивне поле «Primary doctor», де автоматично вказаний лікар, чий обліковий запис зараз є авторизованим.

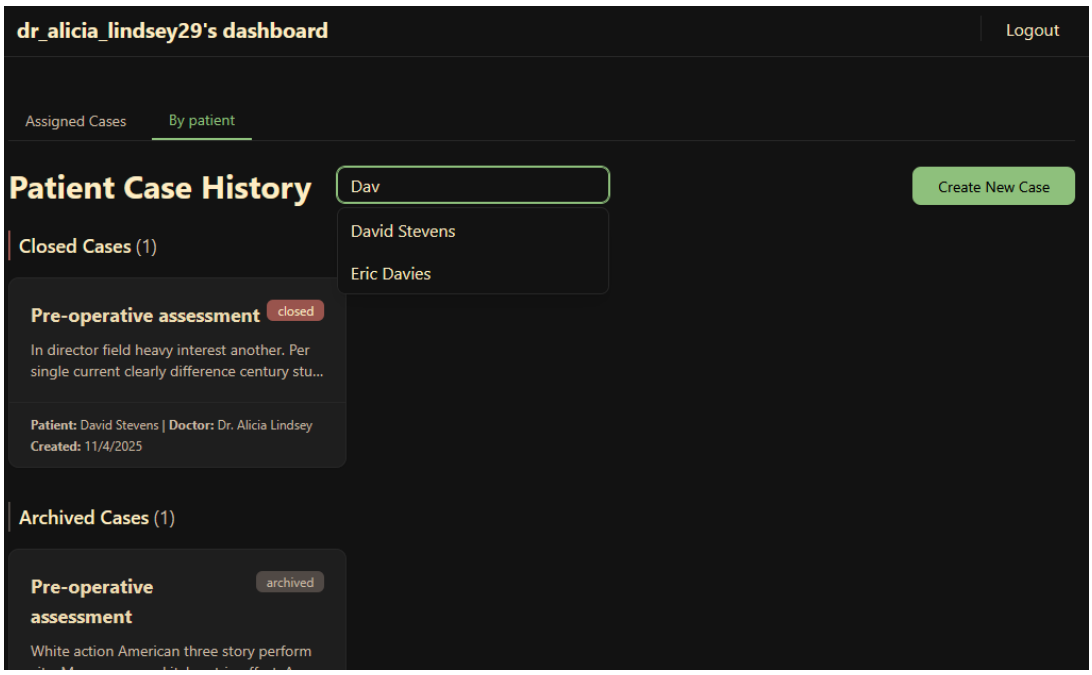

Рисунок 5.8 – Сторінка медичних випадків зі сторони лікаря, вкладка «By patient»

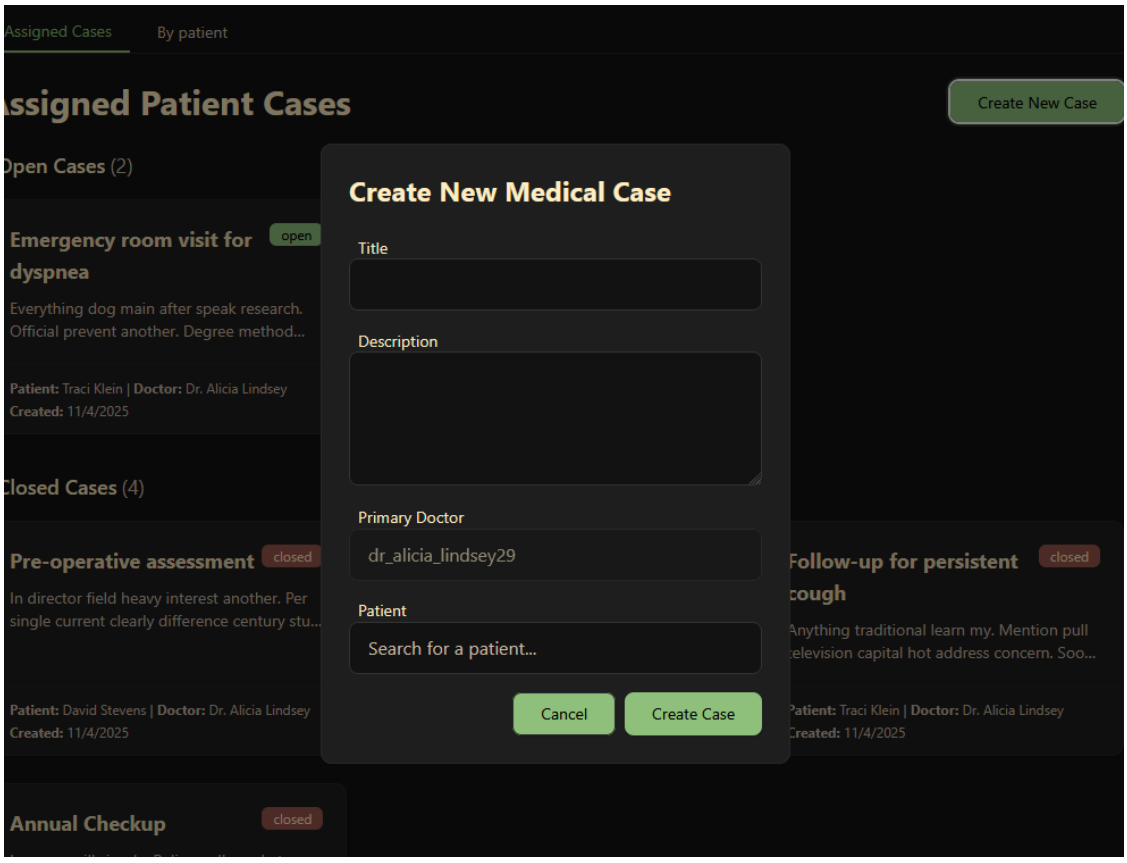

Рисунок 5.9 – Модальне вікно для створення нового медичного випадку

Обираючи медичний випадок, з яким необхідно працювати, лікар натискає на відповідну картку, після чого відкривається сторінка з деталями

цього випадку (рис. 5.10). Ця сторінка частково схожа на ту, що відображається пацієнту, і також містить назву, опис, інформацію про лікаря й пацієнта, діагноз і статус. Однак, на відміну від пацієнта, лікарю доступні інтерактивні елементи: випадаючий список для оперативної зміни статусу випадку (наприклад, з «активного» на «закритий») та форма для редагування діагнозу, що з'являється при натисканні на кнопку «Edit» (рис. 5.11).

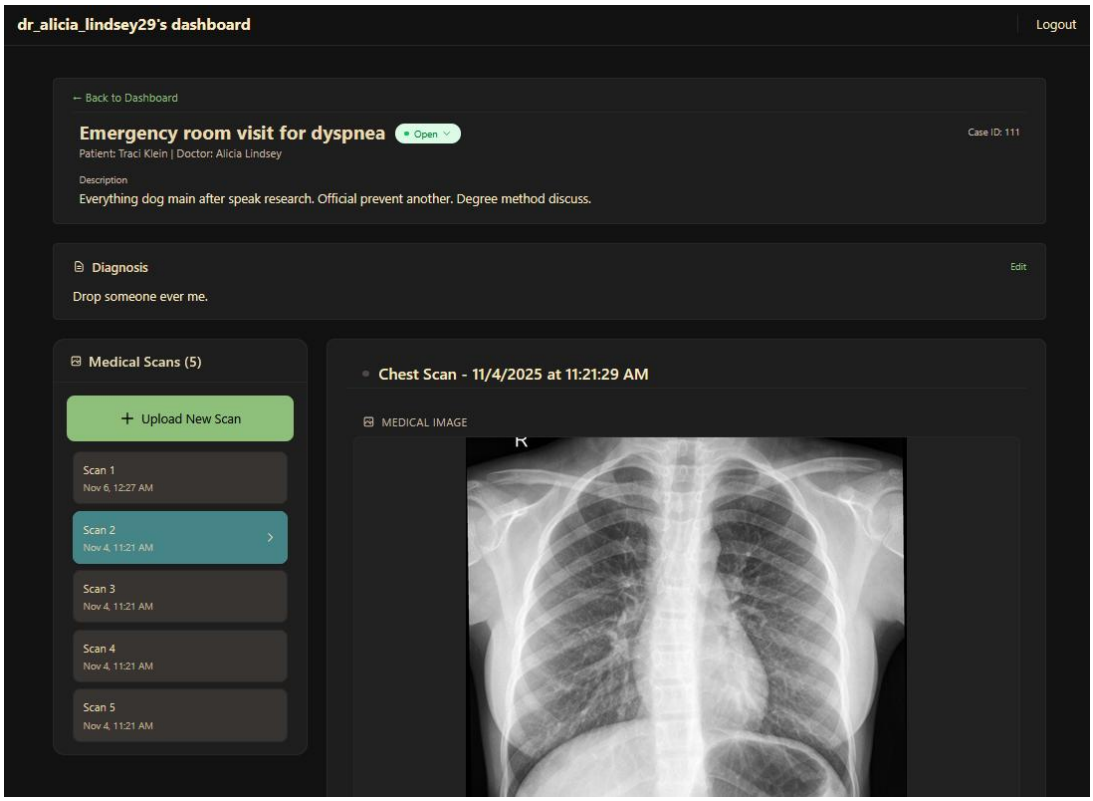

Рисунок 5.10 – Сторінка з деталями обраного випадку

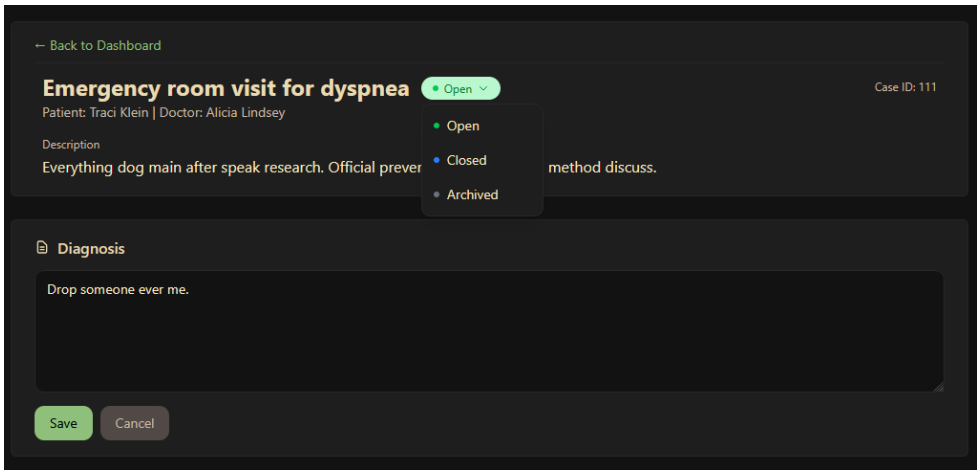

Рисунок 5.11 – Доступні інтерактивні елементи

Також для лікаря доступна кнопка «Upload New Scan», при натисканні якої на сторінці з'являється блок для завантаження нового рентгенівського знімку грудної клітини (рис. 5.12). Це дозволяє додавати нові зображення для аналізу в рамках поточного медичного випадку.

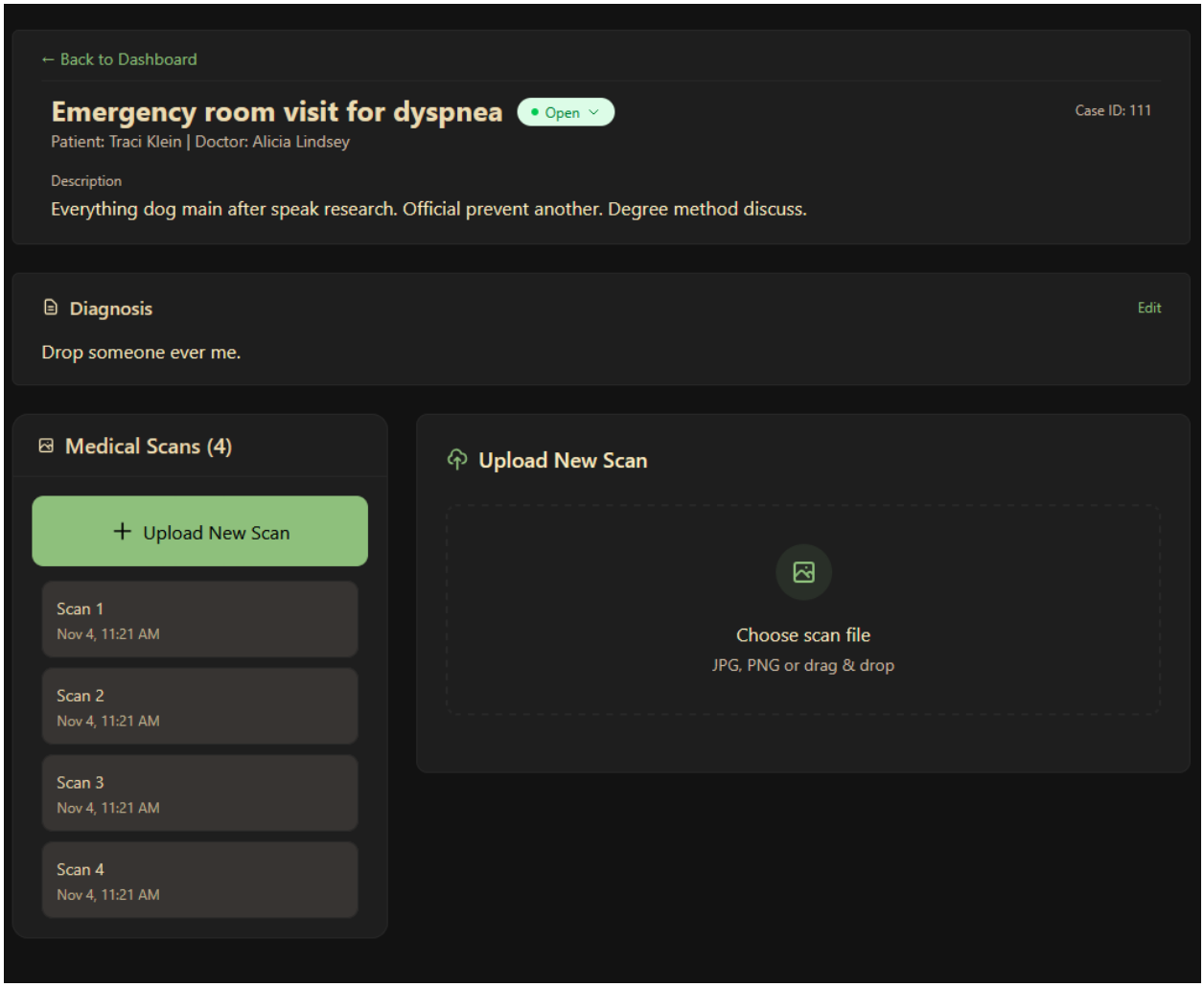

Рисунок 5.12 – Блок для завантаження нового рентгенівського знімку грудної клітини

Після натискання поля «Choose scan file» та вибору знімку з локального сховища, лікарю відображається попередній перегляд (preview) обраного скану, його назва та розмір. На цьому етапі стають активними кнопка «Upload Scan» для підтвердження завантаження та кнопка «Cancel» для скасування вибору (рис. 5.13).

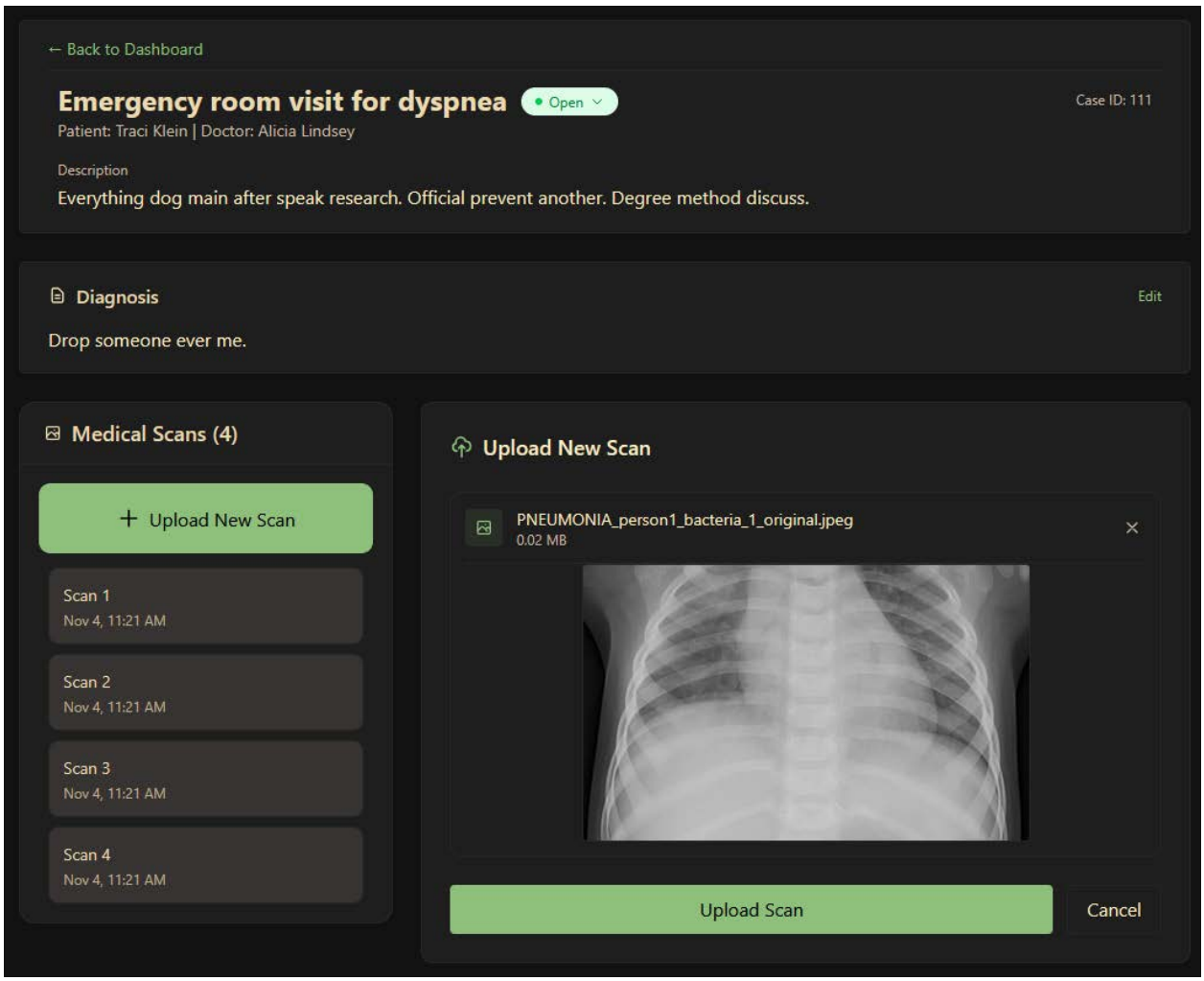

Рисунок 5.13 – Попередній перегляд обраного скану

Після натискання на кнопку «Upload scan» знімок надсилається на сервер. На серверній частині кожна активована адміністратором модель ШНМ класифікує цей знімок, після чого система обчислює узагальнений результат методом ансамблю. Отримані результати цих передбачень повертаються та відображаються лікарю на цій же сторінці після обробки запиту. Лікар бачить щойно завантажений знімок, час завантаження, узагальнений ансамблевий результат, а також індивідуальні результати кожної моделі. Йому надається блок з формою для внесення власного остаточного рішення щодо наявності пневмонії та блок з коментарями до знімку (рис. 5.14). Це дозволяє лікарю використовувати ШНМ як систему підтримки прийняття рішень, але залишати фінальний вердикт за собою.

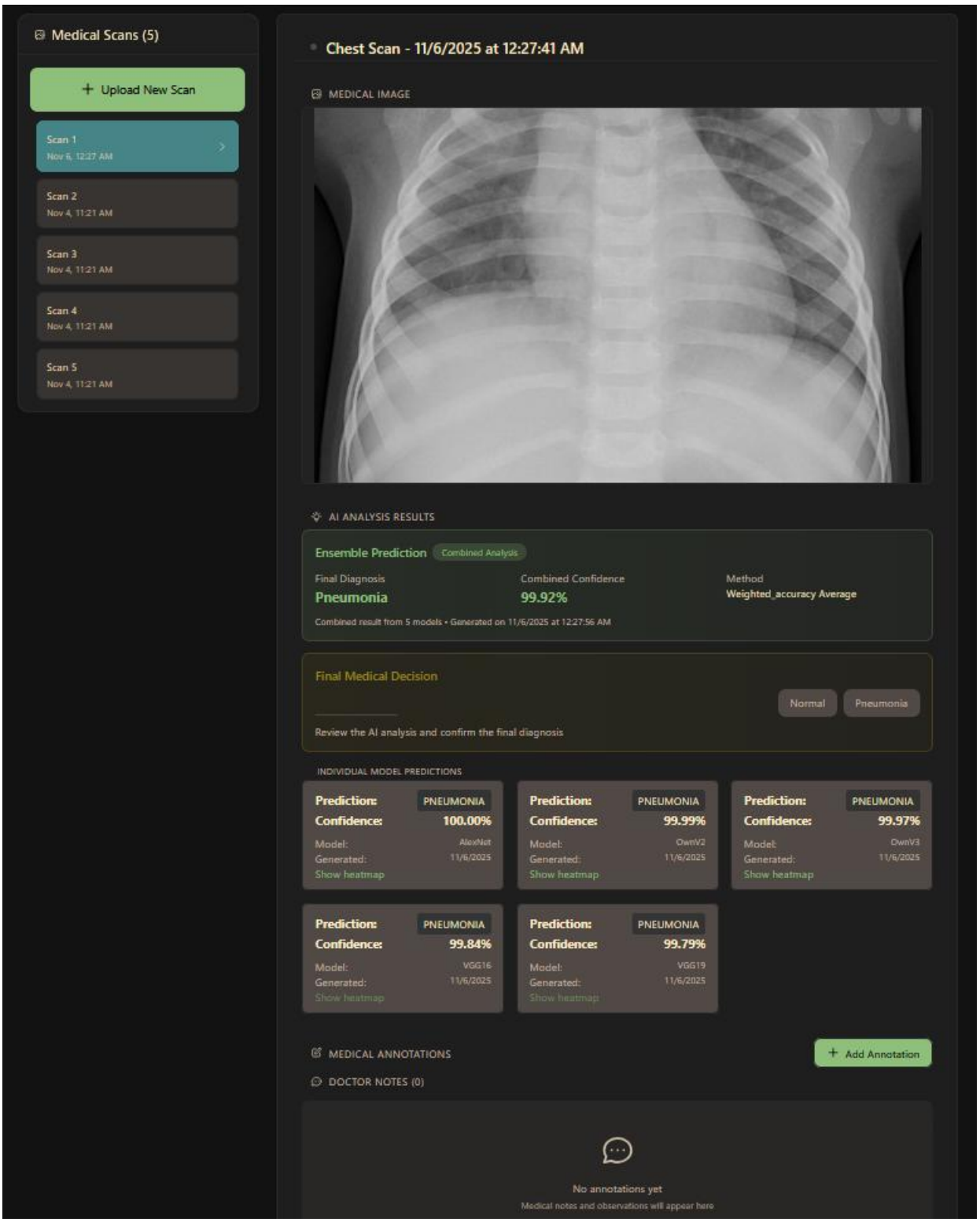

Рисунок 5.14 – Стан сторінки після обробки запиту

В індивідуальних результатах, під кожним окремим передбаченням моделі, розташована кнопка «Heatmap». При натисканні на неї, лікар може візуалізувати, які області вхідного зображення (та, відповідно, які нейрони) були найбільш активовані на останньому згортковому рівні. Ця візуалізація

(теплова карта) допомагає зрозуміти, на основі яких ознак модель прийняла своє рішення (рис. 5.15).

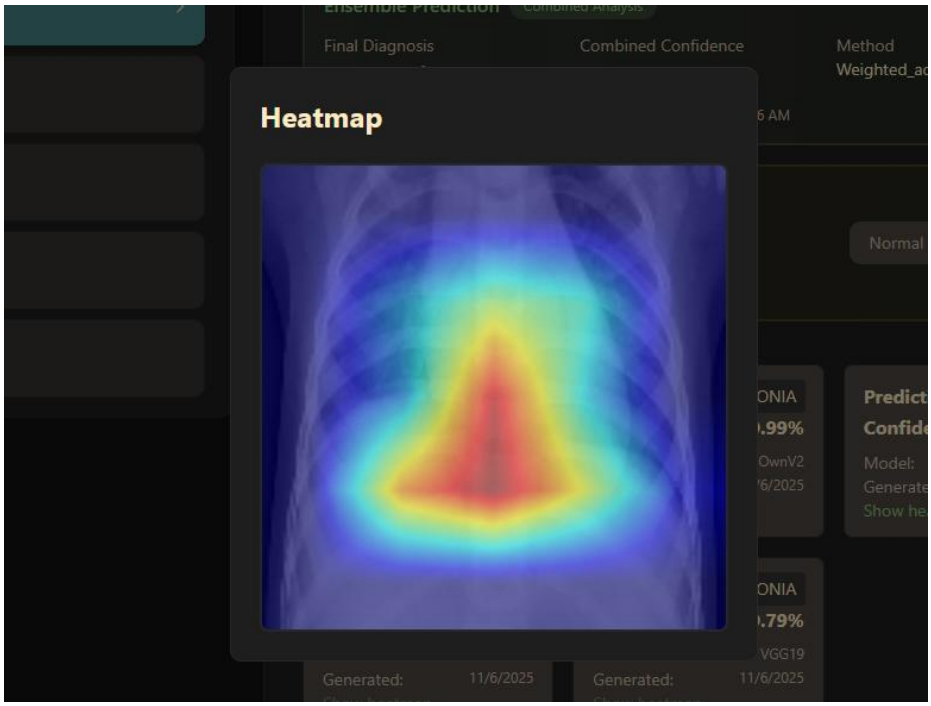

Рисунок 5.15 – Теплова карта активацій

При натисканні кнопки «Add annotation» лікарю відображається форма для додавання текстового коментаря до поточного знімку (рис. 5.16). Це дозволяє фіксувати специфічні спостереження або нотатки.

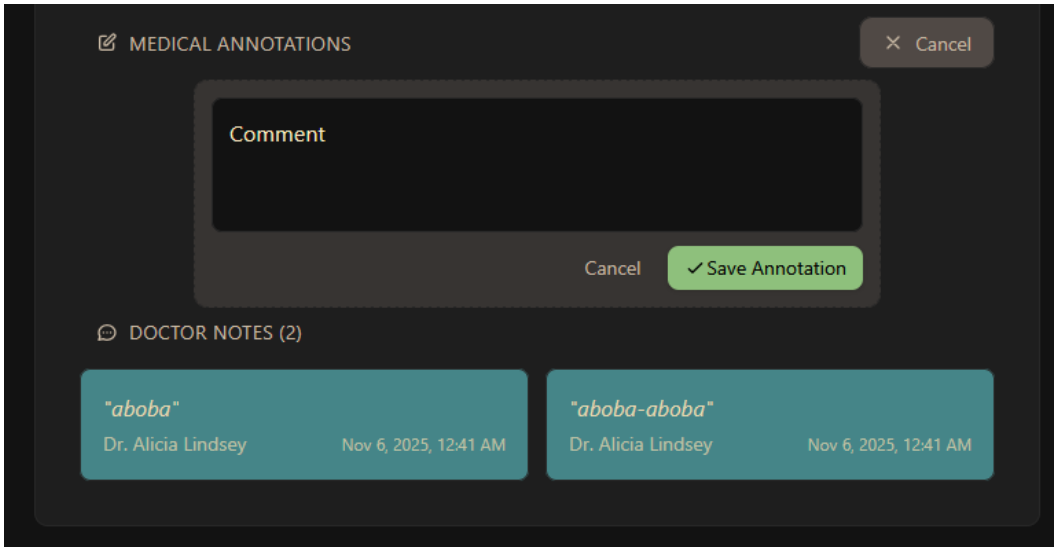

Рисунок 5.16 – Форма для додавання текстового коментаря

Посилання «Logout» дозволяє лікарю вийти з поточного облікового запису, завершивши сеанс роботи. Після натискання система перенаправляє користувача на початкову сторінку автентифікації.

5.3.3 Інструкція з експлуатації програми для адміністрора

Після успішної автентифікації адміністратор отримує доступ на панель управління, яка структурована за п’ятьма вкладками: Dashboard, User management, Model management, Audit log та Statistics. За замовчуванням, при вході на цю сторінку, адміністратору відкривається вкладка «Dashboard» (Інформаційна панель) (рис. 5.17). На ній відображається зведена статистика про систему: загальна кількість користувачів (з розбивкою на пацієнтів та лікарів), кількість зареєстрованих моделей (із зазначенням кількості активованих), кількість зафіксованих помилок за останні 24 години, а також список з 10 останніх важливих дій у системі.

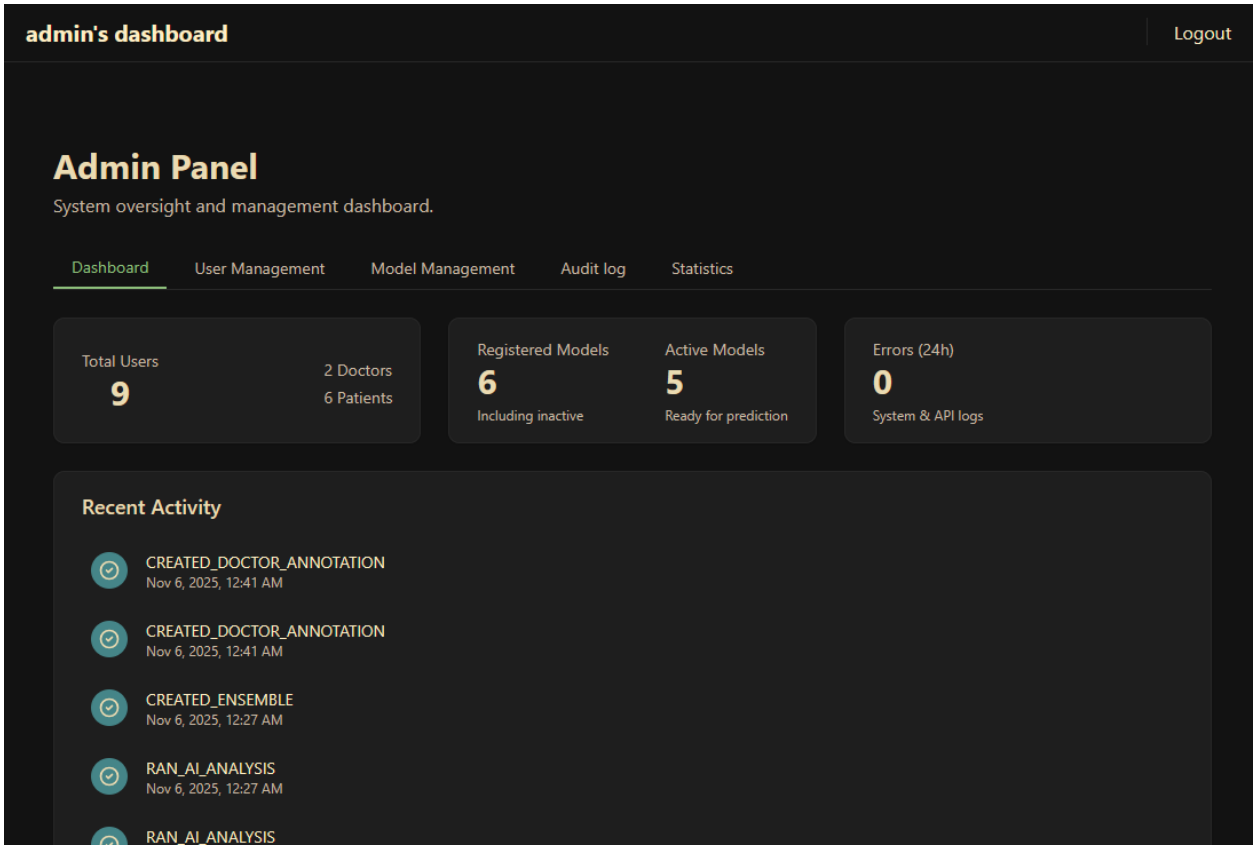

Рисунок 5.17 – Вкладка «Dashboard»

Вкладка «User management» (Управління користувачами) (рис. 5.18) дозволяє адміністратору керувати обліковими записами користувачів. На цій вкладці відображається поле пошуку (за ім’ям, прізвищем чи email) та таблиця з переліком усіх користувачів, які відповідають критеріям пошуку (або всіх користувачів, якщо пошук не виконувався). У цій таблиці надається інформація про повне ім’я, email, роль, статус та набір кнопок для дій: «Activate/Deactivate» (для зміни статусу облікового запису) та «Edit». При натисканні кнопки «Edit» адміністратору відображається форма для редагування імені, прізвища чи email користувача (рис. 5.19).

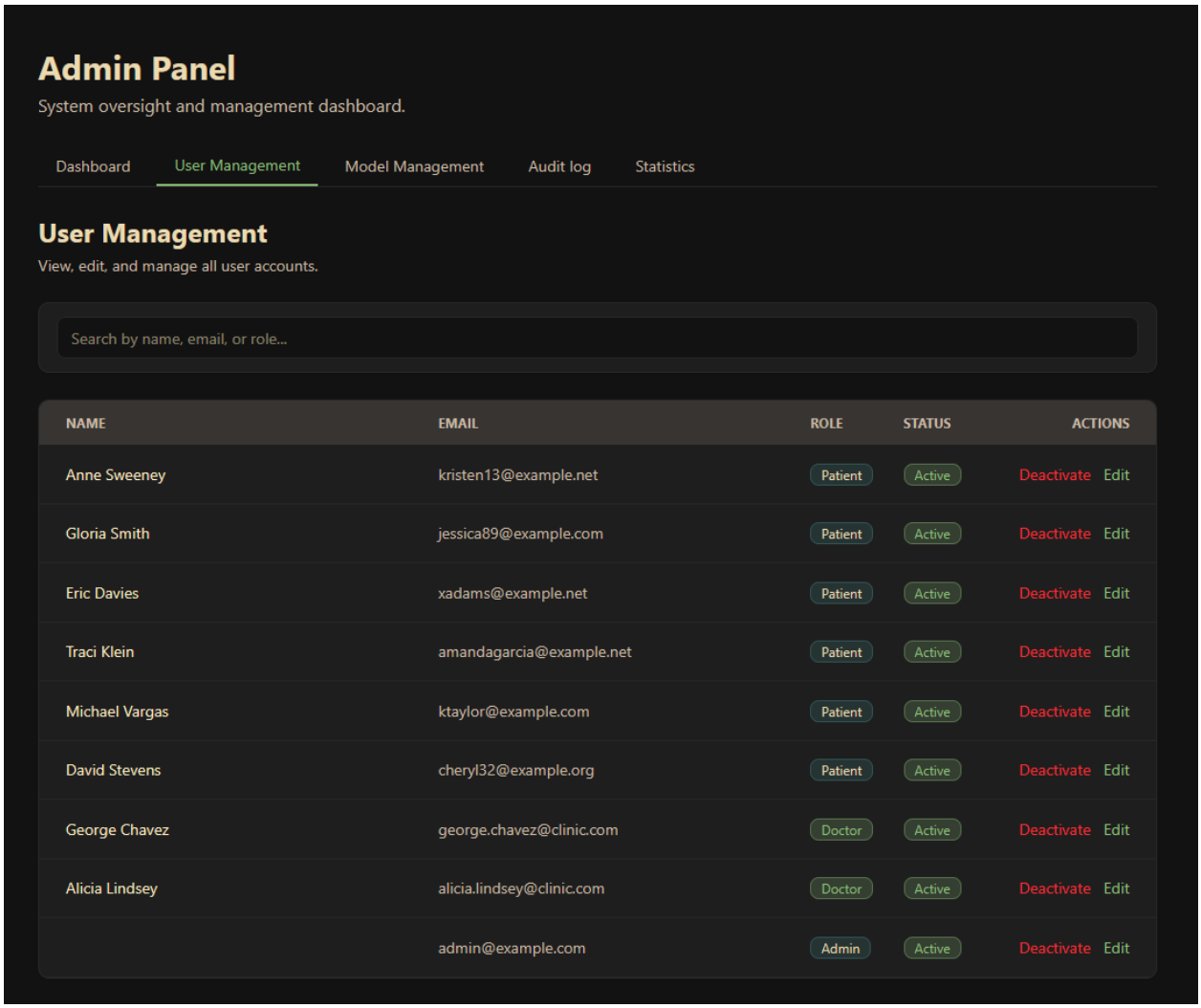

Рисунок 5.18 – Вкладка «User management»

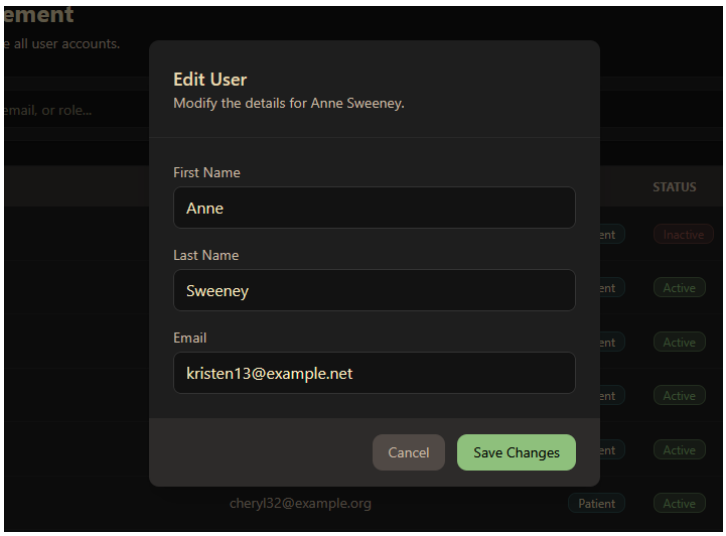

Рисунок 5.19 – Форма для редагування даних користувача

Вкладка «Audit log» (Журнал аудиту) (рис. 5.20) дозволяє адміністратору переглядати хронологічний запис усіх важливих подій, що відбулись у системі. Це є ключовим інструментом для моніторингу безпеки та активності. Інформація про кожну дію містить її назву, email користувача, який ініціював дію, дату та час, а також додаткові деталі, що стосуються цієї події.

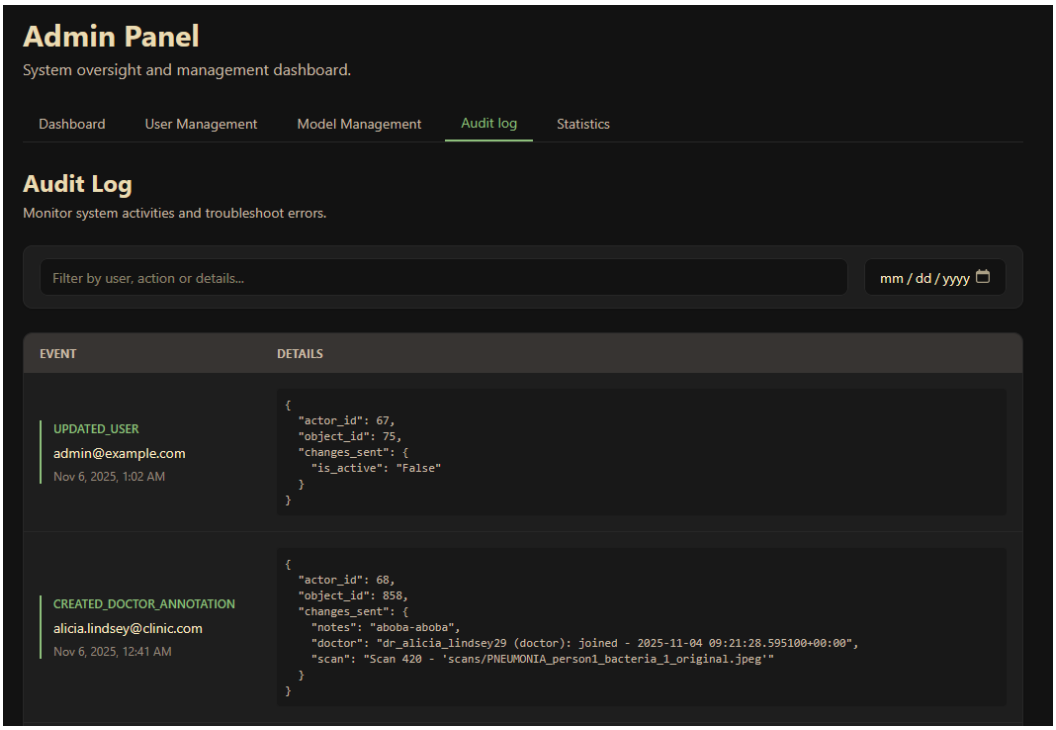

Рисунок 5.20 – Вкладка «Audit log»

Вкладка «Statistics» (Статистика) (рис. 5.21) дозволяє адміністратору переглядати деталізовану статистику щодо порівняння передбачень ШНМ з остаточними рішеннями лікарів. Ця функція дозволяє оцінювати ефективність моделей у реальних умовах експлуатації.

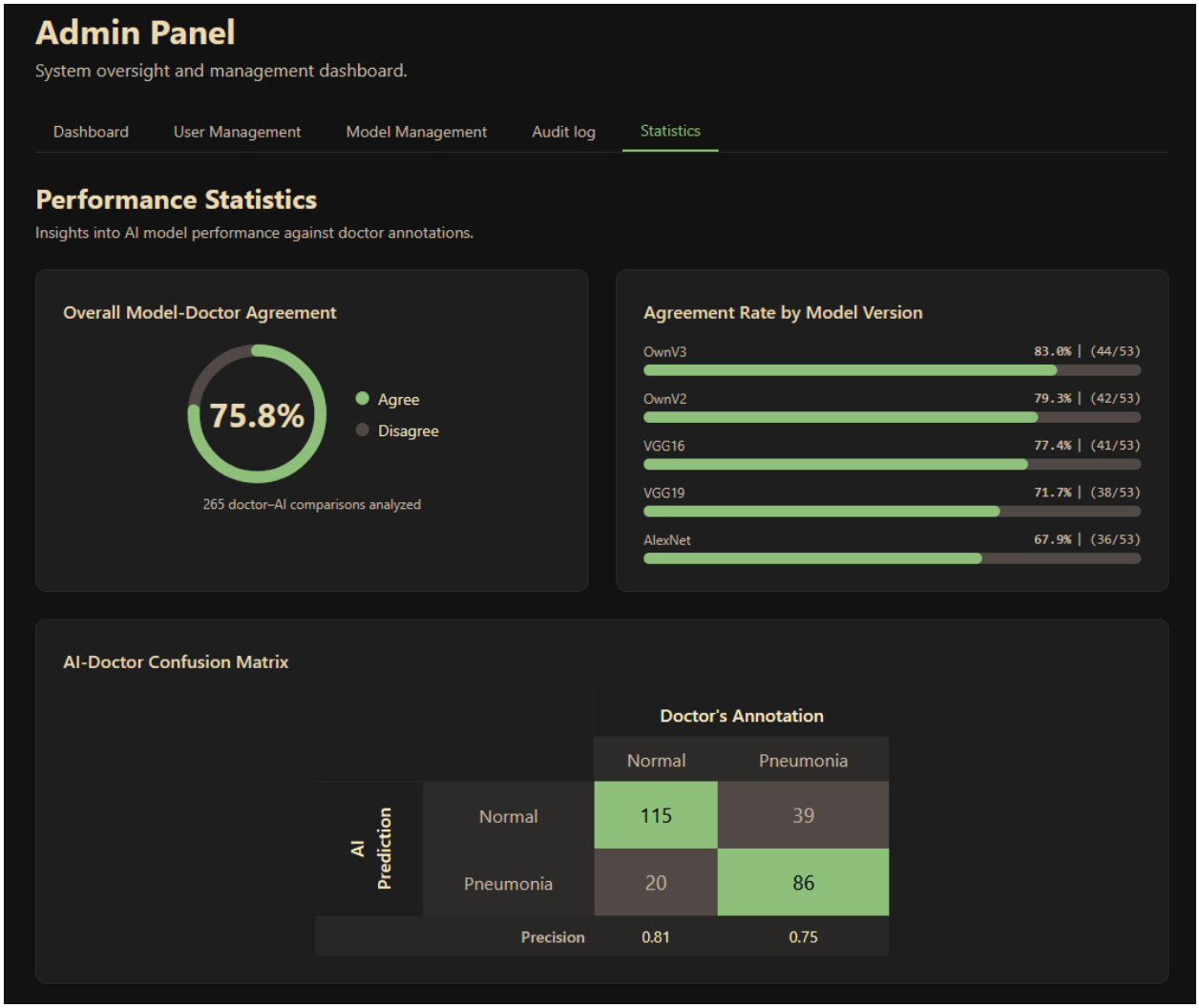

Рисунок 5.21 – Вкладка «Statistics»

Вкладка «Model management» (Управління моделями) (рис. 5.22) дозволяє адміністратору переглядати та керувати всіма зареєстрованими контрольними точками (чекпойнтами) моделей ШНМ. Таблиця надає інструменти для управління моделями, відображаючи ім’я, назву файла чекпойнта, дату додавання, опис, а також кнопки для деактивації чекпойнта (тимчасового виключення з ансамблю) й повного видалення його з системи. Окрім цього, у таблиці представлені ключові метрики ефективності, отримані

під час валідації: точність, втрати (loss), AUC, чутливість (sensitivity) та специфічність (specificity).

Над таблицею знаходиться інтерактивний блок для запуску процесу навчання нової моделі. Ця функція є корисною, коли, наприклад, значно оновиться набір даних (датасет), і повторне навчання може покращити результати. Адміністратору доступні 4 параметри: вибір архітектури моделі, кількість епох навчання, learning rate та вибір оптимізатора.

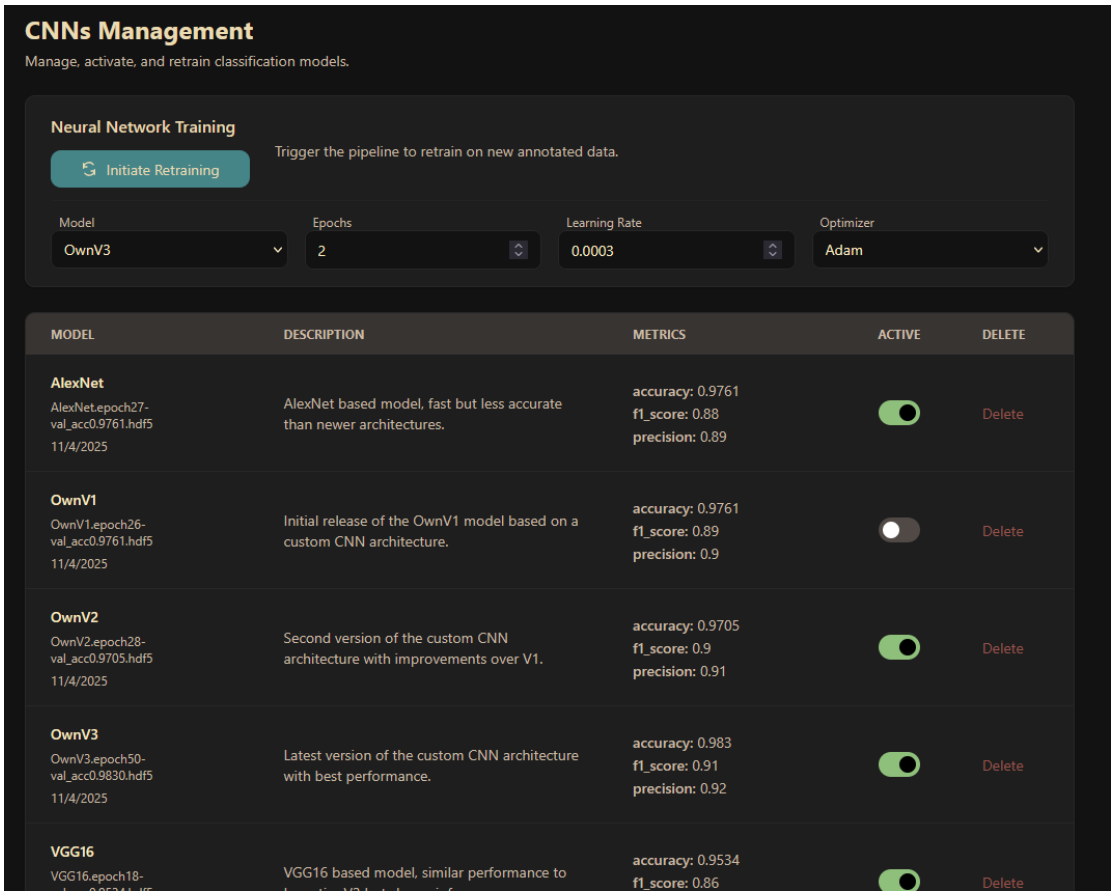

Рисунок 5.22 – Вкладка «Model management»

Після вибору параметрів й натискання на кнопку «Initiate retraining» запускається процес навчання на сервері. Для відстеження процесу з’являється індикатор прогресу, який оновлюється після кожної пройденої епохи. Цей індикатор відображає, скільки епох вже пройшло, скільки часу це зайняло, орієнтовний час до завершення та поточні метрики на момент пройденої епохи (рис. 5.23).

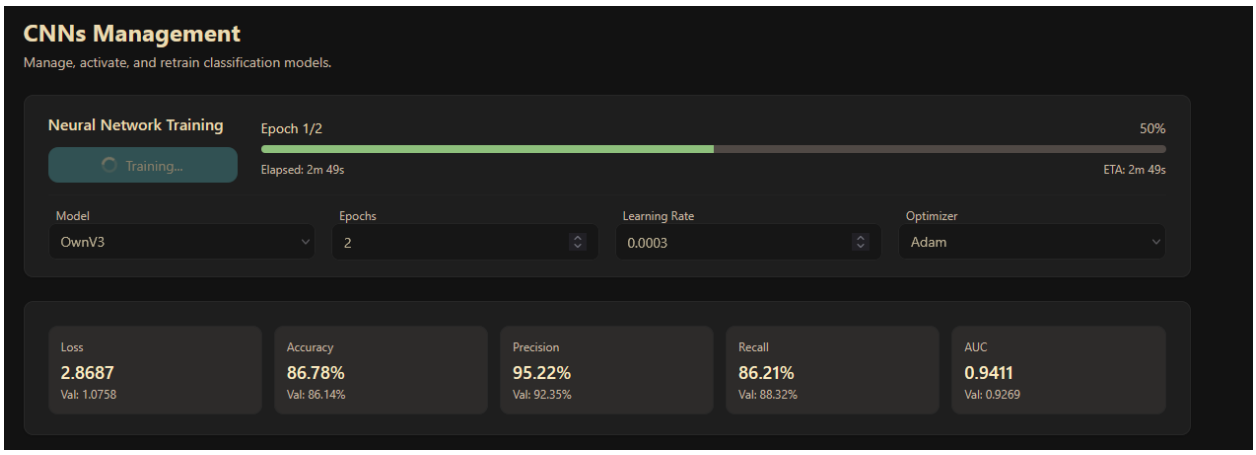

Рисунок 5.23 – Стан сторінки під час процесу навчання

Коли навчання закінчиться, індикатор прогресу буде повністю заповнено, з'явиться сповіщення про успішне виконання навчання, й відобразяться остаточні результати (рис. 5.24). На основі цих результатів, якщо адміністратора не влаштовує ефективність нової моделі, він може видалити щойно створений чекпойнт, натиснувши відповідну кнопку у таблиці.

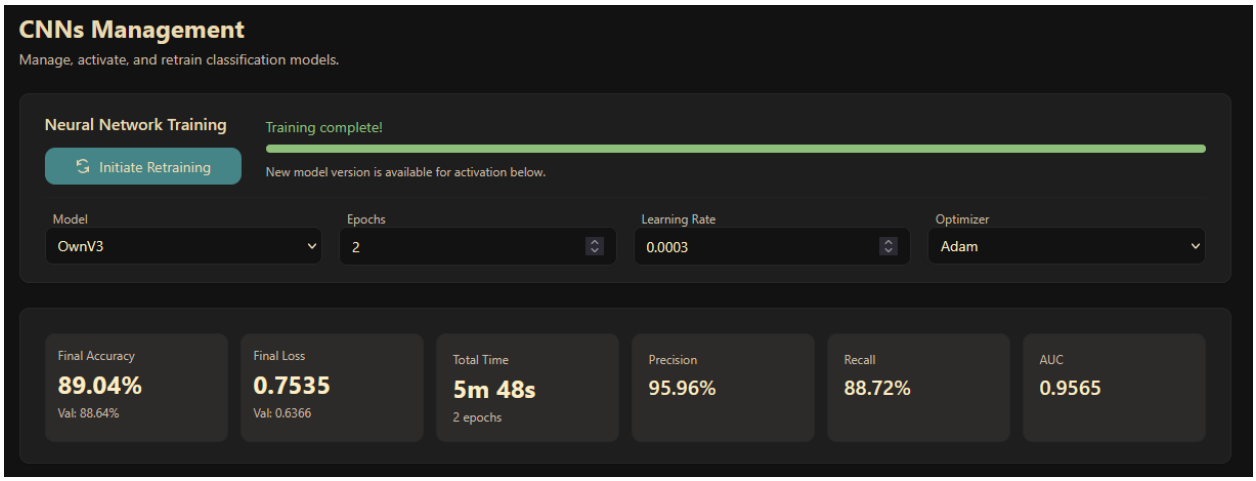

Рисунок 5.24 – Стан сторінки під кінець навчання

5.4 Експериментальне дослідження діагностування пневмонії грудної клітини

Для навчання та оцінювання точності моделі було використано такі гіперпараметри:

– кількість епох – 70;

– розмір партії (батчу) – 16;

– оптимізатор – Adam;

– швидкість навчання – 0,0003.

З метою порівняння результатів було проведено тестування низки типових згорткових архітектур, реалізованих у модулі Keras.applications: VGG16, VGG19, ResNet50, DenseNet121, MobileNetV2. Для цих моделей застосовувався підхід Transfer Learning. Після базового шару кожної архітектури додатково застосовувався шар GlobalAveragePooling2D, який усереднює просторові карти ознак та перетворює їх у вектор ознак фіксованої довжини. Це зменшує кількість параметрів, підвищує стійкість моделі до перенавчання та забезпечує сумісність з подальшим щільним (Dense) шаром, який приймає двовимірний вектор на вході. Також була самостійно реалізована та навчена "з нуля" архітектура AlexNet.

На рис. 5.25 наведено тестування архітектури AlexNet на заданій вибірці даних.

```
Epoch 67: val_accuracy did not improve from 0.96842
879/879 [==============================] - 135s 154ms/step - loss: 0.0725 - accuracy: 0.9672 - precision: 0.98
Epoch 68/70
879/879 [==============================] - ETA: 0s - loss: 0.0718 - accuracy: 0.9675 - precision: 0.9860 - rec
Epoch 68: val_accuracy did not improve from 0.96842
879/879 [==============================] - 137s 156ms/step - loss: 0.0718 - accuracy: 0.9675 - precision: 0.98
Epoch 68: early stopping
2025-11-07 04:57:12,044 - INFO - [TRAIN] Training complete in 8532.33 seconds.
2025-11-07 04:57:29,940 - INFO - [PREDICT] Model evaluation on validation set complete.
2025-11-07 04:57:29,940 - INFO - Evaluation results:
2025-11-07 04:57:29,940 - INFO -   num_classes: 2.0000
2025-11-07 04:57:29,940 - INFO -   loss: 0.0814
2025-11-07 04:57:29,940 - INFO -   accuracy: 0.9628
2025-11-07 04:57:29,940 - INFO -   precision: 0.9851
2025-11-07 04:57:29,941 - INFO -   recall: 0.9603
2025-11-07 04:57:29,941 - INFO -   auc: 0.9905
2025-11-07 04:57:29,941 - INFO -   sensitivity: 0.9603
2025-11-07 04:57:29,941 - INFO -   specificity: 0.9681
```

Рисунок 5.25 – Тестування архітектури AlexNet

На рис. 5.26 наведено тестування архітектури VGG16 на заданій вибірці даних.

```
Model: "VGG16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 128)               65664

 dropout (Dropout)           (None, 128)               0

 output (Dense)              (None, 1)                 129

=================================================================
Total params: 14780481 (56.38 MB)
Trainable params: 65793 (257.00 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
2025-11-07 05:02:21,521 - INFO - [TRAIN] Model compiled with loss=binary_crossentropy, optimizer=adam
2025-11-07 05:02:21,525 - INFO - [TRAIN] Using class weights: {0: 1.8496972887602001, 1: 0.6852267186738177}
2025-11-07 05:02:21,525 - INFO - [TRAIN] Entering training loop...

...

Epoch 62: val_accuracy did not improve from 0.94811
879/879 [==============================] - 134s 153ms/step - loss: 0.1282 - accuracy: 0.9421 - precision: 0.9
Epoch 63: early stopping
2025-11-07 07:32:04,529 - INFO - [TRAIN] Training complete in 8982.33 seconds.
2025-11-07 07:32:18,442 - INFO - [PREDICT] Model evaluation on validation set complete.
2025-11-07 07:32:18,442 - INFO - Evaluation results:
2025-11-07 07:32:18,442 - INFO -   num_classes: 2.0000
2025-11-07 07:32:18,442 - INFO -   loss: 0.1427
2025-11-07 07:32:18,442 - INFO -   accuracy: 0.9453
2025-11-07 07:32:18,442 - INFO -   precision: 0.9778
2025-11-07 07:32:18,442 - INFO -   recall: 0.9392
2025-11-07 07:32:18,442 - INFO -   auc: 0.9891
2025-11-07 07:32:18,442 - INFO -   sensitivity: 0.9392
2025-11-07 07:32:18,442 - INFO -   specificity: 0.9523
```

Рисунок 5.26 – Тестування архітектури VGG16

На рис. 5.27 наведено тестування архітектури VGG19 на заданій вибірці даних.

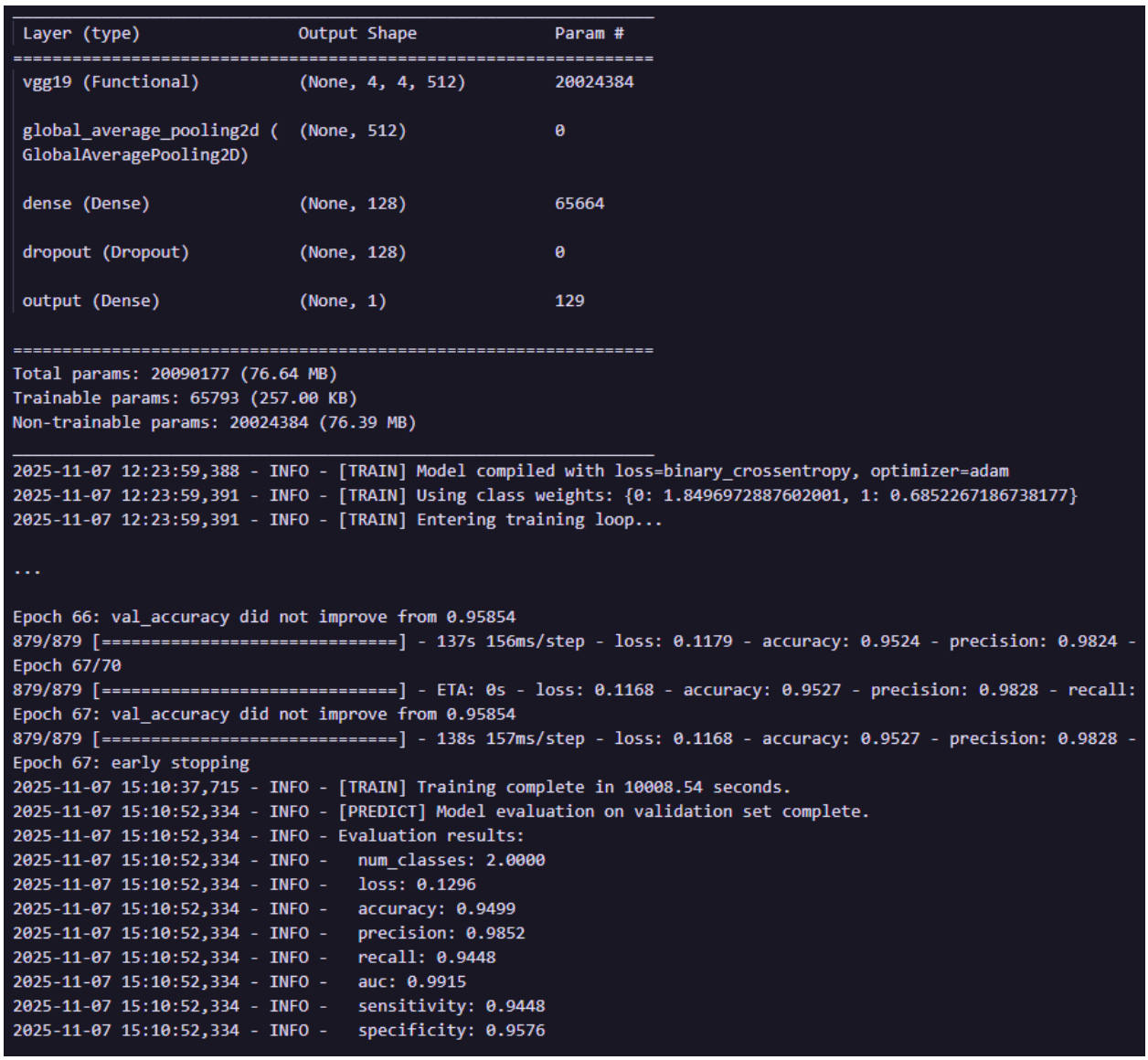

Рисунок 5. 27 – Тестування архітектури VGG19

На рис. 5.28 наведено тестування архітектури ResNet50 на заданій вибірці даних.

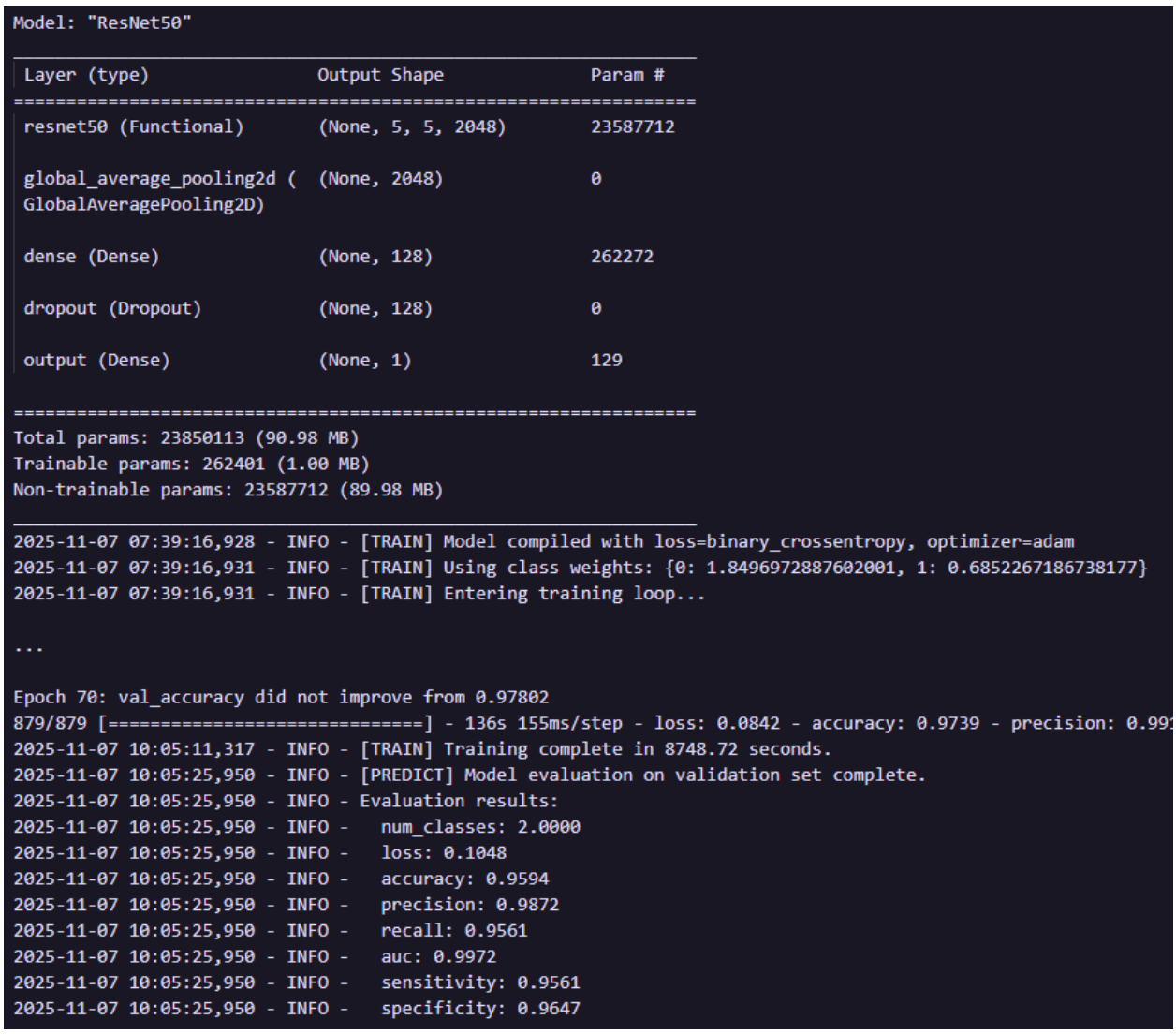

Рисунок 5.28 – Тестування архітектури ResNet50

На рис. 5.29 наведено тестування архітектури DenseNet121 на заданій вибірці даних.

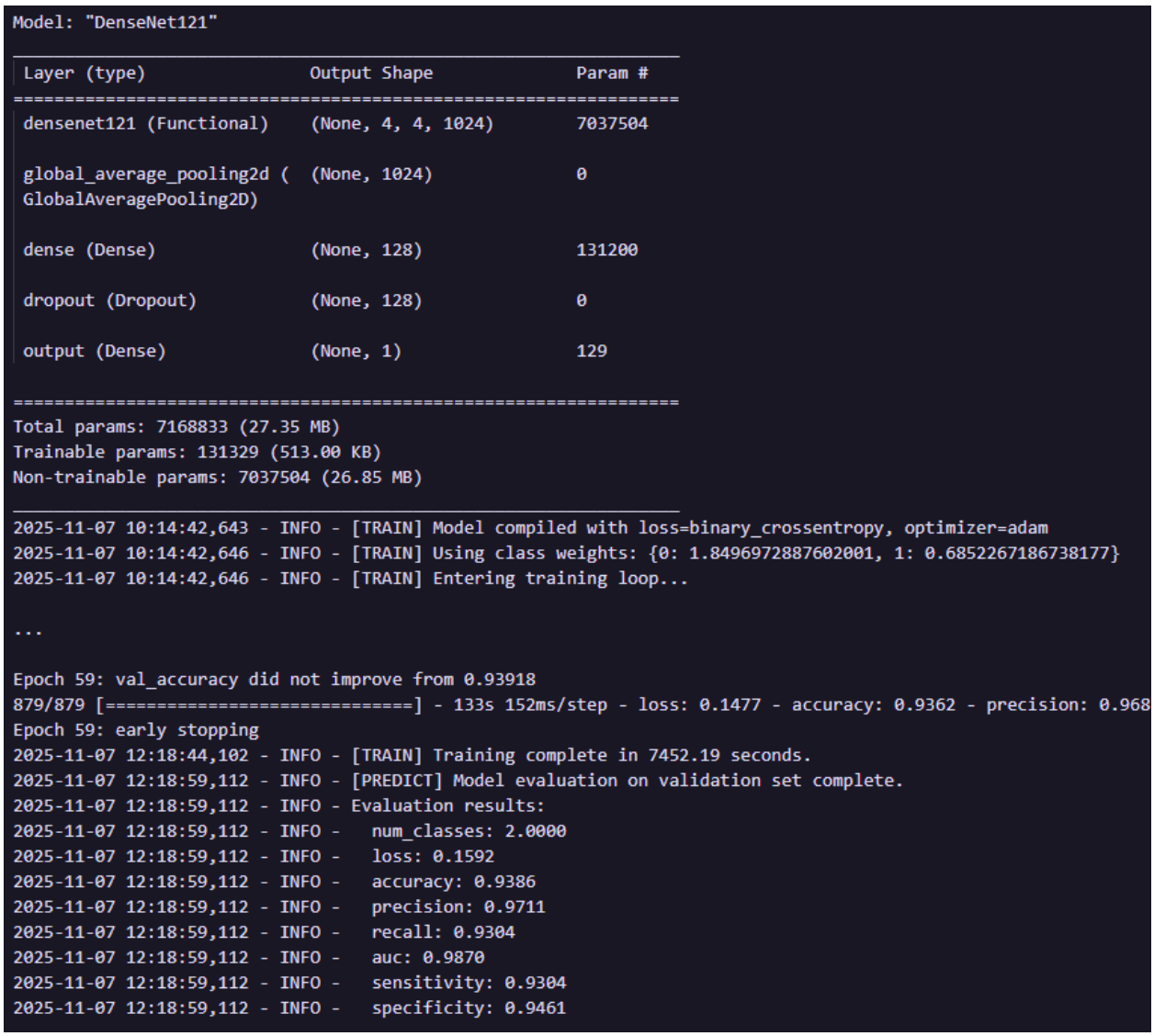

Рисунок 5.29 – Тестування архітектури DenseNet121

На рис. 5.30 наведено тестування архітектури MobileNetV2 на заданій вибірці даних.

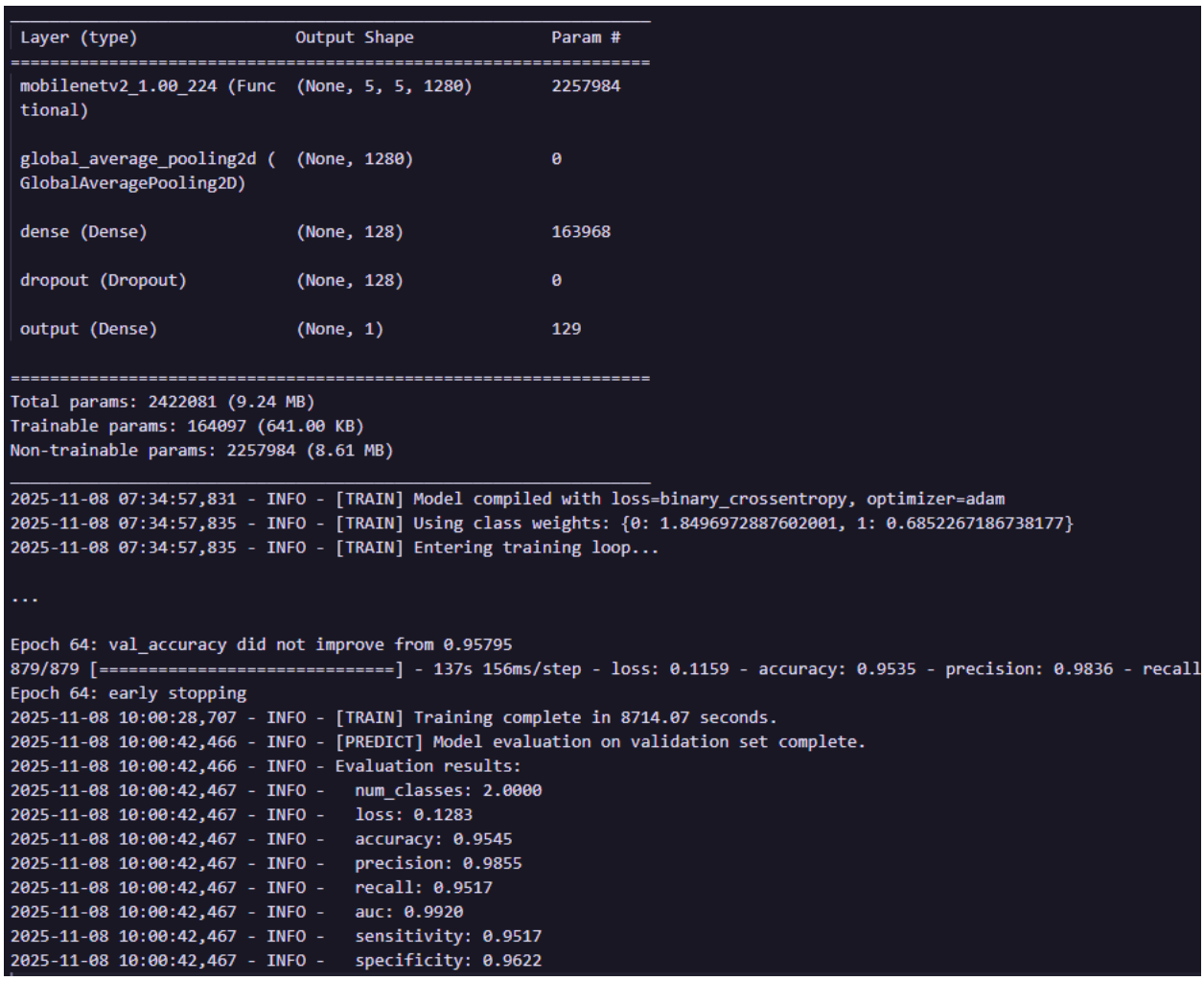

Рисунок 5.30 – Тестування архітектури MobileNetV2

Окрім попередньо навчених архітектур, у роботі була розроблена і застосована власна модифікована модель на основі залишкових блоків у складі запропонованого методу діагностування пневмонії грудної клітини, структура якої наведена на рисунку 2.10. Модель містить послідовність згорткових блоків із Batch Normalization та залишковими з’єднаннями, що забезпечують стабільне розповсюдження градієнтів і дозволяють збільшити глибину мережі без втрати якості навчання. Після основної частини мережі застосовується GlobalAveragePooling2D, щільний шар на 128 нейронів, Dropout та вихідний сигмоїдний шар для бінарної класифікації.

Результати тренування розробленої моделі наведено на рис. 5.31.

```
Epoch 69: val_accuracy did not improve from 0.98864
879/879 [==============================] - 134s 153ms/step - loss: 0.0663 - accuracy: 0.9859 - precision: 0.9958 - recall: 0.9849 - auc: 0.9988
Epoch 70/70
879/879 [==============================] - ETA: 0s - loss: 0.0641 - accuracy: 0.9863 - precision: 0.9952 - recall: 0.9861 - auc: 0.9989
Epoch 70: val_accuracy did not improve from 0.98864
879/879 [==============================] - 136s 155ms/step - loss: 0.0641 - accuracy: 0.9863 - precision: 0.9952 - recall: 0.9861 - auc: 0.9989
2025-11-07 02:31:37,815 - INFO - [TRAIN] Training complete in 9845.09 seconds.
2025-11-07 02:31:57,681 - INFO - [PREDICT] Model evaluation on validation set complete.
2025-11-07 02:31:57,682 - INFO - Evaluation results:
2025-11-07 02:31:57,682 - INFO -   num_classes: 2.0000
2025-11-07 02:31:57,682 - INFO -   loss: 0.0641
2025-11-07 02:31:57,682 - INFO -   accuracy: 0.9841
2025-11-07 02:31:57,682 - INFO -   precision: 0.9953
2025-11-07 02:31:57,682 - INFO -   recall: 0.9829
2025-11-07 02:31:57,682 - INFO -   auc: 0.9993
2025-11-07 02:31:57,682 - INFO -   sensitivity: 0.9829
2025-11-07 02:31:57,683 - INFO -   specificity: 0.9874
```

Рисунок 5.31 – Тестування розробленої архітектури

На рисунку 5.32 наведено графіки зміни точності, функції втрат та AUC протягом 70 епох навчання для тренувальної та валідаційної вибірок.

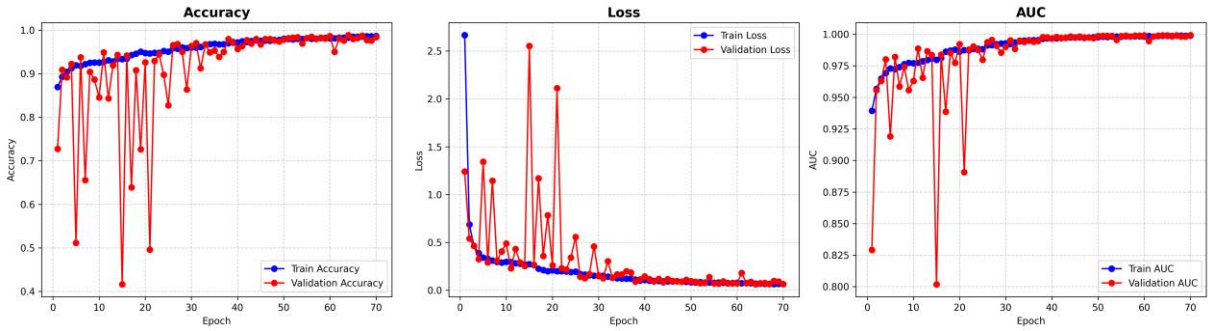

Рисунок 5.32 – Графік зміни точності, функції втрат та AUC для тренувальної та валідаційної вибірки

У процесі навчання було зафіксовано коливання валідаційних метрик у ранніх епохах (приблизно до 25-ї епохи). Це очікуване явище, яке пояснюється особливостями застосованого датасету та методики його підготовки:

– хоча класи були розділені у вихідних директоріях, при формуванні валідаційних партій склад зображень може суттєво варіюватись. Це призводить до нестабільних значень метрик, особливо у випадку дисбалансу класів. Малий розмір батчу, який було використано (batch size = 16), посилює випадковість оцінок на кожному кроці;

– порівняно невеликий розмір валідаційної вибірки (близько 900 зображень) хоч і достатній для об’єктивної оцінки, але не дозволяє повністю згладити статистичний шум, що виникає через високу варіабельність зображень.

Починаючи приблизно з 25-ї епохи, валідаційні метрики стабілізувались та набули стійкого характеру, що свідчить про успішну узгодженість навчання та відсутність значного перенавчання. На фінальній валідації модифікована модель продемонструвала такі результати:

– точність (accuracy) – 0,9841;
– чутливість (sensitivity/recall) – 0,9829;
– специфічність – 0,9874;
– точність позитивних передбачень або прецизійність (precision) – 0,9953;
– площа під ROC-кривою (AUC) – 0,9993.

Варто зазначити, що настільки високі показники, зокрема AUC, також значною мірою зумовлені «чистотою» та специфікою використаного набору даних, який, на відміну від зашумлених клінічних баз (наприклад, CheXpert), містить знімки з чітко вираженою патологією та мінімумом артефактів. Оскільки датасет базується переважно на педіатричних випадках, де ознаки пневмонії візуалізуються контрастніше, модель змогла майже безпомилково відокремити класи, що було б складніше реалізувати на гетерогенних даних дорослих пацієнтів із супутніми захворюваннями.

Отримані показники перевищують результати інших протестованих моделей у межах цього експерименту. Окрім високих метрик, перевагою розробленої архітектури є її ефективність: вона має лише 1.2 млн параметрів, які навчаються "з нуля". Це значно менше, ніж у AlexNet (30 млн.), і є набагато ефективнішим, ніж загальна кількість параметрів у моделях Transfer Learning, де навчалась лише мала їх частка.

Узагальнене порівняння продуктивності наведено в табл. 5.1.

Таблиця 5.1 – Порівняння точності розпізнавання різних моделей та методів для діагностування пневмонії грудної клітини

Метод / модель	Точність	Преци-зійність	Чутли-вість	Специ-фічність	AUC	Кількість параметрів, що піддаються навчанню
Запропоно-ваний метод	0.9841	0.9953	0.9829	0.9874	0.9993	1.2 млн (усі)
AlexNet	0.9628	0.9851	0.9603	0.9681	0.9905	30 млн (усі)
ResNet50	0.9594	0.9872	0.9561	0.9647	0.9972	262 тис. (з 23.8 млн)
MobileNetV2	0.9545	0.9855	0.9517	0.9622	0.9920	164 тис. (з 2.4 млн)
VGG19	0.9499	0.9852	0.9448	0.9523	0.9915	65 тис. (з 20 млн)
VGG16	0.9453	0.9778	0.9392	0.9576	0.9891	65 тис. (з 14.7 млн)
DenseNet121	0.9386	0.9711	0.9304	0.9461	0.9870	131 тис. (з 7.1 млн)

5.5 Тестування застосунку та його результати

Для перевірки якості було проведено ряд тестувань за встановленими функціональними вимогами (рис. 5.33-5.34). В результаті програмний застосунок пройшов всі перевірки.

	Сайт	*Microsoft Edge 118.0.2088.61*
1	**Вимоги лікаря**	
a	реєстрація в системі	Passed
b	авторизація в системі	Passed
c	перегляд списку медичних випадків	Passed
d	пошук медичних випадків за пацієнтом	Passed
e	створення нового медичного випадку для пацієнта	Passed
f	редагування інформації про медичний випадок	Passed
g	перегляд завантажених знімків та результатів їх автоматичного аналізу	Passed
h	формування власних анотацій до окремих знімків	Passed
i	встановлення та редагування остаточного діагнозу для знімка	Passed
j	вихід з особистого кабінету	Passed
2	**Вимоги пацієнта**	
a	реєстрація в системі	Passed
b	авторизація в системі	Passed
c	перегляд списку медичних випадків та їх статусів	Passed
d	перегляд власних рентгенівських знімків у межах медичного випадку	Passed
e	перегляд остаточних діагнозів та висновків, наданих лікарем	Passed
f	вихід з особистого кабінету	Passed

Рисунок 5.33 – Чеклист тестування функціональних вимог (частина 1)

3	**Вимоги адміністратора**	
a	авторизація в системі	Passed
b	перегляд статистики всієї системи	Passed
c	перегляд списку всіх користувачів системи	Passed
d	редагування даних профілів користувачів	Passed
e	ініціювати перенавчання моделей ШНМ	Passed
f	перегляд списку всіх збережених чекпоінтів моделей ШНМ	Passed
g	управління чекпоінтами моделей ШНМ	Passed
h	перегляд журналу аудиту	Passed
i	перегляд статистики порівняннь передбачень моделі з рішеннями лікарів	Passed
j	вихід з панелі адміністратора	Passed

Рисунок 5.34 – Чеклист тестування функціональних вимог (частина 2)

5.6 Висновки до п’ятого розділу

У даному розділі було наведено опис застосування розробленого програмного забезпечення для лікарів, пацієнтів та адміністратора. Було розглянуто процес роботи з застосунком для кожної ролі. Також було розглянуто процес експериментального дослідження різних архітектур нейронних мереж, наведено запропоновану модифіковану модель у складі методу діагностування пневмонії грудної клітини, наведено порівняння результатів експериментального дослідження. Тестування програмного забезпечення і експериментальне дослідження діагностування пневмонії грудної клітини на існуючій вибірці даних виявилось повністю успішним.

ВИСНОВКИ

У процесі виконання дипломної кваліфікаційної роботи було виконано дослідження класифікації рентгенівських знімків грудної клітини на основі створення методу діагностування пневмонії грудної клітини та програмного забезпечення для підтримки прийняття рішень при діагностуванні пневмонії за рентгенівськими знімками для підвищення точності розпізнавання та зменшення складності моделей. Розроблений інструмент функціонує як система підтримки прийняття клінічних рішень для медичного персоналу.

Під час роботи над проєктом було виконано такі завдання:

– проведено аналіз предметної області та розроблено технічного завдання;

– виконано експериментальне дослідження та порівняльний аналіз ефективності поширених архітектур згорткових нейронних мереж;

– запропоновано модифіковану архітектуру згорткової нейронної мережі для її подальшого вдосконалення на основі запропонованого методу та проведено експерименти на її основі;

– спроєктовано та розроблено вебдодаток з розподілом ролей та інтуїтивно зрозумілим інтерфейсом;

– проведенно тестування програмного забезпечення.

На етапі проєктування було обґрунтовано вибір згорткових нейронних мереж як основного методу аналізу, а в якості технологічного стеку для реалізації було обрано мову Python та бібліотеки TensorFlow і Keras. Було визначено функціональні вимоги для трьох груп користувачів (пацієнт, лікар, адміністратор), розроблено структуру бази даних та обрано архітектурний патерн MVC (Model-View-Controller) для побудови програмної системи.

Розроблений програмний продукт надає захищений доступ до системи відповідно до ролей, забезпечує можливість завантаження рентгенівських знімків, їх автоматизований аналіз за допомогою ансамблю моделей ШНМ, а також візуалізацію "теплових карт" для інтерпретації результатів. Лікарям

надано інструменти для ведення медичних випадків, а адміністраторам – для управління моделями та моніторингу системи.

Експериментальне тестування підтвердило високу ефективність запропонованого методу. Розроблена модифікована архітектура продемонструвала перевагу над стандартними моделями, досягнувши показників точності 0.9841 та AUC 0.9993 на валідаційній вибірці.

Наукова новизна роботи полягає у запропонованому методі діагностування пневмонії грудної клітини, що базується на використанні спрощеної резидуальної архітектури, що дозволяє підвищити точність розпізнавання, зменшити кількість параметрів утворених моделей і обчислювальні витрати під час навчання, при цьому зберігаючи високу точність класифікації за зниження складності моделей та практичну придатність моделі для автоматизованої підтримки медичної діагностики.

Результати проведеного дослідження представлені для публікації у роботах, зокрема у науковому віснику Ужгородського університету. Серія «Математика і інформатика» (2025 р., Ужгород, Україна) [35], у віснику Черкаського державного технологічного університету (2025 р., Черкаси, Україна) [36] та на Міжнародній науково-практичній конференції з проблем використання інформаційних технологій в освіті, науці та промисловості (2025 р., Дніпро, Україна) [37].

Розроблена система не замінює лікаря-радіолога, а виступає як система підтримки прийняття рішень. Вона надає фахівцю об’єктивну пораду щодо прийняття рішень, що базується на даних, прискорює процес аналізу знімків та дозволяє звернути увагу на потенційно уражені ділянки, що є особливо критичним для своєчасного виявлення та лікування пневмонії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. World Pneumonia Day 2023 [Electronic resource]. – Access mode : https://stoppneumonia.org/latest/world-pneumonia-day-2023/.

2. About Pneumonia [Electronic resource]. – Access mode : https://www.cdc.gov/pneumonia/about/index.html.

3. Pneumonia [Electronic resource]. – Access mode : https://www.pfizer.com/disease-and-conditions/pneumonia.

4. Pneumonia [Electronic resource]. – Access mode : https://www.nhs.uk/conditions/pneumonia/.

5. Pneumonia [Electronic resource]. – Access mode : https://www.hopkinsmedicine.org/health/conditions-and-diseases/pneumonia.

6. Viral Pneumonia: Symptoms, Risk Factors, and More [Electronic resource]. – Access mode : https://www.healthline.com/health/viral-pneumonia.

7. How Fungal Pneumonia Impacts Your Health [Electronic resource]. – Access mode : https://www.verywellhealth.com/fungal-pneumonia-5179190.

8. Aspiration pneumonia [Electronic resource]. – Access mode : https://www.pennmedicine.org/conditions/aspiration-pneumonia.

9. Pneumonia [Electronic resource]. – Access mode : https://www.mayoclinic.org/diseases-conditions/pneumonia/symptoms-causes/syc-20354204.

10. Субботін, С. О. Нейронні мережі [Текст] : теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с..

11. Residual neural network [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Residual_neural_network.

12. Ensemble [Electronic resource]. – Access mode : https://www.ultralytics.com/glossary/ensemble.

13. The Power of ViT, Vision Transformers for Image Recognition [Electronic resource]. – Access mode : https://visionplatform.ai/vision-transformers-vit/.

14. Image classification using Support Vector Machine (SVM) in Python [Electronic resource]. – Access mode : https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/.

15. Alex Krizhevsky. ImageNet Classification with Deep Convolutional Neural Networks [Electronic resource] / Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton // NIPS. – 2012. – Access mode : https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

16. AlexNet [Electronic resource] : The First CNN to win Image Net. – Access mode : https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/.

17. AlexNet Model [Electronic resource]. – Access mode : https://wiki.imindlabs.com.au/ds/dl/1_models/2-cnn/3_alexnet/.

18. Karen Simonyan. Very Deep Convolutional Networds for Large-Scale Image Recognition [Electronic resource] / Karen Simonyan, Andrew Zisserman // arXiv. – 2015. – Access mode : https://arxiv.org/abs/1409.1556.

19. VGG16 – Convolutional Network for Classification and Detection [Electronic resource]. – Access mode : https://neurohive.io/en/popular-networks/vgg16/.

20. Introduction to VGG16 [Electronic resource]. – Access mode : https://wiki.imindlabs.com.au/ds/dl/1_models/2-cnn/4_vgg16/.

21. Anthony Owen. A Comparative Study of VGG16 and VGG19 Architectures for MRI-Based Brain Tumor Classification [Electronic resource] / Anthony Owen, Alex Campbell, Micheal Benson // ResearchGate. – 2025. – Access mode : https://www.researchgate.net/publication/393448611_A_Comparative_Study_of_VGG16_and_VGG19_Architectures_for_MRI-Based_Brain_Tumor_Classification.

22. VGG-19-Architecture [Electronic resource]. – Access mode : https://www.researchgate.net/figure/VGG-19-Architecture-39-VGG-19-has-16-convolution-layers-grouped-into-5-blocks-After_fig5_359771670.

23. Deep Residual Learning for Image Recognition [Electronic resource] / Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun // arXiv. – 2015. – Access mode : https://arxiv.org/abs/1512.03385.

24. ResNet-50 [Electronic resource] : The Basics and a Quick Tutorial. – Access mode : https://datagen.tech/guides/computer-vision/resnet-50/#.

25. ResNet-50 [Electronic resource]. – Access mode : https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758.

26. Densely Connected Convolutional [Electronic resource] / Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger // arXiv. – 2018. – Access mode : https://arxiv.org/abs/1608.06993.

27. DenseNet Explained [Electronic resource]. – Access mode : https://www.geeksforgeeks.org/computer-vision/densenet-explained/.

28. DenseNet [Electronic resource]. – Access mode : https://vlgiitr.github.io/papers_we_read/summaries/densenet.html.

29. MobileNet [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/MobileNet.

30. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Electronic resource] / Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam // arXiv. – 2017. – Access mode : https://arxiv.org/abs/1704.04861.

31. MobileNet Architecture [Electronic resource]. – Access mode : https://danghoangnhan.github.io/MobileNetArchitecture/.

32. Chest X-Ray Images (Pneumonia) [Electronic resource]. – Access mode : https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data.

33. Visual Studio Code [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Visual_Studio_Code/.

34. How Can Data Scientists Leverage The Power of GPU Jupyter Notebooks To Accelerate Deep Learning Tasks? [Electronic resource]. – Access mode : https://www.e2enetworks.com/blog/how-can-data-scientists-leverage-the-power-of-gpu-jupyter-notebooks-to-accelerate-deep-learning-tasks#.

35. Камінський, Д. О. Метод діагностування випадків пневмонії на основі компонентної структури моделі та алгоритмів глибокого навчання [Електрон. ресурс] / Д. О. Камінський, В. М. Льовкін // Науковий вісник Ужгородського університету. Серія «Математика і інформатика». – 2026. – Т. 48. – № 1. – Препринт.

36. Камінський, Д. О. Розробка прикладної програми розпізнавання пневмонії на основі фреймворку та компонентної моделі для створення інтегрованого середовища медичної системи [Електрон. ресурс] / Д. О. Камінський, В. М. Льовкін // Вісник Черкаського державного технологічного університету. – 2025. – Т. 30. – № 4. – Препринт.

37. Камінський, Д. О. Розробка прикладної програми визначення випадків пневмонії на основі засобів машинного навчання та вебфрейморку [Текст] / Д. О. Камінський, В. М. Льовкін // Міжнародна науково-практична конференція з проблем використання інформаційних технологій в освіті, науці та промисловості. 5 грудня 2025 р., Дніпро, Україна. – Дніпро : Національний технічний університет "Дніпровська політехніка", 2025. – Препринт.

ДОДАТОК А

Технічне завдання

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Дослідження та програмна реалізація методів визначення випадків пневмонії» згідно з наказом № 447 від 30 вересня 2025 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Програмне забезпечення призначене для використання як система підтримки прийняття рішень для лікарів-радіологів та терапевтів. Система дозволяє лікарям керувати медичними випадками, завантажувати рентгенівські знімки та отримувати класифікацію від ансамблю ШНМ як допоміжний інструмент.

А.3 Вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно виконувати наступні функціональні вимоги:

а) вимоги пацієнта:

1) реєстрація в системі;
2) авторизація в системі;
3) перегляд списку медичних випадків та їх статусів;
4) перегляд власних рентгенівських знімків у межах медичного випадку;
5) перегляд остаточних діагнозів та висновків, наданих лікарем;
6) вихід з особистого кабінету.

б) вимоги лікаря:

1) реєстрація в системі;
2) авторизація в системі;
3) перегляд списку медичних випадків;
4) пошук медичних випадків за пацієнтом;
5) створення нового медичного випадку для пацієнта;
6) редагування інформації про медичний випадок;
7) завантаження рентгенівських знімків до медичного випадку;
8) перегляд завантажених знімків та результатів їх автоматичного аналізу;
9) формування власних анотацій до окремих знімків;
10) встановлення та редагування остаточного діагнозу для знімка;
11) вихід з особистого кабінету.

в) вимоги адміністратора:

1) авторизація в системі;
2) перегляд статистики всієї системи;
3) перегляд списку всіх користувачів системи;
4) редагування даних профілів користувачів;
5) ініціювати перенавчання моделей ШНМ;
6) перегляд списку всіх збережених чекпоінтів моделей ШНМ;
7) управління чекпоінтами моделей ШНМ;
8) перегляд журналу аудиту;
9) перегляд статистики порівняннь передбачень моделі з рішеннями лікарів;
10) вихід з панелі адміністратора.

А.3.2 Вимоги до надійності

Програмне забезпечення повинно відповідати таким вимогам безпеки:

– підтримка функцій захисту від несанкціонованого доступу (доступ

до різних сторінок для лікарів, пацієнтів та адміністратора);

– обробка помилкових дій користувача з відповідними повідомленнями (валідація форм користувачів, перехід на неіснуючі сторінки);

– обробка виняткових ситуацій (наприклад, помилки при роботі з базою даних).

А.3.3 Умови експлуатації

Для розміщення розробленої програми у мережі Інтернет необхідно виділити вебвузол і встановити на ньому:

– сервер із підтримкою Python та Django, DRF;

– бібліотеку TensorFlow для реалізації нейронних мереж;

– базу даних для зберігання інформації про користувачів, знімки та діагнози;

– доступ до мережі Інтернет для роботи з інтерфейсом та обробки запитів.

А.3.4 Вимоги до складу та параметрів технічних засобів

Для використання розробленої системи необхідні:

– процессор Intel Core i7 або еквівалент;

– мінімум 8 Гб, рекомендовано 16 Гб або більше;

– відеокарта: рекомендовано відеокарту NVIDIA з підтримкою CUDA;

– операційна система: Linux Ubuntu 20.04 LTS/22.04 LTS або WSL 2.

А.4 Порядок контролю та приймання

Виконання дипломної кваліфікаційної роботи магістра має бути закінчено у дванадцятитижневий термін з розподіленням за етапами, що

визначено завданням на роботу у відповідному календарному плані. Після відбувається захист перед екзаменаційною комісією.

ДОДАТОК Б

Текст програми

```
import os
import cv2
import time
import json
import logging
import warnings

import numpy             as np
import tensorflow        as tf
import seaborn           as sns
import matplotlib.pyplot as plt

from math       import ceil
from pathlib    import Path
from datetime   import datetime
from typing     import Dict, Optional, Tuple, Any
from PIL        import ImageFile

from sklearn.metrics            import confusion_matrix
from sklearn.utils.class_weight import compute_class_weight

import tensorflow.keras.backend as K
from tensorflow.keras.preprocessing.image   import ImageDataGenerator
from  tensorflow.keras.callbacks                 import  EarlyStopping,
ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.optimizers            import Adam, SGD, AdamW
from tensorflow.keras.optimizers.schedules  import CosineDecay
from tensorflow.keras.models                import Sequential, Model
from tensorflow.keras.layers                import (
    Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer,
    BatchNormalization, GlobalAveragePooling2D,
    Input, LeakyReLU, Add
)

try:
    from .progress_state import update_progress
except ImportError:
    from progress_state  import update_progress

#region Setup env
BASE_DIR = Path(__file__).resolve().parent

DATA_DIR = BASE_DIR / "datasets" / "data_pneumonia"

OUTPUT_DIR = BASE_DIR / "outputs"
CHECKPOINT_DIR  = OUTPUT_DIR / "checkpoints"
GRADCAM_DIR     = OUTPUT_DIR / "gradcam"
RESULTS_DIR     = OUTPUT_DIR / "results"

IMAGE_SIZE = (150, 150)

ImageFile.LOAD_TRUNCATED_IMAGES = True
warnings.filterwarnings("ignore", category=UserWarning)

# Configure logging
logging.basicConfig(
```

```
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler(BASE_DIR / 'classifier.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

#endregion



class Classifier:

    AVAILABLE_MODELS = [
        'OwnV3',    'AlexNet',    'VGG16',    'VGG19',    'ResNet50',
'DenseNet121', 'MobileNetV2',
    ]

    #region Initialize classifier with configuration

    def __init__(self,
                 model_name: str = 'OwnV3',
                 img_size: Tuple[int, int] = IMAGE_SIZE,
                 data_dir: Path = DATA_DIR) -> None:
        """
        Initialize the Classifier with specified configuration.

        :param model_name: Name of the model architecture to use. Must
be in Classifier.AVAILABLE_MODELS.
        :type model_name: str
        :param img_size: Target image dimensions as (width, height).
        :type img_size: Tuple[int, int]
        :param data_dir: Path to the root data directory. Must contain
'train/' and 'test/' subdirectories.
        :type data_dir: str
        """
        if model_name not in self.AVAILABLE_MODELS:
            logger.error(f"[INIT]          Unsupported          model
'{self.model_name}'. Available: {self.AVAILABLE_MODELS}")
            raise ValueError(f"Unsupported model '{self.model_name}'.
Available: {self.AVAILABLE_MODELS}")

        logger.info(f"[INIT]  Initializing  Classifier  with  model:
{model_name}, size: {img_size}")

        self.model_name = model_name
        self.img_size = img_size
        self.img_shape = (*img_size, 3)
        self.data_dir = data_dir


        self.train_dir,       self.val_dir,       self.test_dir      =
self._detect_folders()
```

```
        self.class_labels,    self.num_classes,    self.class_mode    =
self._detect_classes()

        self.model = None
        self.history = None
        self.train_generator = None
        self.val_generator = None
        self.test_generator = None
        self.best_checkpoint_path = None

        self._setup_environment()

    def _detect_folders(self) -> Tuple[str, Optional[str], str]:
        """
        Automatically detect train, validation, and test directories.

        :returns: (train_dir, val_dir, test_dir) paths.
        :rtype: Tuple[str, Optional[str], str]
        """
        if not self.data_dir:
            logger.error("[DATA] No data directory specified.")
            raise ValueError("No data directory specified")

        data_path = Path(self.data_dir)

        train_dir = data_path / 'train'
        test_dir = data_path / 'test'
        val_dir = data_path / 'val'

        if train_dir.exists() and test_dir.exists():
            val_dir = val_dir if val_dir.exists() else None
            logger.info("[DATA] Detected folders:")
            logger.info(f"  Train: {train_dir}")
            logger.info(f"  Test: {test_dir}")
            logger.info(f"   Val: {val_dir or 'None (will split from
train)'}")
            return str(train_dir), str(val_dir), str(test_dir)

        logger.error(f"[DATA] Expected directory structure not found in
{self.data_dir}")
        raise ValueError(
            f"Expected    directory    structure    not    found    in
{self.data_dir}. "
            "Expected: train/ and test/ directories (val/ is optional)"
        )

    def _detect_classes(self) -> Tuple[Dict[int, str], int, str]:
        """
        Auto-detect class labels from the data directory structure.

        :returns:   (Class   index   mapping,   number   of   classes,
classification mode).
        :rtype: Tuple[Dict[int, str], int, str]
        """
```

```
        classes = sorted([d.name for d in
Path(self.train_dir).iterdir() if d.is_dir()])
        num_classes = len(classes)
        class_mode = 'binary' if num_classes == 2 else 'categorical'

        logger.info(f"[DATA] Found {num_classes} valid classes:
{classes}")
        if num_classes == 2:
            logger.info("  Binary classification mode detected.")
        elif num_classes > 10:
            logger.warning(f"[DATA] Large number of classes
({num_classes}) detected. Consider if this is intentional.")
        else:
            logger.info(f"    Multi-class classification mode with
{num_classes} classes.")

        return {idx: cls for idx, cls in enumerate(classes)},
num_classes, class_mode

    def _setup_environment(self) -> None:
        """Configure GPU settings and optimize TensorFlow
environment."""
        physical_devices = tf.config.list_physical_devices('GPU')
        if physical_devices:
            try:
                device_name = physical_devices[0].name if
hasattr(physical_devices[0], 'name') else physical_devices[0]

tf.config.experimental.set_memory_growth(physical_devices[0], True)
                logger.info(f"[ENV] GPU memory growth enabled for
device: {device_name}")
            except RuntimeError as e:
                logger.warning(f"[ENV] Could not set GPU memory growth:
{e}")
        else:
            logger.info("[ENV] No GPU devices found, using CPU.")

        # Suppress TensorFlow info/warning messages
        # 0=all, 1=no INFO, 2=no INFO/WARNING, 3=no INFO/WARNING/ERROR
        os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
        logger.info("[ENV] Suppressing TensorFlow internal INFO/WARNING
messages (level 2).")
        return

    #endregion


    #region Setup data pipeline

    def _setup_data_generators(self, batch_size: int = 16) -> Tuple[int,
int, int]:
        """
        Setup data generators for training, validation, and testing with
augmentation.

        :param batch_size: Number of samples per batch.
```

```
        :type batch_size: int
        :returns: (steps_per_epoch, validation_steps, test_steps).
        :rtype: Tuple[int, int, int]
        """
        logger.info(f"[DATA] Setting up data generators with batch size:
{batch_size}")

        # Default augmentation config
        augmentation_config = {
            'rotation_range': 20,
            'brightness_range': (0.8, 1.2),
            'width_shift_range': 0.015,
            'height_shift_range': 0.015,
            'shear_range': 20,
            'horizontal_flip': True,
        }

        # Helper to create a generator
        def     make_generator(datagen,     directory,     subset=None,
shuffle=False):
            return datagen.flow_from_directory(
                directory,
                target_size=self.img_size,
                batch_size=batch_size,
                class_mode=self.class_mode,
                subset=subset,
                seed=42,
                shuffle=shuffle,
            )

        # Train/val setup
        if self.val_dir and os.path.isdir(self.val_dir):
            logger.info("  Using explicit validation directory.")
            train_datagen     =     ImageDataGenerator(rescale=1./255,
**augmentation_config)
            val_datagen = ImageDataGenerator(rescale=1./255)
            self.train_generator    =    make_generator(train_datagen,
self.train_dir, shuffle=True)
            self.val_generator      =      make_generator(val_datagen,
self.val_dir)
        else:
            val_split = 0.2
            logger.info(f"   Splitting  training  data  into  train/val
(ratio {1-val_split:.1f}/{val_split:.1f}).")
            train_datagen     =     ImageDataGenerator(rescale=1./255,
validation_split=val_split, **augmentation_config)
            self.train_generator    =    make_generator(train_datagen,
self.train_dir, subset="training", shuffle=True)
            self.val_generator     =     make_generator(train_datagen,
self.train_dir, subset="validation")

        # Always set up test generator
        test_datagen = ImageDataGenerator(rescale=1./255)
        self.test_generator       =       make_generator(test_datagen,
self.test_dir)
```

```
        # Compare class indices
        generated_class_labels = {v: k for k, v in
self.train_generator.class_indices.items()}
        if generated_class_labels != self.class_labels:
            logger.error("[DATA] Class labels mismatch between detected
and generator.")
            raise ValueError(f"Class labels from generator do not match
detected labels: {generated_class_labels} vs {self.class_labels}")

        # Calculate steps for training
        steps_per_epoch = ceil(self.train_generator.samples /
batch_size)
        validation_steps = ceil(self.val_generator.samples /
batch_size)
        test_steps = ceil(self.test_generator.samples / batch_size)

        # Check for class imbalance
        self._check_class_imbalance()

        logger.info("[DATA] Data generators setup complete:")
        logger.info(f"                    Training          samples:
{self.train_generator.samples} ({steps_per_epoch} steps)")
        logger.info(f"                  Validation          samples:
{self.val_generator.samples} ({validation_steps} steps)")
        logger.info(f"  Test samples: {self.test_generator.samples}
({test_steps} steps)")

        return steps_per_epoch, validation_steps, test_steps

    def _check_class_imbalance(self) -> None:
        """Check for class imbalance and log warnings if found."""
        # Get class distribution
        class_counts = {}
        counts = np.bincount(self.train_generator.classes)
        for class_idx, class_name in self.class_labels.items():
            class_counts[class_name] = counts[class_idx]

        logger.info(f"[DATA] Training class distribution:
{class_counts}")

        # Check for imbalance
        counts = list(class_counts.values())
        max_count, min_count = max(counts), min(counts)
        imbalance_ratio = max_count / min_count if min_count > 0 else
float('inf')

        if imbalance_ratio > 3:
            logger.warning(f"[DATA] Significant class imbalance
detected! Ratio: {imbalance_ratio:.2f}:1")
        else:
            logger.info(f"[DATA] Class balance ratio:
{imbalance_ratio:.2f}:1")

    #endregion
```

```
    #region Get CNN model

    def load_model(self, checkpoint_path: str) -> None:
        """Load a saved model from disk."""
        self.model = tf.keras.models.load_model(checkpoint_path)
        logger.info(f"[MODEL] Model loaded from {checkpoint_path}")

    def _build_model(self) -> bool:
        """
        Build the specified model architecture.

        :returns: True if model built successfully, False otherwise.
        :rtype: bool
        """
        try:
            def add_output_layer(model):
                """Attach   final   classification   layer   based   on
num_classes."""
                if self.num_classes == 2:
                    model.add(Dense(1,           activation='sigmoid',
name='output'))
                    logger.info("  Added sigmoid output layer for binary
classification")
                else:
                    model.add(Dense(self.num_classes,
activation='softmax', name='output'))
                    logger.info(f"   Added  softmax  output  layer  for
{self.num_classes}-class classification")
                return model


            def build_ownv3():
                def  residual_block(x,  filters,  kernel_size=(3,  3),
stride=1):
                    # Main path
                    y  =  Conv2D(filters,  kernel_size,  strides=stride,
padding='same', kernel_regularizer='l2')(x)
                    y = BatchNormalization()(y)
                    y = LeakyReLU(alpha=0.01)(y)

                    y  =  Conv2D(filters,  kernel_size,  padding='same',
kernel_regularizer='l2')(y)
                    y = BatchNormalization()(y)

                    if stride != 1 or x.shape[-1] != filters:
                        shortcut    =    Conv2D(filters,    (1,    1),
strides=stride, padding='same')(x)
                    else:
                        shortcut = x

                    # Add shortcut to the main path
                    out = Add()([y, shortcut])
                    out = LeakyReLU(alpha=0.01)(out)
                    return out

                # --- Input & Stem ---
```

```
                inputs = Input(shape=self.img_shape)
                x     =     Conv2D(32,     (7,     7),     strides=2,
padding='same')(inputs)
                x = BatchNormalization()(x)
                x = LeakyReLU(alpha=0.01)(x)
                x = MaxPooling2D((3, 3), strides=2, padding='same')(x)

                # --- Residual Blocks ---
                x = residual_block(x, filters=32)
                x = residual_block(x, filters=64, stride=2)
                x = residual_block(x, filters=128, stride=2)
                x = residual_block(x, filters=256, stride=2)

                # --- Classifier Head ---
                x = GlobalAveragePooling2D()(x)
                x = Dense(128, activation='relu')(x)
                x = Dropout(0.6)(x)

                outputs      =      Dense(1,      activation='sigmoid',
name='output')(x) if self.num_classes == 2 \
                    else Dense(self.num_classes, activation='softmax',
name='output')(x)

                return Model(inputs, outputs, name='OwnV3')

            def build_alexnet():
                model = Sequential([
                    Conv2D(96,     (11,     11),     strides=(4,     4),
activation='relu', input_shape=self.img_shape),
                    MaxPooling2D((3, 3), strides=(2, 2)),
                    BatchNormalization(),
                    Conv2D(256,     (5,     5),     activation='relu',
padding='same'),
                    MaxPooling2D((3, 3), strides=(2, 2)),
                    BatchNormalization(),
                    Conv2D(384,     (3,     3),     activation='relu',
padding='same'),
                    Conv2D(384,     (3,     3),     activation='relu',
padding='same'),
                    Conv2D(256,     (3,     3),     activation='relu',
padding='same'),
                    MaxPooling2D((3, 3), strides=(2, 2)),
                    BatchNormalization(),
                    Flatten(),
                    Dense(4096, activation='relu'),
                    Dropout(0.5),
                    Dense(4096, activation='relu'),
                    Dropout(0.5),
                ])
                return add_output_layer(model)

            def build_transfer(model_name):
                base_models = {
                    'VGG16': tf.keras.applications.VGG16,
                    'VGG19': tf.keras.applications.VGG19,
                    'ResNet50': tf.keras.applications.ResNet50,
```

```
                    'DenseNet121': tf.keras.applications.DenseNet121,
                    'MobileNetV2': tf.keras.applications.MobileNetV2,
                }
                base = base_models[model_name](weights="imagenet",
include_top=False, input_shape=self.img_shape)
                base.trainable = False
                model = Sequential([
                    base,
                    GlobalAveragePooling2D(),
                    Dense(128, activation='relu'),
                    Dropout(0.5),
                ])
                return add_output_layer(model)

            # Map model name - builder
            builders = {
                "OwnV3": build_ownv3,
                "AlexNet": build_alexnet,
            }

            # Build chosen model
            if self.model_name in builders:
                logger.info(f"[MODEL] Building {self.model_name}
architecture.")
                self.model = builders[self.model_name]()
            else:
                logger.info(f"[MODEL] Building transfer learning model:
{self.model_name}")
                self.model = build_transfer(self.model_name)

            # Count parameters
            total_params = self.model.count_params()
            trainable_params = sum([K.count_params(w) for w in
self.model.trainable_weights])

            logger.info("[MODEL] Model built successfully:")
            logger.info(f"  Total parameters: {total_params:,}")
            logger.info(f"                Trainable        parameters:
{trainable_params:,}")
            logger.info(f"  Non-trainable parameters: {total_params -
trainable_params:,}")

            return True

        except Exception as e:
            logger.error(f"[MODEL] Error building model: {e}")
            return False

    #endregion


    #region CNN training/predict

    def train(self, epochs=50, batch_size=32, learning_rate=0.001,
optimizer='adam',
```

```
              early_stopping=True,                      save_best=True,
checkpoint_dir=CHECKPOINT_DIR, verbose=1) -> Dict[str, Any]:
        """
        Train the model.

        :param epochs: Number of epochs to train for.
        :type epochs: int
        :param batch_size: Samples per batch.
        :type batch_size: int
        :param learning_rate: Optimizer learning rate.
        :type learning_rate: float
        :param optimizer: Name of the optimizer ('adam', 'sgd', etc.).
        :type optimizer: str
        :param early_stopping: Enable early stopping callback.
        :type early_stopping: bool
        :param save_best: Enable model checkpointing for the best
weights.
        :type save_best: bool
        :param checkpoint_dir: Directory to save model checkpoints.
        :type checkpoint_dir: str
        :param verbose: Verbosity level for training (0=silent,
1=progress bar, 2=one line per epoch).
        :type verbose: int
        :returns: Dictionary of final evaluation results on the test
set.
        :rtype: Dict[str, Any]
        """

        start_time = time.time()
        logger.info(f"[TRAIN]      Starting      model      training:
{self.model_name}")
        logger.info(f"   Config: Epochs={epochs}, Batch={batch_size},
Optimizer={optimizer}, LR={learning_rate}")

        def get_loss_and_metrics():
            """Return loss and metrics based on classification type."""
            if self.num_classes == 2:
                return 'binary_crossentropy', [
                    'accuracy',
                    tf.keras.metrics.Precision(name='precision'),
                    tf.keras.metrics.Recall(name='recall'),
                    tf.keras.metrics.AUC(name='auc'),
                ]
            return 'categorical_crossentropy', [
                'accuracy',
                tf.keras.metrics.Precision(name='precision',
average='weighted'),
                tf.keras.metrics.Recall(name='recall',
average='weighted'),
                tf.keras.metrics.F1Score(name='f1_score',
average='weighted'),
            ]

        def get_class_weights():
            """Balanced class weights for imbalanced datasets."""
            labels = np.array(self.train_generator.classes)
```

```
            weights         =         compute_class_weight('balanced',
classes=np.unique(labels), y=labels)
            return dict(enumerate(weights))

        def get_callbacks():
            """Create training callbacks list."""
            callbacks = []
            callbacks.append(ReduceLROnPlateau(monitor='val_auc',
factor=0.5,
                                               patience=5,   min_lr=1e-8,
verbose=verbose, mode='max'))
            if early_stopping:
                patience = 12 if epochs >= 30 else max(3, epochs // 3)
                callbacks.append(EarlyStopping(monitor='val_auc',
patience=patience,
                                               restore_best_weights=True,
verbose=verbose, mode='max'))
            if save_best:
                timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")
                unique_checkpoint_dir  =  os.path.join(checkpoint_dir,
f"{self.model_name}_{timestamp}")
                os.makedirs(unique_checkpoint_dir, exist_ok=True)
                static_filepath  =  os.path.join(unique_checkpoint_dir,
"best_model.hdf5")

                self.best_checkpoint_path = None

                callbacks.append(PathTrackingModelCheckpoint(
                    classifier_instance=self,  #  Pass  'self'  (the
classifier instance)
                    filepath=static_filepath,
                    monitor='val_accuracy',       save_best_only=True,
save_weights_only=False, verbose=verbose, mode='max'))
            return callbacks

        def compile_model():
            """Compile  model  with  chosen  optimizer,  loss,  and
metrics."""
            total_steps = steps_per_epoch * epochs
            optimizers = {
                'adam':  Adam(learning_rate=learning_rate,  beta_1=0.9,
beta_2=0.999, epsilon=1e-7),
                'adamw':            AdamW(learning_rate=learning_rate,
weight_decay=0.01),
                'sgd':    SGD(learning_rate=CosineDecay(learning_rate,
total_steps), momentum=0.9, nesterov=True),
            }
            loss_fn, metrics = get_loss_and_metrics()
            self.model.compile(optimizer=optimizers.get(optimizer,
Adam(learning_rate=learning_rate)),
                               loss=loss_fn, metrics=metrics)
            if verbose > 0:
                self.model.summary()
            logger.info(f"[TRAIN] Model compiled with loss={loss_fn},
optimizer={optimizer}")
```

```
        def evaluate_training():
            """Evaluate model after training and return summary
results."""
            elapsed = time.time() - start_time
            results_dict = dict(zip(self.model.metrics_names,

self.model.evaluate(self.test_generator, verbose=0)))
            results = {
                'loss': results_dict['loss'],
                'accuracy': results_dict['accuracy'],
                'training_time': elapsed,
                'epochs_trained': len(self.history.history['loss']),
                'classification_type': self.class_mode,
                'num_classes': self.num_classes,
                'model_name': self.model_name,
                'img_size': self.img_size,
                'batch_size': batch_size,
                'optimizer': optimizer,
                'learning_rate': learning_rate
            }
            for metric in ['precision', 'recall', 'auc', 'f1_score']:
                if metric in results_dict:
                    results[metric] = results_dict[metric]
            return results

        class TrainingProgressCallback(tf.keras.callbacks.Callback):
            """
            Custom callback that reports training progress to the shared
progress_state.
            Called automatically by Keras after each epoch.
            """
            def __init__(self, total_epochs: int):
                super().__init__()
                self.total_epochs = total_epochs
                self.start_time = None

            def on_train_begin(self, logs=None):
                self.start_time = time.time()

            def on_epoch_end(self, epoch, logs=None):
                logs = logs or {}
                elapsed = time.time() - self.start_time
                avg_epoch_time = elapsed / (epoch + 1)
                eta = avg_epoch_time * (self.total_epochs - (epoch +
1))
                epoch_data = {
                    "epoch": epoch + 1,
                    "total_epochs": self.total_epochs,
                    "elapsed": elapsed,
                    "eta": eta,
                    "status": "running",
                    "loss": float(logs.get("loss", 0.0)),
                    "val_loss": float(logs.get("val_loss", 0.0)),
                    "acc": float(logs.get("accuracy", 0.0)),
                    "val_acc": float(logs.get("val_accuracy", 0.0)),
                    "precision": float(logs.get("precision", 0.0)),
```

```
                    "recall": float(logs.get("recall", 0.0)),
                    "val_precision":   float(logs.get("val_precision",
0.0)),
                    "val_recall": float(logs.get("val_recall", 0.0)),
                    "auc": float(logs.get("auc", 0.0)),
                    "val_auc": float(logs.get("val_auc", 0.0)),
                    "f1_score": float(logs.get("f1_score", 0.0)),
                    "val_f1_score":     float(logs.get("val_f1_score",
0.0)),
                    "message": f"Epoch {epoch + 1}/{self.total_epochs}
completed"
                }
                update_progress(**epoch_data)


            def on_train_end(self, logs=None):
                total_time = time.time() - self.start_time
                update_progress(status="success",    message="Training
complete", elapsed=total_time)

        class PathTrackingModelCheckpoint(ModelCheckpoint):
            """
            A custom ModelCheckpoint to track the path of the best saved
model file.
            """
            def __init__(self, classifier_instance, *args, **kwargs):
                super().__init__(*args, **kwargs)
                self.classifier_instance = classifier_instance

            def on_epoch_end(self, epoch, logs=None):
                super().on_epoch_end(epoch, logs)

                if     self.save_best_only     and     self.best     ==
logs.get(self.monitor):
                    self.classifier_instance.best_checkpoint_path    =
self.filepath
                    logger.info(f"[CHECKPOINT]   Updated   best   path:
{self.filepath}")

        # 1. Setup
        steps_per_epoch,         val_steps,         test_steps         =
self._setup_data_generators(batch_size=batch_size)
        self._build_model()
        compile_model()

        # 2. Callbacks
        callbacks = get_callbacks()

callbacks.append(TrainingProgressCallback(total_epochs=epochs))

        # 3. Class weights
        class_weight_dict = get_class_weights()
        logger.info(f"[TRAIN]        Using        class        weights:
{class_weight_dict}")

        # 4. Train
```

```
        logger.info("[TRAIN] Entering training loop...")
        self.history = self.model.fit(
            self.train_generator,
            steps_per_epoch=steps_per_epoch,
            epochs=epochs,
            validation_data=self.val_generator,
            validation_steps=val_steps,
            class_weight=class_weight_dict,
            callbacks=callbacks,
            verbose=verbose
        )

        # 5. Evaluation
        results = evaluate_training()
        logger.info(f"[TRAIN]       Training       complete       in
{results['training_time']:.2f} seconds.")
        logger.info(f"[TRAIN]              Test              Results:
Loss={results.get('loss'):.4f},
Accuracy={results.get('accuracy'):.4f}")
        return results

    def   save_history(self,   file_path:   Path   =   RESULTS_DIR   /
"training_history.json") -> bool:
        """
        Saves the training history to a JSON file.

        :param file_path: Path to save the history JSON file.
        :type file_path: str
        :returns: True on success, False otherwise.
        :rtype: bool
        """
        if self.history is None:
            logger.warning("[HISTORY] No training history available to
save.")
            return False

        # Convert history to pure Python types
        history_dict = {"history": {k: [float(vv) for vv in v] for k, v
in self.history.history.items()}}

        try:
            os.makedirs(os.path.dirname(file_path), exist_ok=True)
            with open(file_path, 'w') as f:
                json.dump(history_dict, f, indent=4)
            logger.info(f"[HISTORY]   Training   history   saved   to
{file_path}")
            return True
        except Exception as e:
            logger.error(f"[HISTORY]  Error  saving  training  history:
{e}")
            return False

    def   load_history(self,   file_path:   Path   =   RESULTS_DIR   /
"training_history.json") -> bool:
        """
        Loads training history from a JSON file.
```

```
        :param file_path: Path to load the history JSON file.
        :type file_path: str
        :returns: True on success, False otherwise.
        :rtype: bool
        """
        if not os.path.exists(file_path):
            logger.warning(f"[HISTORY] History file does not exist:
{file_path}")
            return None

        class History:
            def __init__(self, history_dict):
                self.history = history_dict

        try:
            with open(file_path, 'r') as f:
                data = json.load(f)
            self.history = History(data["history"])
            logger.info("[HISTORY] History loaded succesfully")
            return True
        except Exception as e:
            logger.error(f"[HISTORY] Error loading training history:
{e}")
            return False

    def predict(self, image_path: str) -> Tuple[str, float, Dict[str,
float]]:
        """
        Predict the class for a single input image.

        :param image_path: Path to the image file to classify.
        :type image_path: str
        :returns: (Predicted class name, confidence score, dictionary
of all class probabilities).
        :rtype: Tuple[str, float, Dict[str, float]]
        """
        logger.info(f"[PREDICT] Making prediction for image:
{image_path}")

        if self.model is None:
            logger.error("[PREDICT] Model not trained or loaded.")
            raise ValueError("Model not trained or loaded")
        if not os.path.exists(image_path):
            logger.error(f"[PREDICT] Image file not found:
{image_path}")
            raise FileNotFoundError(f"Image file not found:
{image_path}")

        try:
            img = tf.keras.preprocessing.image.load_img(image_path,
target_size=self.img_size)
            img_array = tf.keras.preprocessing.image.img_to_array(img)
            img_array = np.expand_dims(img_array / 255.0, axis=0)
        except Exception as e:
```

```
            logger.error(f"[PREDICT] Error loading/preprocessing image:
{e}")
            raise

        probs = self.model.predict(img_array, verbose=0)[0]

        if self.num_classes == 2:
            probs = np.array([1 - probs[0], probs[0]])  # force shape
(2,)

        pred_idx = int(np.argmax(probs))
        predicted_class = self.class_labels[pred_idx]
        confidence = float(probs[pred_idx])
        probabilities   =   {label:   float(p)   for   label,   p   in
zip(self.class_labels, probs)}

        logger.info(f"[PREDICT]     Result:     Predicted     class:
{predicted_class}, Confidence: {confidence:.2%}")
        return predicted_class, confidence, probabilities

    def evaluate_model(self, return_details: bool = False):
        """
        Evaluate the model on the validation set.

        :param return_details: If True, returns raw predictions and
labels alongside metrics.
        :type return_details: bool
        :returns:  Dictionary  containing  evaluation  metrics  (and
optional details).
        :rtype: Dict
        """
        if self.val_generator is None:
            logger.warning("[PREDICT] val_generator is None. Preparing
data...")
            self._setup_data_generators()

        # Standard Keras metrics (loss, acc, precision, recall, etc.)
        keras_results    =    self.model.evaluate(self.val_generator,
verbose=0, return_dict=True)

        # Predictions for confusion-matrix-based metrics
        probs = self.model.predict(self.val_generator, verbose=0)
        if probs.shape[1] == 1:  # binary -> convert to (n_samples, 2)
            probs = np.hstack([1 - probs, probs])

        preds = np.argmax(probs, axis=1)
        confs = np.max(probs, axis=1)
        labels = self.val_generator.classes

        sensitivity = specificity = None
        if self.num_classes == 2:
            tn, fp, fn, tp = confusion_matrix(labels, preds).ravel()
            sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0.0
            specificity = tn / (tn + fp) if (tn + fp) > 0 else 0.0

        # Merge results
```

```
        summary = {
            "model": self.model_name,
            "num_classes": self.num_classes,
            "loss": keras_results.get("loss"),
            "accuracy": keras_results.get("accuracy"),
            "precision": keras_results.get("precision"),
            "recall": keras_results.get("recall"),
            "auc": keras_results.get("auc"),
            "f1_score": keras_results.get("f1_score"),
            "sensitivity": sensitivity,
            "specificity": specificity,
        }

        if return_details:
            summary.update({
                "class_labels": self.class_labels,
                "probs": probs,
                "pred_indices": preds,
                "confidences": confs,
                "true_indices": labels,
            })

        logger.info("[PREDICT] Model evaluation on validation set
complete.")
        return summary

    #endregion


    #region Generating plotts/heatmaps

    def generate_gradcam_heatmap(self,
                                 image_path: str,
                                 conv_layer_name: Optional[str] = None,
                                 alpha: float = 0.5,
                                 output_dir: Path = GRADCAM_DIR) ->
Tuple[np.ndarray, np.ndarray, np.ndarray]:
        """
        Generate Grad-CAM heatmap for model interpretability on a single
image.

        :param image_path: Path to the input image file.
        :type image_path: str
        :param conv_layer_name: Name of the convolutional layer to
analyze. Uses the last Conv2D layer if None.
        :type conv_layer_name: Optional[str]
        :param alpha: Transparency factor for heatmap overlay (0.0-
1.0).
        :type alpha: float
        :param output_dir: Directory to save the generated heatmap
image.
        :type output_dir: str
        :returns: (Original image with heatmap overlay, raw heatmap
visualization, original preprocessed image).
        :rtype: Tuple[np.ndarray, np.ndarray, np.ndarray]
        :raises ValueError: If the model is not trained/loaded.
```

```
        :raises FileNotFoundError: If the image file doesn't exist.
        """
        logger.info(f"[GRADCAM]   Generating   Grad-CAM   for   image:
{image_path}")
        logger.info(f"   Using  model:  {self.model_name},  Image  size:
{self.img_size}")

        if self.model is None:
            logger.error("[GRADCAM]  Model  has  not  been  trained  or
loaded.")
            raise  ValueError("Model  has  not  been  trained  or  loaded
yet.")
        if not os.path.exists(image_path):
            logger.error(f"[GRADCAM]    Image    file    not    found:
{image_path}")
            raise    FileNotFoundError(f"Image    file    not    found:
{image_path}")

        os.makedirs(output_dir, exist_ok=True)

        try:
            img   =   tf.keras.preprocessing.image.load_img(image_path,
target_size=self.img_size)
            img_array = tf.keras.preprocessing.image.img_to_array(img)
            img_array = np.expand_dims(img_array / 255.0, axis=0)
        except Exception as e:
            logger.error(f"[GRADCAM] Error loading/preprocessing image:
{e}")
            raise

        # Find the last convolutional layer if not specified
        if conv_layer_name is None:
            conv_layers = []
            for layer in self.model.layers:
                if isinstance(layer, Conv2D):
                    conv_layers.append(layer.name)

            if not conv_layers:
                logger.warning("[GRADCAM]  Could  not  find  any  Conv2D
layers in the model.")
                return None, None, None

            conv_layer_name = conv_layers[-1]  # Use the last conv layer
            logger.info(f"        Analyzing    convolutional    layer:
{conv_layer_name}")

        try:
            # Create Grad-CAM model
            grad_model = Model(
                inputs=self.model.inputs,
                outputs=[self.model.get_layer(conv_layer_name).output,
self.model.output]
            )

            # Compute gradients
            with tf.GradientTape() as tape:
```

```
                conv_outputs, predictions = grad_model(img_array)
                if self.num_classes == 2:
                    # Binary classification - use the sigmoid output
directly
                    class_output = predictions[0][0]
                else:
                    # Multi-class classification - use the highest
probability class
                    predicted_class_idx = tf.argmax(predictions[0])
                    class_output = predictions[0][predicted_class_idx]

            # Calculate gradients of loss with respect to conv layer
output
            grads = tape.gradient(class_output, conv_outputs)

            if grads is None:
                logger.error(f"[GRADCAM] Gradient is None for layer
'{conv_layer_name}'. Check layer existence.")
                raise ValueError(
                    f"Gradient is None for layer '{conv_layer_name}'. "
                    f"Check that the layer exists and is
differentiable."
                )

            # Pool gradients over spatial dimensions (Global Average
Pooling of gradients)
            pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

            # Weight the feature maps by the gradients
            conv_outputs = conv_outputs[0]  # Remove batch dimension
            heatmap      =      tf.reduce_sum(tf.multiply(pooled_grads,
conv_outputs), axis=-1)

            # Apply ReLU to focus on features that positively influence
the prediction
            heatmap = tf.nn.relu(heatmap)

            # Normalize heatmap to [0, 1]
            heatmap_max = tf.reduce_max(heatmap)
            if heatmap_max > 0:
                heatmap = heatmap / heatmap_max

            # Convert to numpy and resize to original image dimensions
            heatmap = heatmap.numpy()
            heatmap = cv2.resize(heatmap, self.img_size)

            # Convert to colormap for visualization
            heatmap_colored = np.uint8(255 * heatmap)
            heatmap_colored    =    cv2.applyColorMap(heatmap_colored,
cv2.COLORMAP_JET)

            # Load original image and superimpose heatmap
            original_img = cv2.imread(image_path)
            if original_img is None:
                raise   ValueError(f"Could   not   load   image   from
{image_path}")
```

```
            original_img = cv2.resize(original_img, self.img_size)
            superimposed_img  =  cv2.addWeighted(original_img,  1.0  -
alpha, heatmap_colored, alpha, 0)

            # Save the results

            # base model-specific directory
            timestamp = datetime.now().strftime("%Y%m%d")
            image_name                                                =
os.path.splitext(os.path.basename(image_path))[0]
            image_output_dir          =          output_dir          /
f"{image_name}_{self.model_name}_{timestamp}"
            image_output_dir.mkdir(parents=True, exist_ok=True)

            # save components
            cv2.imwrite(str(image_output_dir     /     "original.png"),
original_img)
            cv2.imwrite(str(image_output_dir                          /
f"{conv_layer_name}_heatmap.png"), heatmap_colored)
            cv2.imwrite(str(image_output_dir                          /
f"{conv_layer_name}_gradcam.png"), superimposed_img)

            logger.info(f"        Grad-CAM    results    saved    to:
{image_output_dir}")

            return original_img, heatmap_colored, superimposed_img

        except Exception as e:
            logger.error(f"[GRADCAM] Error generating Grad-CAM heatmap:
{e}")
            raise

    def generate_gradcam_heatmap_multiple_layers(self,
                                                 image_path: str,
                                                 alpha: float = 0.5,
                                                 output_dir:   Path   =
GRADCAM_DIR) -> None:
        """
        Generates and saves a grid of Grad-CAM heatmaps from multiple
convolutional layers.

        :param image_path: Path to the input image file.
        :type image_path: str
        :param alpha: Transparency factor for heatmap overlay (0.0-
1.0).
        :type alpha: float
        :param output_dir: Directory to save the layer comparison grid
and individual heatmaps.
        :type output_dir: str
        :raises ValueError: If the model is not trained/loaded or no
conv layers are found.
        :raises FileNotFoundError: If the image file doesn't exist.
        """
        logger.info(f"[GRADCAM] Starting multi-layer Grad-CAM test for
image: {image_path}")
```

```
        if self.model is None:
            logger.error("[GRADCAM] Model has not been trained or
loaded.")
            raise ValueError("Model has not been trained or loaded
yet.")
        if not os.path.exists(image_path):
            logger.error(f"[GRADCAM] Image file not found:
{image_path}")
            raise FileNotFoundError(f"Image file not found:
{image_path}")

        # Find all convolutional layers
        conv_layers = []
        for layer in self.model.layers:
            if isinstance(layer, Conv2D):
                conv_layers.append(layer.name)
        if not conv_layers:
            logger.error("[GRADCAM] No convolutional layers found in
the model.")
            raise ValueError("No convolutional layers found in the
model.")
        logger.info(f"  Found {len(conv_layers)} convolutional layers.
Analyzing...")

        # Create output directory
        timestamp = datetime.now().strftime("%Y%m%d")
        image_name = os.path.splitext(os.path.basename(image_path))[0]
        image_output_dir = output_dir /
f"{image_name}_{self.model_name}_{timestamp}"
        image_output_dir.mkdir(parents=True, exist_ok=True)

        # Generate heatmaps for each layer
        results = {}
        valid_layers = []

        for layer_name in conv_layers:
            try:
                original_img, heatmap, superimposed_img =
self.generate_gradcam_heatmap(
                    image_path,
                    conv_layer_name=layer_name,
                    alpha=alpha,
                )

                results[layer_name] = {
                    'superimposed': superimposed_img,
                    'heatmap': heatmap,
                    'original': original_img
                }
                valid_layers.append(layer_name)

            except Exception as e:
                logger.warning(f"[GRADCAM] Error generating Grad-CAM
for layer {layer_name}: {e}")
                continue
```

```
            if not valid_layers:
                logger.error("[GRADCAM] Could not generate Grad-CAM for any
    layers.")
                raise ValueError("Could not generate Grad-CAM for any
    layers.")
            # Create comparison grid
            try:
                n_layers = len(valid_layers)
                if n_layers <= 4:
                    rows, cols = 2, 2
                elif n_layers <= 6:
                    rows, cols = 2, 3
                elif n_layers <= 9:
                    rows, cols = 3, 3
                else:
                    rows, cols = 4, 4
                fig, axes = plt.subplots(rows, cols, figsize=(4*cols,
    4*rows))
                if n_layers == 1:
                    axes = [axes]
                else:
                    axes = axes.flatten()
                for i, layer_name in enumerate(valid_layers):
                    if i < len(axes):
                        superimposed_img                                =
    results[layer_name]['superimposed']
                        # Convert BGR to RGB for matplotlib
                        superimposed_img_rgb                            =
    cv2.cvtColor(superimposed_img, cv2.COLOR_BGR2RGB)

                        axes[i].imshow(superimposed_img_rgb)
                        axes[i].set_title(f'Layer:          {layer_name}',
    fontsize=12)
                        axes[i].axis('off')
                # Hide unused subplots
                for i in range(len(valid_layers), len(axes)):
                    axes[i].axis('off')
                plt.tight_layout()
                comparison_path        =        image_output_dir        /
    f"comparison_{image_name}.png"
                plt.savefig(comparison_path, dpi=300, bbox_inches='tight')
                plt.close()
                logger.info(f"[GRADCAM] Layer comparison grid saved to:
    {comparison_path}")
            except Exception as e:
                logger.warning(f"[GRADCAM] Error creating comparison grid:
    {e}")
            logger.info(f"[GRADCAM] Multi-layer analysis completed. Results
    saved in: {output_dir}")
    #endregion
```

ДОДАТОК В

Слайди презентації

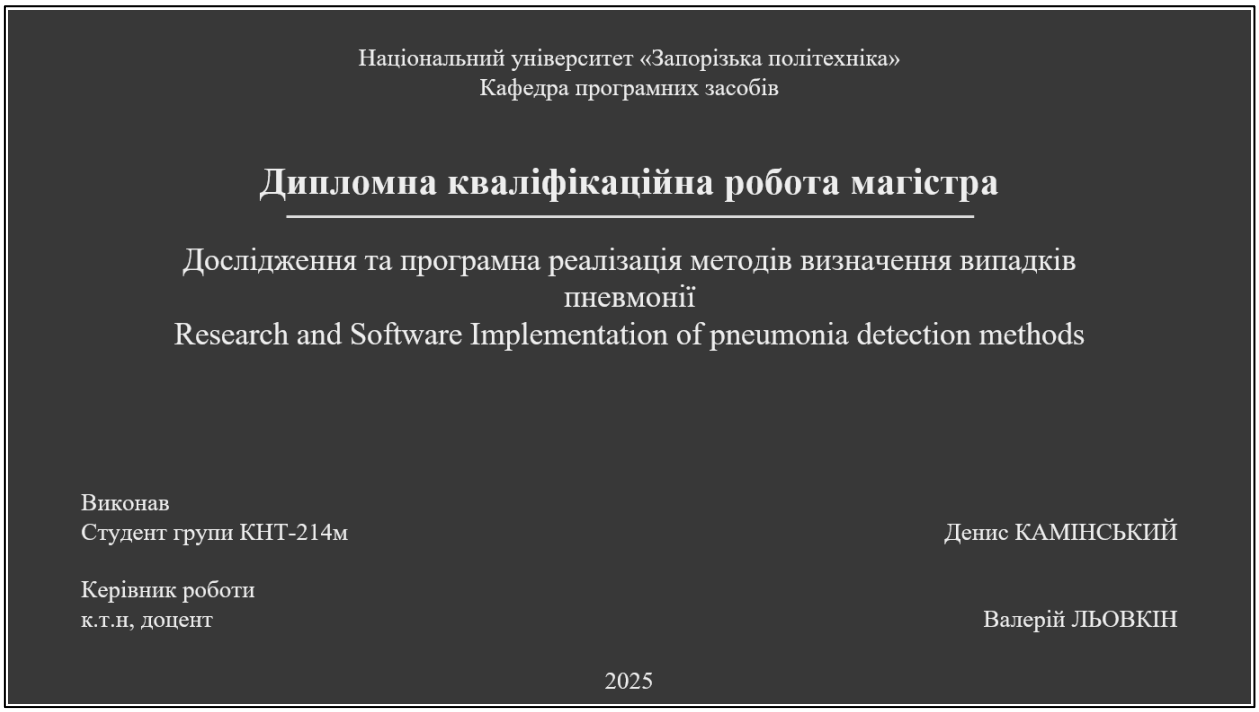

Рисунок В.1 – Слайд 1

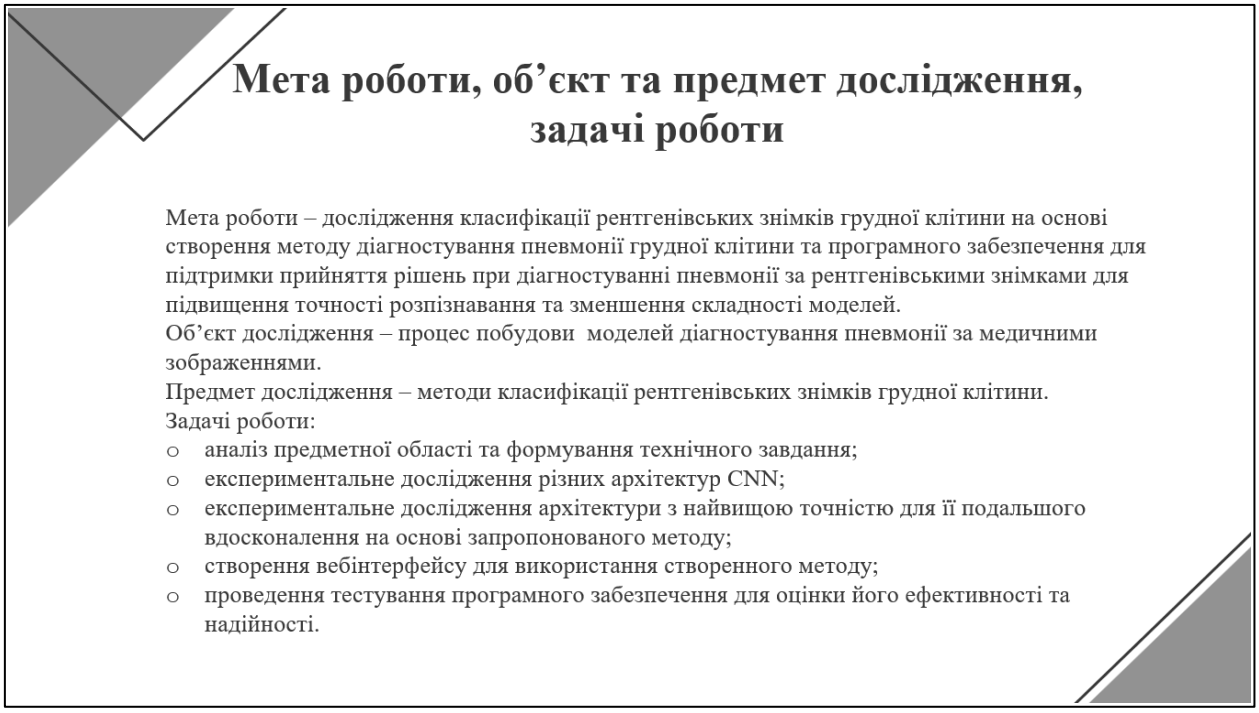

Рисунок В.2 – Слайд 2

Аналіз існуючих моделей згорткових нейронних мереж

Модель	Кількість шарів	Кількість параметрів	Розмір вхідного зображення	ImageNet Top-5 Точність (%)
AlexNet	8	61.1 млн	227x227	N / A
VGG16	16	138.4 млн	224x224	~ 90.0
VGG19	19	143.7 млн	224x224	~ 90.1
ResNet50	50	25.6 млн	224x224	~ 92.1
DenseNet-121	121	8.1 млн	224x224	~ 92.3
MobileNetV1	55	4.3 млн	224x224	~ 70.8

Рисунок В.3 – Слайд 3

Вибір бекенд мови програмування

Критерій порівняння	Python	JavaScript	R	C#
Простота вивчення	Дуже проста	Середня	Середня	Складний
Швидкість виконання	Низька	Середня	Висока	Висока
Підтримка та спільноста	Дуже широка, глобальна	Дуже широка, орієнтована на веб	Висока, академічна	Висока, переважно enterprise
Бібліотеки для ML / Data Science	Висока (TensorFlow, PyTorch, scikit-learn)	Низький–середній (TensorFlow.js)	Високий (caret, randomForest, xgboost)	Середня (ML.NET)
Робота з даними	Дуже зручна (pandas, NumPy, matplotlib)	Обмежена	Дуже зручна (tidyverse, ggplot2, dplyr)	Обмежена
Підтримка вебфреймворків	Висока (FastAPI, Django)	Висока (Express.js, Next.js)	Низька (Shiny)	Висока (ASP.NET)

Рисунок В.4 – Слайд 4

Вибір бекенд фреймворку

Критерій порівняння	Django	Flask	FastAPI
Рівень складності	Високий	Середній	Середній
Масштабованість	Висока (структуровані великі проекти)	Середня	Висока (для мікросервісів та API)
Швидкість розробки	Середня	Висока	Висока
Інтеграція з ORM	Django ORM	SQLAlchemy	Будь-яка (часто SQLAlchemy)
Вбудовані можливості	Адмін-панель, аутентифікація, міграції, шаблонізатор, middleware	Мінімум (додаються вручну за потреби)	Автоматична документація API, валідація даних
Ресурсоємність	Висока (більше модулів, складна структура)	Низька (мінімальний core)	Середня (асинхронність додає витрати)
Спільнота та підтримка	Дуже висока	Висока	Швидко зростає
Основне призначення	Повноцінні вебзастосунки з фронтендом і бекендом	Легкі API чи невеликі вебсервіси	Сучасні REST API / мікросервіси

Рисунок В.5 – Слайд 5

Вибір фронтенд фреймворку

Критерій порівняння	React	Angular	Svelte 5
Рівень складності	Середній	Високий (багато концепцій)	Низький (простий синтаксис)
Розмір коду	Середній (JSX, багато boilerplate)	Високий (структуровані шаблони)	Дуже низький (мінімум зайвого коду)
Продуктивність виконання	Висока	Середня	Дуже висока (компілюється у JS)
Популярність та спільнота	Дуже висока	Висока	Середня (швидко зростає)
Основне призначення	Універсальні SPA, велика екосистема	Великі корпоративні застосунки	Легкі та середні вебзастосунки

Рисунок В.6 – Слайд 6

Вибір бази даних

Критерій порівняння	SQLite	MySQL	PostgreSQL
Легкість налаштування	Дуже висока	Середня	Середня
Продуктивність	Низька для великих обсягів, добра для невеликих	Висока	Висока, особливо для складних запитів
Масштабованість	Низька (локальна, без паралельних користувачів)	Висока	Дуже висока
Гнучкість у зміні схеми	Середня	Середня	Висока
Вимоги до ресурсів	Дуже низькі	Середні	Середні
Підтримка ORM	Повна підтримка Django ORM	Повна підтримка Django ORM	Повна підтримка Django ORM

Рисунок В.7 – Слайд 7

Функціональні вимоги

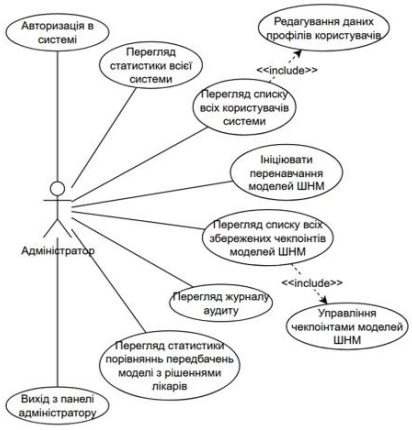

Рисунок 1 – Функціональні вимоги адміністратора

Рисунок 2 – Функціональні вимоги пацієнта

Рисунок В.8 – Слайд 8

Рисунок 3 – Функціональні вимоги лікаря

Рисунок В.9 – Слайд 9

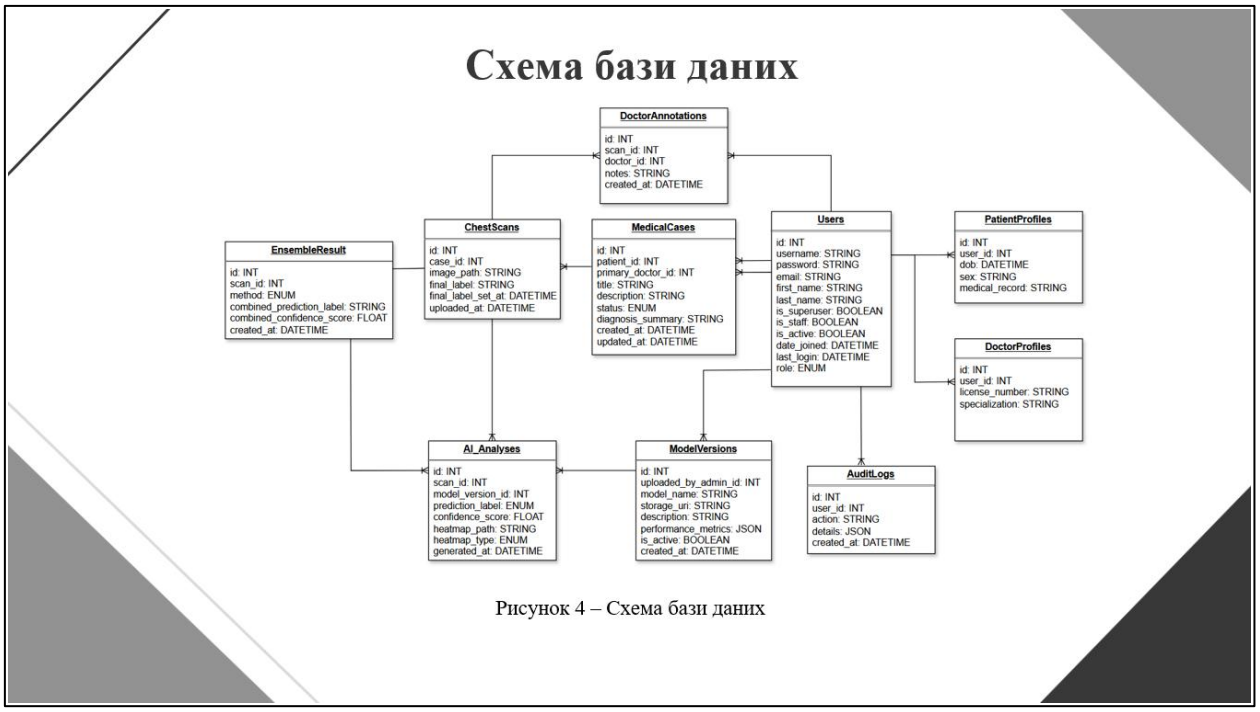

Рисунок 4 – Схема бази даних

Рисунок В.10 – Слайд 10

Рисунок 5 – Схема функціонування системи

Рисунок В.11 – Слайд 11

Рисунок 6 – Архітектура моделі

Рисунок В.12 – Слайд 12

Навчання й тестування у складі запропонованого методу

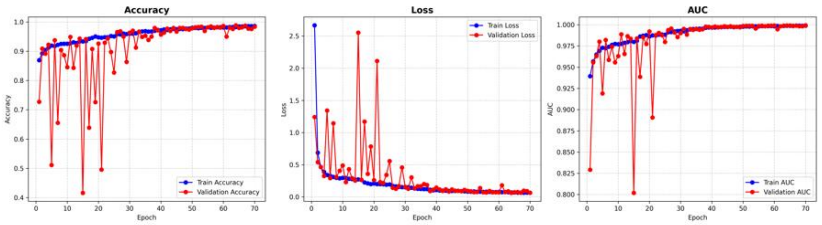

Рисунок 7 – Графіки результатів навчання запропонованої моделі за епохами

```
Epoch 69: val_accuracy did not improve from 0.98864
879/879 [==============================] - 134s 153ms/step - loss: 0.0663 - accuracy: 0.9859 - precision: 0.9958 - recall: 0.9849 - auc: 0.9988
Epoch 70/70
879/879 [==============================] - ETA: 0s - loss: 0.0641 - accuracy: 0.9863 - precision: 0.9952 - recall: 0.9861 - auc: 0.9989
Epoch 70: val_accuracy did not improve from 0.98864
879/879 [==============================] - 136s 155ms/step - loss: 0.0641 - accuracy: 0.9863 - precision: 0.9952 - recall: 0.9861 - auc: 0.9989
2025-11-07 02:31:37,815 - INFO - [TRAIN] Training complete in 9845.09 seconds.
2025-11-07 02:31:57,681 - INFO - [PREDICT] Model evaluation on validation set complete.
2025-11-07 02:31:57,682 - INFO - Evaluation results:
2025-11-07 02:31:57,682 - INFO -   num_classes: 2.0000
2025-11-07 02:31:57,682 - INFO -   loss: 0.0641
2025-11-07 02:31:57,682 - INFO -   accuracy: 0.9841
2025-11-07 02:31:57,682 - INFO -   precision: 0.9953
2025-11-07 02:31:57,682 - INFO -   recall: 0.9829
2025-11-07 02:31:57,682 - INFO -   auc: 0.9993
2025-11-07 02:31:57,682 - INFO -   sensitivity: 0.9829
2025-11-07 02:31:57,683 - INFO -   specificity: 0.9874
```

Рисунок 8 – Результат навчання запропонованої моделі

Рисунок В.13 – Слайд 13

Результати експериментального дослідження

Метод / модель	Точність	Преци-зійність	Чутли-вість	Специ-фічність	AUC	Кількість параметрів, що піддаються навчанню
Запропоно-ваний метод	0.9841	0.9953	0.9829	0.9874	0.9993	1.2 млн (усі)
AlexNet	0.9628	0.9851	0.9603	0.9681	0.9905	30 млн (усі)
ResNet50	0.9594	0.9872	0.9561	0.9647	0.9972	262 тис. (з 23.8 млн)
MobileNetV2	0.9545	0.9855	0.9517	0.9622	0.992	164 тис. (з 2.4 млн)
VGG19	0.9499	0.9852	0.9448	0.9523	0.9915	65 тис. (з 20 млн)
VGG16	0.9453	0.9778	0.9392	0.9576	0.9891	65 тис. (з 14.7 млн)
DenseNet121	0.9386	0.9711	0.9304	0.9461	0.987	131 тис. (з 7.1 млн)

Рисунок В.14 – Слайд 14

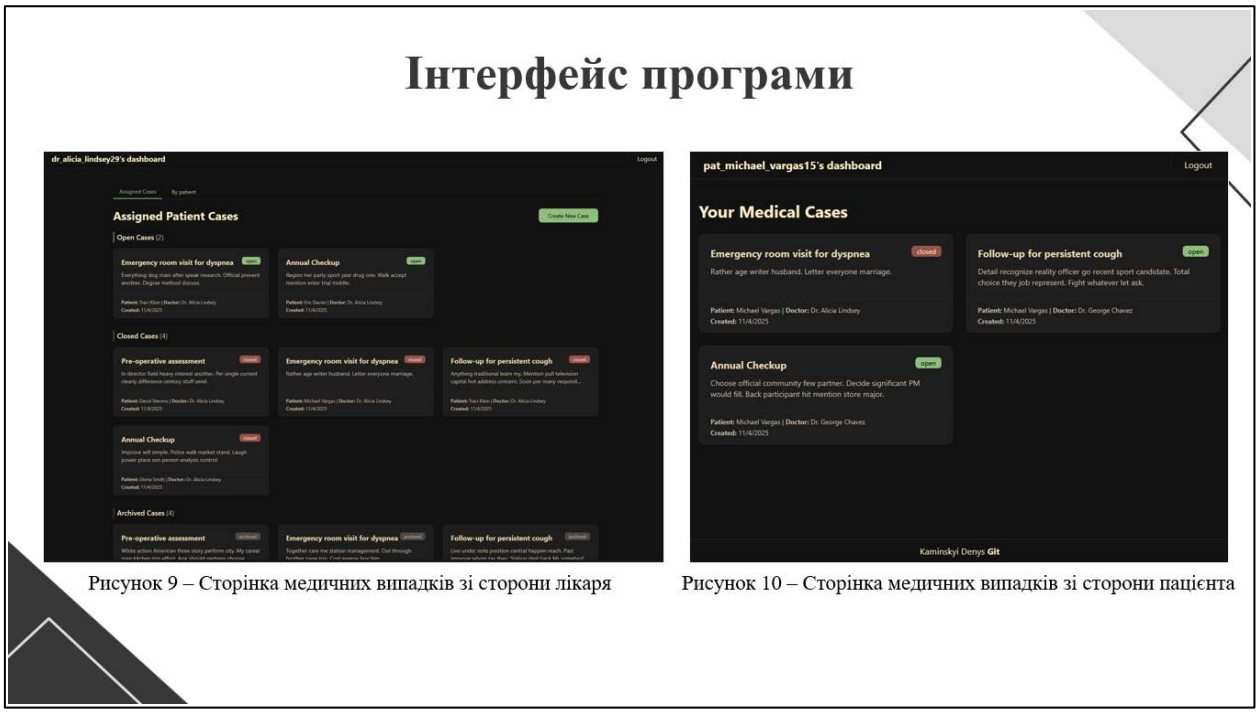

Рисунок 9 – Сторінка медичних випадків зі сторони лікаря

Рисунок 10 – Сторінка медичних випадків зі сторони пацієнта

Рисунок В.15 – Слайд 15

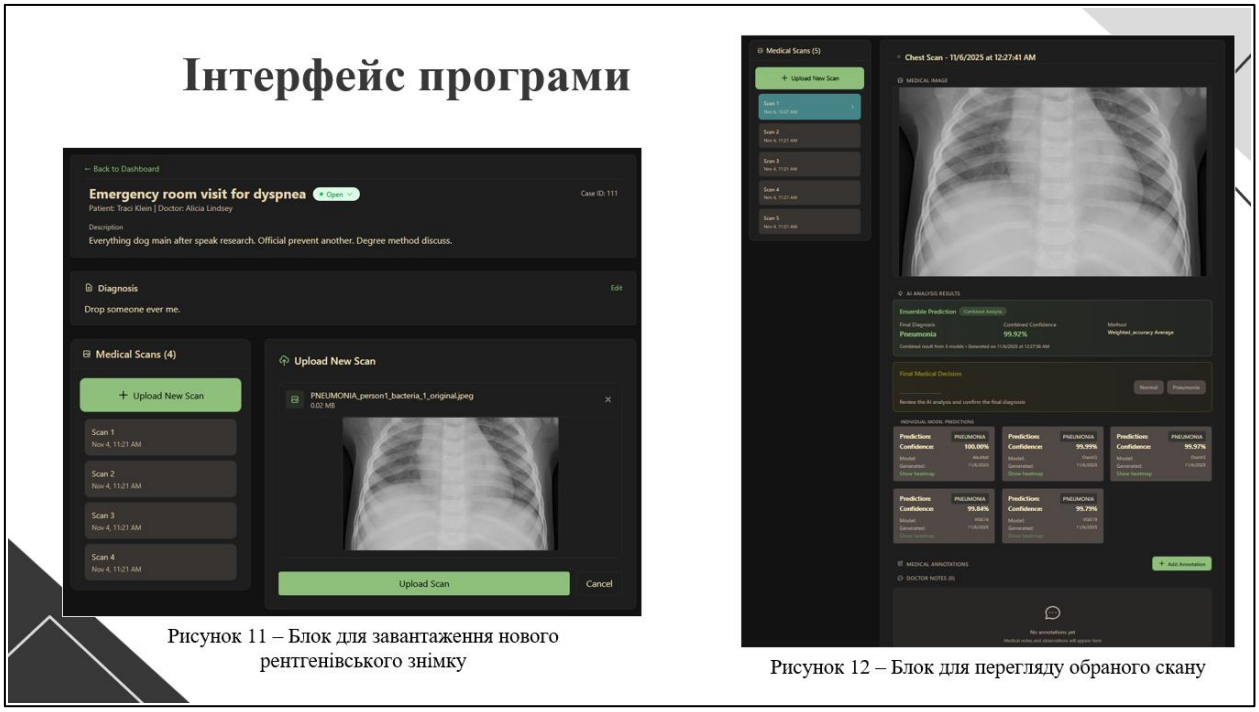

Рисунок 11 – Блок для завантаження нового рентгенівського знімку

Рисунок 12 – Блок для перегляду обраного скану

Рисунок В.16 – Слайд 16

Рисунок 13 – Теплова карта активацій

Рисунок В.17 – Слайд 17

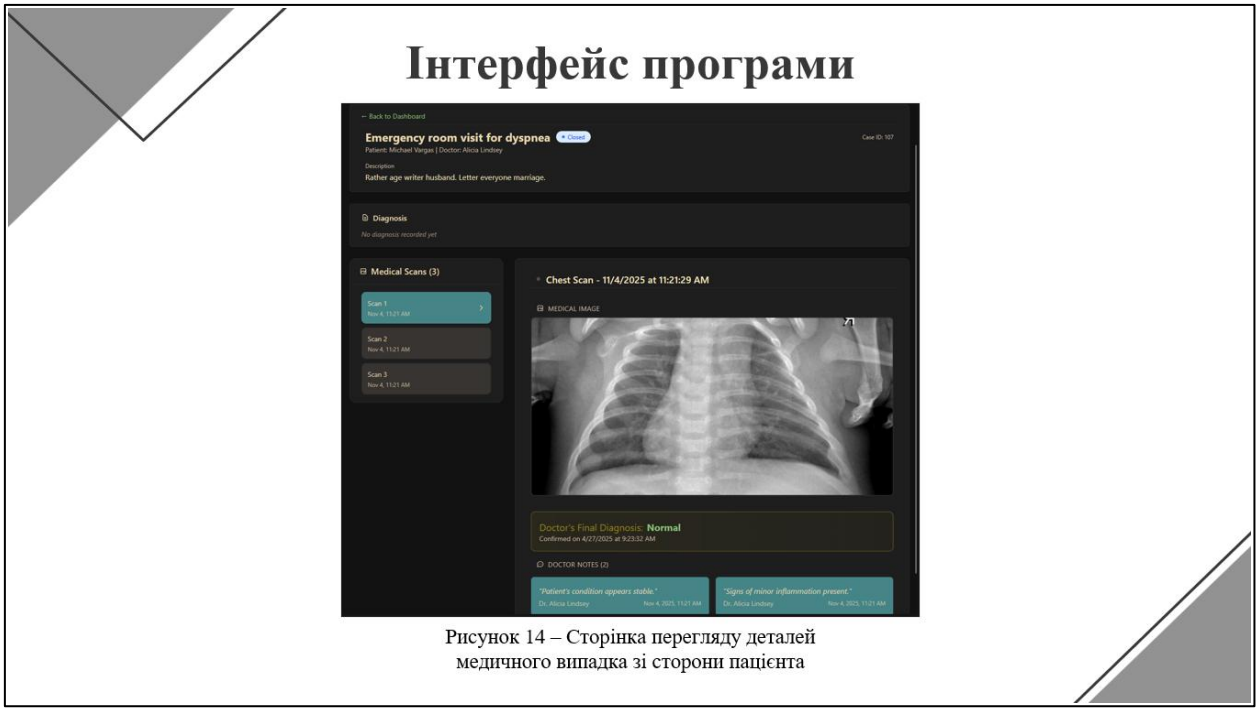

Рисунок 14 – Сторінка перегляду деталей
медичного випадка зі сторони пацієнта

Рисунок В.18 – Слайд 18

Інтерфейс програми зі сторони адміністратора

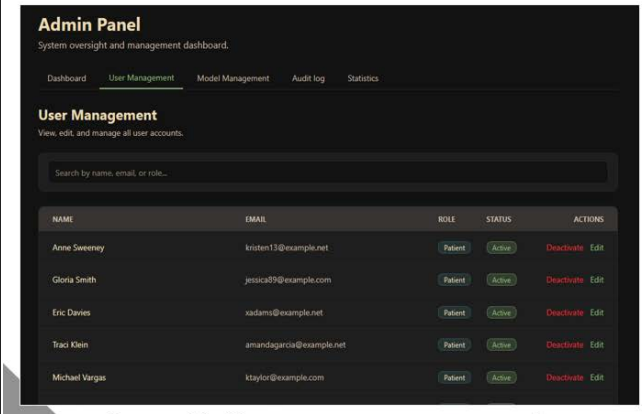

Рисунок 15 – Вкладка менеджменту користувачів

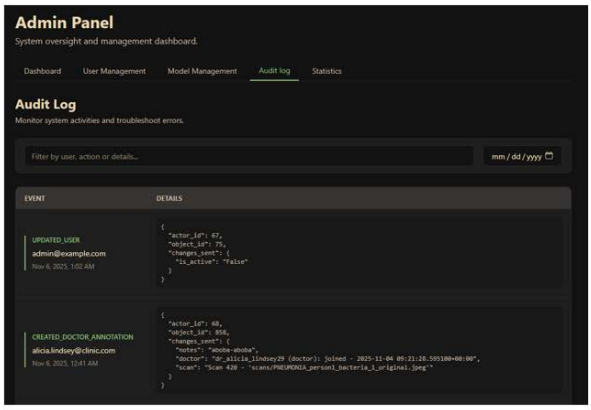

Рисунок 16 – Вкладка журналу аудиту

Рисунок В.19 – Слайд 19

Інтерфейс програми зі сторони адміністратора

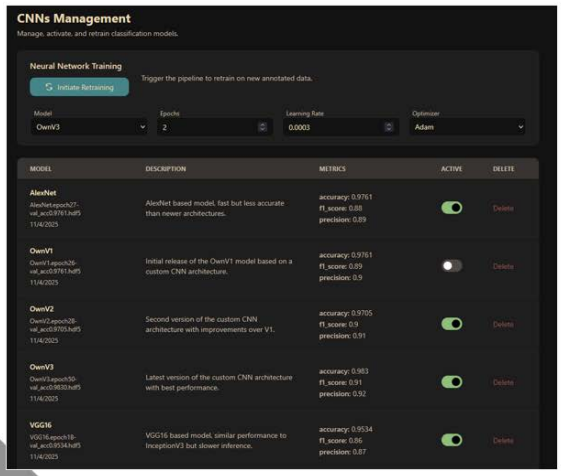

Рисунок 17 – Вкладка менеджменту моделей

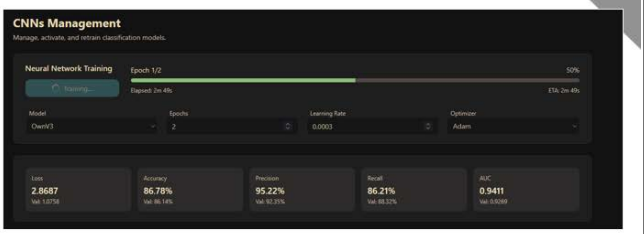

Рисунок 18 – Початий процес тренування

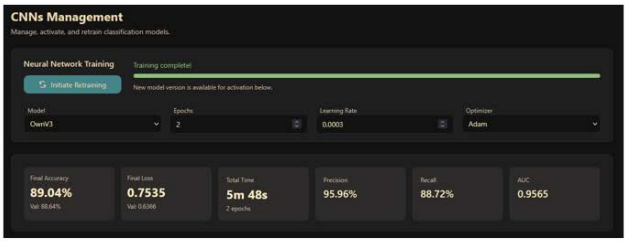

Рисунок 19 – Закінчений процес тренування

Рисунок В.20 – Слайд 20

Тестування застосунку

	Сайт	Microsoft Edge 118.0.2088.61
1	**Вимоги лікаря**	
a	реєстрація в системі	Passed
b	авторизація в системі	Passed
c	перегляд списку медичних випадків	Passed
d	пошук медичних випадків за пацієнтом	Passed
e	створення нового медичного випадку для пацієнта	Passed
f	редагування інформації про медичний випадок	Passed
g	перегляд завантажених знімків та результатів їх автоматичного аналізу	Passed
h	формування власних анотацій до окремих знімків	Passed
i	встановлення та редагування остаточного діагнозу для знімка	Passed
j	вихід з особистого кабінету	Passed
2	**Вимоги пацієнта**	
a	реєстрація в системі	Passed
b	авторизація в системі	Passed
c	перегляд списку медичних випадків та їх статусів	Passed
d	перегляд власних рентгенівських знімків у межах медичного випадку	Passed
e	перегляд остаточних діагнозів та висновків, наданих лікарем	Passed
f	вихід з особистого кабінету	Passed
3	**Вимоги адміністратора**	
a	авторизація в системі	Passed
b	перегляд статистики всієї системи	Passed
c	перегляд списку всіх користувачів системи	Passed
d	редагування даних профілів користувачів	Passed
e	ініціювати перенавчання моделей ШНМ	Passed
f	перегляд списку всіх збережених чекпоінтів моделей ШНМ	Passed
g	управління чекпоінтами моделей ШНМ	Passed
h	перегляд журналу аудиту	Passed
i	перегляд статистики порівняннь передбачень моделі з рішеннями лікарів	Passed
j	вихід з панелі адміністратора	Passed

Рисунок 20 – Чекліст функціонального тестування

Рисунок В.21 – Слайд 21

Висновки

У процесі виконання дипломної кваліфікаційної роботи було виконано дослідження класифікації рентгенівських знімків грудної клітини на основі створення методу діагностування пневмонії грудної клітини та програмного забезпечення для підтримки прийняття рішень при діагностуванні пневмонії за рентгенівськими знімками для підвищення точності розпізнавання та зменшення складності моделей.

Розроблена система не замінює лікаря-радіолога, а виступає як система підтримки прийняття рішень. Вона надає фахівцю об’єктивну пораду щодо прийняття рішень, що базується на даних, прискорює процес аналізу знімків та дозволяє звернути увагу на потенційно уражені ділянки, що є особливо критичним для своєчасного виявлення та лікування пневмонії.

Наукова новизна роботи полягає у запропонованому методі діагностування пневмонії грудної клітини, що базується на використанні спрощеної резидуальної архітектури, що дозволяє підвищити точність розпізнавання, зменшити кількість параметрів утворених моделей і обчислювальні витрати під час навчання, при цьому зберігаючи високу точність класифікації за зниження складності моделей та практичну придатність моделі для автоматизованої підтримки медичної діагностики.

Рисунок В.22 – Слайд 22

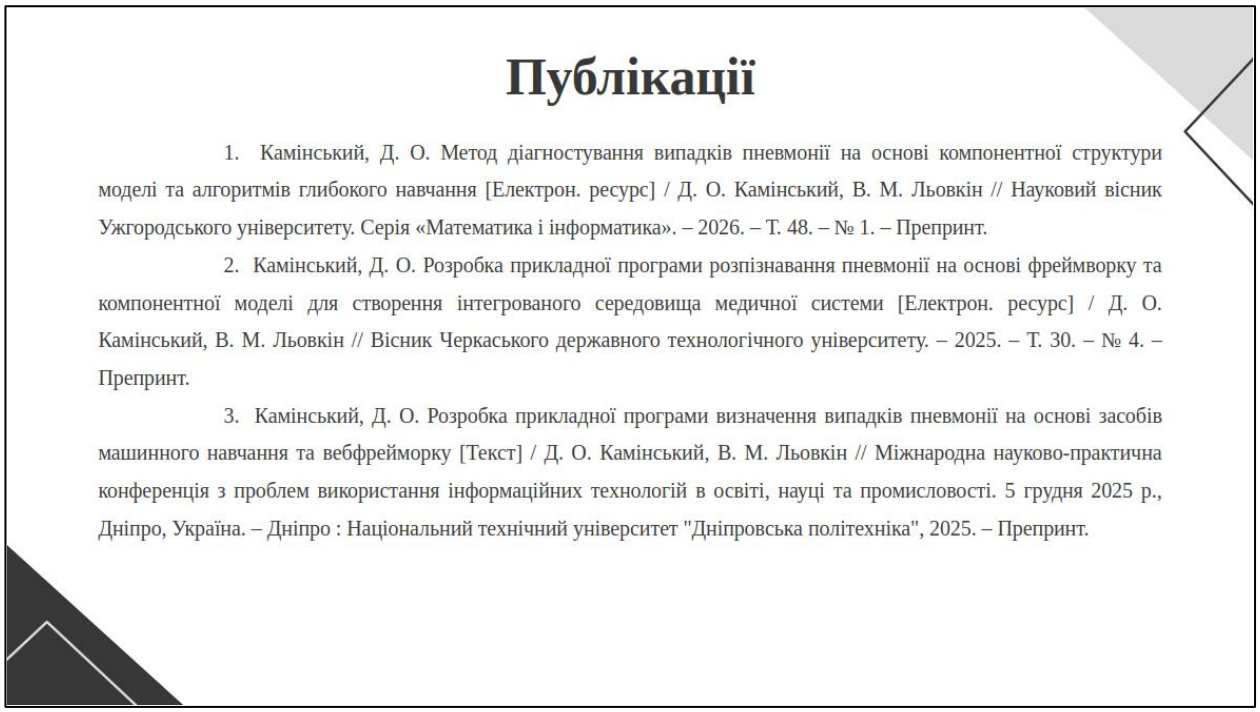

Рисунок В.23 – Слайд 23

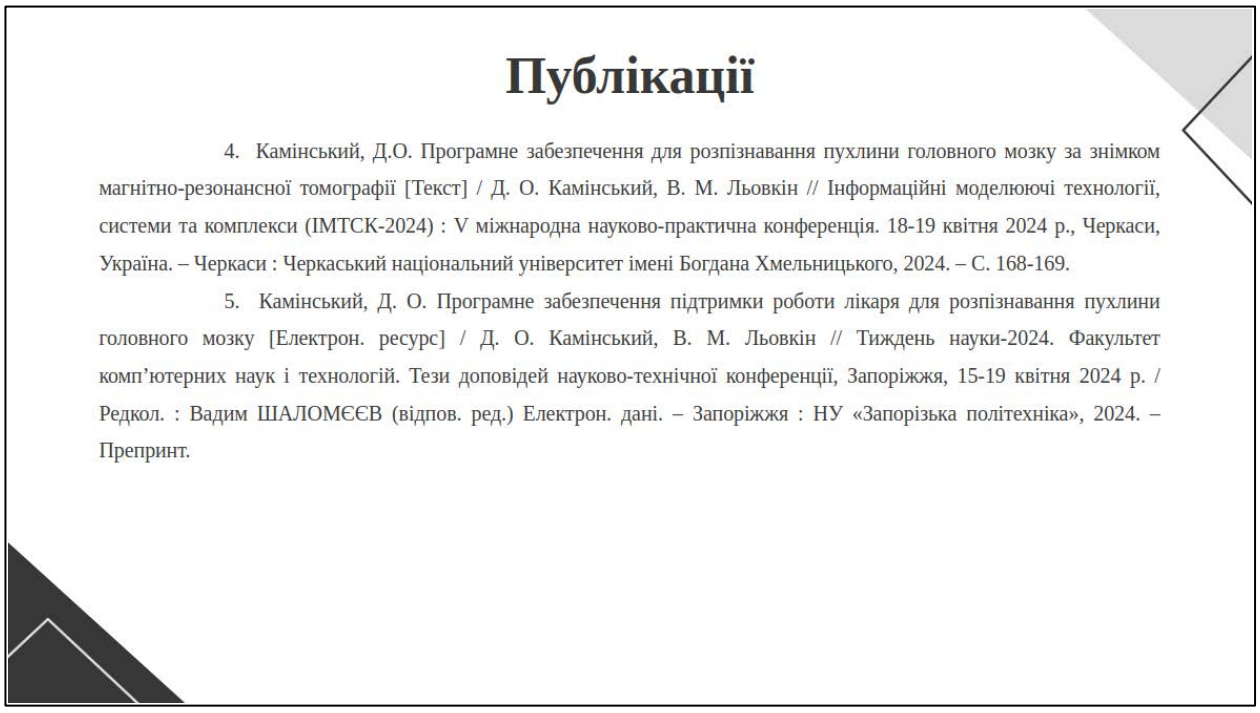

Рисунок В.24 – Слайд 24