УДК 20.57
УКПП 72.19.12-00.00, 72.19.30-00.00
№ ДР 0120U105119
інв. № _____

ТОВ "ТАНХОСТ"
02081, м. Київ, вул. Здолбунівська, 7Д, корпус Б1, тел.: +380 444 92 29 19

ЗАТВЕРДЖУЮ
Директора ТОВ "ТАНХОСТ"
Андрій Олександрович
_____ (підпис)
_____ 2021 р.
(місяць)

ЗАКЛЮЧНИЙ ЗВІТ СУБВИКОНАВЦЯ

ПРО РЕЗУЛЬТАТИ РЕАЛІЗАЦІЇ ПРОЄКТУ 2020.01/0490

за договором № 192/01/0490 від 18 травня 2021 року

Платформа штучного інтелекту для дистанційного автоматизованого виявлення та діагностики захворювань людини

Науковий керівник Проєкту
В. о. директора
_____ (підпис) А. О. Кільколих

2021

Розглянуто та прийнято Вченою (технічною) радою Субвиконавця, протокол № ___ від ___ грудня 2021 р.

СПИСОК АВТОРІВ

№	Посада, вчений ступінь, вчене звання	Ініціали, прізвище (частина звіту, що підготовлена автором)
1	Ключовий виконавець проєкту, в. о. директора (підпис) // (дата)	А. О. Кільколих (реферат, вступ, розд. 1, 2, 3, висновки)
2	Виконавець проєкту, бухгалтер (підпис) // (дата)	О. В. Глущенко (реферат)
3	Виконавець проєкту, менеджер (управитель) з організації консультативних послуг (підпис) // (дата)	Є. А. Некраха (висновки)
4	Виконавець проєкту, менеджер (управитель) з адміністративної діяльності (підпис) // (дата)	А. С. Волощук (висновки)
5	Виконавець проєкту, адміністратор системи (підпис) // (дата)	Є. М. Дорошенко (розд. 1, 2, 3)
6	Виконавець проєкту, програміст (підпис) // (дата)	В. С. Мартиненко (розд. 1, 2, 3)
7	Виконавець проєкту, програміст (підпис) (дата)	І. В. Панасюк (розд. 1, 2, 3)
8	Виконавець проєкту, програміст (підпис) // (дата)	В. Р. Білецький (розд. 1, 2, 3)

РЕФЕРАТ

Звіт: 172 сторінок, 93 рисунки, 7 таблиці, 90 джерел.

Ключові слова: *штучний інтелект для охорони здоров'я, глибинні нейронні мережі, ідентифікація, класифікація, семантична сегментація.*

Об'єкт дослідження: процеси дистанційного автоматизованого виявлення (Computer-Aided Detection - CADe) і дистанційної автоматизованої діагностики (Computer-Aided Diagnostics - CADx) хвороб людини.

Мета роботи: створення платформи штучного інтелекту (ШІ) для розробки, втілення та інтеграції методів ШІ для CADe/CADx на основі обробки наборів медичних даних великого обсягу (датасетів).

Методи дослідження: побудова методології використання методів ШІ, а саме методів глибинного навчання (deep learning — DL) для автоматизованого аналізу медичних даних різного типу і розмірності, включаючи одновимірні електромагнітні, двовимірні рентгенографічні, тривимірні комп'ютерної томографії (КТ/МРТ), електроенцефалографії (ЕЕГ), тощо.

Відповідно до календарного плану під час виконання проєкту були виконані наступні завдання:

1. Налаштовано обладнання у конфігурації віртуального прототипу запланованого обладнання із графічним прискорювачем (віртуальна машина із дистанційним мережевим доступом).
2. Встановлено програмне забезпечення (на основі хмарних серверів та клієнтів) для розробки DL-моделей, зберігання стандартних наборів даних (JSRT, ChestXray, CheXpert, ISIC, ЕЕГ, та інші).
3. Виконано загальні тести обладнання та програмного забезпечення.
4. Виконано в повному обсязі початкові базові випробування стандартних та інших наборів даних на конфігурації віртуального прототипу фізичного обладнання.
5. Виконано дослідження та порівняльний аналіз різних стратегій побудови теплових мап (ймовірної локалізації захворювання).
6. Виконано тестування та аналіз результатів прогнозування (Prediction Tests and Analysis – PTA) для моделей DTL / ADL із використанням штучного збільшення різноманітності даних різного ступеню інтенсивності.

7. Досліджено, розроблено та протестовано платформу (хмарний сервер і клієнтська сторона) штучного інтелекту (ШІ-платформу), налаштовано пілотну конфігурацію ШІ-платформи (з веб-демонстрацією) для широкого спектру ШІ-моделей (базових DL-моделей, TDL-19 моделей, ADL-19 моделей).

Відповідно до календарного плану протягом проєкту було отримано наступні науково-технічні результати:

1. Конфігурація придбаного обладнання із спеціалізованими прискорювачами типу GPU, яку налаштовано для використання в локальному та хмарному середовищі для потреб розробленої ШІ-платформи.
2. Віртуальні версії компонентів ШІ-платформи для використання на придбаному обладнанні із спеціалізованими прискорювачами типу GPU для використання в локальному та хмарному середовищі (віртуальні машини та контейнери із дистанційним мережевим доступом).
3. Програмне забезпечення (на основі хмарних серверів та клієнтів) для розробки та практичного використання DL-моделей, для зберігання та розширення наборів даних (JSRT, ChestXray, CheXpert, ISIC, ЕЕГ, та інші) та для взаємодії із користувачами (медичними працівниками) в практичних умовах.
4. Встановлені та адаптовані для використання набори закордонних та національних медичних даних (JSRT, ChestXray, CheXpert, CXR COVID-19, ISIC, ЕЕГ, та інші).
5. Розроблені науково обґрунтовані DL-моделі (на базі архітектур DenseNet, ResNet, InceptionResNet, Inception, Xception, EfficientNet, VGG та їхніх модифікованих версій та ансамблів), які були та адаптовані для використання в якості ШІ-асистентів лікаря на базі розробленої ШІ-платформи.
6. Результати тестів обладнання та програмного забезпечення для компонентів ШІ-платформи та ШІ-платформи в цілому для застосування у практичних умовах.
7. Конфігурація придбаного обладнання (модулі для периферійних обчислень (включно із Google Coral, Intel Movidius, тощо) та розробленого програмного забезпечення для використання в польових умовах.

8. ШІ-платформа (інтегрована апаратна та програмна інфраструктура) на базі придбаного, налаштованого та протестованого апаратного забезпечення із розробленим програмним забезпеченням медичного призначення (включно із встановленими та адаптованими для використання науково обґрунтованими DL-моделями).
9. Прототип ШІ-платформи (https://med.comsys.kpi.ua), тобто пілотний варіант конфігурації ШІ-платформи (серверна сторона та веб-інтерфейс для медичного працівника).
10. На основі результатів тестів ШІ-платформи, модулю для периферійних обчислень і врахування зворотного зв'язку від медичних працівників зроблено висновки щодо техніко-економічного обґрунтування подальшої імплементації та масштабування ШІ-платформи для використання в CADe та CADx в практичних умовах.

ЗМІСТ

Ст.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 10

ПЕРЕДМОВА 12

ВСТУП 13

РОЗДІЛ 1. ОРГАНІЗАЦІЯ АПАРАТНОГО І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 18

1.1 Конфігурація серверу із графічними прискорювачами 23

1.2 Конфігурація серверу із прискорювачами на основі тензорних обчислень 27

1.3 Конфігурація клієнтського робочого місця із прискорювачами на основі тензорних обчислень 29

1.4 Висновки до РОЗДІЛ 1 33

РОЗДІЛ 2. ВСТАНОВЛЕННЯ ТА НАВЧАННЯ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ (DTL-19) 35

2.1 Вибір/налаштування моделей глибинного навчання для фільтрації 2D рентгенограм легень 36

2.1.1 Вступ 36

2.1.2 Підготовка оточення 36

2.1.3 Попередня обробка даних 37

2.1.4 Завантаження та візуалізація даних 41

2.1.5 Підготовка моделі глибинного навчання (DL) 43

2.1.6 Навчання моделі глибинного навчання (DL) 44

2.1.7 Тестування моделі глибинного навчання (DL) 44

2.1.8 Візуалізація результатів тестування моделі глибинного навчання (DL) 47

2.1.9 Створення інфраструктури для використання у навчальному процесі 48

2.2 Вибір/налаштування моделей глибинного навчання (DL) для ідентифікації ознак COVID-19 із візуалізацією теплових карт . . . 49

2.2.1 Вступ 49
2.2.2 Попередня обробка даних 51
2.2.3 Візуалізація даних 58
2.2.4 Підготовка моделі глибинного навчання (DL) 59
2.2.5 Навчання DenseNet121 60
2.2.6 Візуалізація результатів тестування і створення теплових мап для тестової підмножини даних 62
2.2.7 Візуалізація теплових мап 67
2.3 Вибір/налаштування DL-моделей для оцінки їх пропускної здатності на наявному апаратному забезпеченні 70
2.3.1 Оцінка пропускної здатності моделей глибинного навчання - WU1 (Titan1080/HDD) 70
2.3.2 Оцінка пропускної здатності моделей глибинного навчання – WU2 (A100/SSD) 80
2.3.3 Створення інфраструктури (демо-версія) для тестування знімків від інших зовнішніх джерел 87
2.4 Вибір/налаштування моделей глибинного навчання (DL) для підвищення якості зображень 88
2.4.1 Вступ 88
2.4.2 Метрики 90
2.4.3 Результати 90
2.5 Вибір/налаштування моделей глибинного навчання (DL) для підвищення якості стискання зображень 92
2.5.1 Моделі для підвищення якості стискання зображень 93
2.5.2 Тренування мережі 94
2.6 Вибір/налаштування моделей глибинного навчання (DL) для покращеної сегментації зображень 95
2.6.1 Вступ 95
2.6.2 Моделі для покращеної сегментації зображень 96
2.7 Вибір/налаштування гібридних моделей глибинного навчання (DL) 100
2.7.1 Варіанти гібридизації 100
2.7.2 Метрики 102
2.8 Вибір/налаштування моделей глибинного навчання (DL) із застосуванням штучного розширення різноманітності даних . . . 104

2.8.1 Тренування моделей та аналіз отриманих результатів . . . 105
2.8.2 Аналіз результатів . . . 107
2.9 Рішення на основі технології блокчейн для доказу автентичності медичного контенту глибинного навчання (deep learning content - DLC) . . . 109
2.9.1 Структура системи . . . 109
2.9.2 Варіанти еволюції контенту машинного навчання . . . 113
2.9.3 Відслідковування походження контенту машинного навчання . . . 114
2.9.4 Контент машинного навчання з точки зору користувача . 116
2.10 Висновки до РОЗДІЛ 2 . . . 117

РОЗДІЛ 3. ПЛАТФОРМА ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ДИСТАНЦІЙНОГО АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ТА ДІАГНОСТИКИ ЗАХВОРЮВАНЬ ЛЮДИНИ . . . 120
3.1 Загальний опис платформи . . . 121
3.2 Інструкції для користування платформою . . . 124
3.3 Опис функціональних можливостей платформи . . . 137
3.3.1 ШІ-послуги . . . 137
3.3.2 ШІ-асистенти лікаря (ШІ-моделі) . . . 139
3.4 Компоненти платформи . . . 142
3.4.1 Сервер керування та представлення даних . . . 143
3.4.2 Сервер обробки даних . . . 144
3.4.3 Сервер тренування ШІ-асистентів . . . 146
3.4.4 Система збереження . . . 146
3.5 Технологія конфігурування та використання великої кількості екземплярів спеціалізованого прискорювача GPU-типу . . . 147
3.6 Оцінка способів оптимізації обробки медичних даних і доцільності масштабування обчислювальної інфраструктури . . . 148

ВИСНОВКИ . . . 154

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ . . . 156

СПИСОК РИСУНКІВ . . . 167

СПИСОК ТАБЛИЦЬ 172

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЕЕГ – Електроенцефалографія
КТ – Комп'ютерна томографія
МРТ – Магнітна резонансна томографія
ОС – Операційна система
ПЛР – Полімеразна ланцюгова реакція
ШІ – Штучний інтелект

AI – Artificial Intelligence
ASIC – Application-Specific Integrated Circuit
BCI – Brain-Computer Interface
CADe – Computer-Aided Detection
CADx – Computer-Aided Diagnostics
COVID-19 – COronaVIrus Disease 2019
CPU – Central Processing Unit
CXR – Chest X-ray
DA – Data Augmentation
DC – Dew Computing
DL – Deep Learning
DTL – Deep Transfer Learning
DNN – Deep Neural Network
DSP – Digital Signal Processor
EC – Edge Computing
ECA – Edge Computing Accelerators
FC – Fog Computing
FPGA – Field-Programmable Gate Array
GAL – Grasp-and-Lift
GPGPU – General Purpose Graphic Processing Unit
GPU – Graphic Processing Unit
HPC – High-Performance Computing
IoT – Internet of Things
ISIC – International Skin Imaging Collaboratio
LAN – Local Area Network

LDCT – Low-dose CT
ML – Machine Learning
NPU – Neural Processing Unit
PCR – Polymerase Chain Reaction
PSD – Power Spectral Density
SoC – System on a Chip
TC – Tensor Cores
TPA – Tensor Processing Architecture
TPU – Tensor Processing Unit
t-SNE – t-distributed Stochastic Neighbor Embedding
TTA – Test Time Augmentation
VPU – Vision Processing Unit
WE – Wearable Electronics

ПЕРЕДМОВА

Даний звіт містить опис виконаних завдань і отриманих результатів організацією-субвиконавцем Проєкту, а саме: компанією ТОВ “ТАНХОСТ”.

Звіти субвиконавців щодо їх виконаних завдань і отриманих результатів надаються окремо, а внесок (співавторство) усіх субвиконавців зазначено у відповідних розділах спільного загального звіту (окремий документ):

Субвиконавець. ТОВ “ТАНХОСТ”– реферат, вступ, розділ 1, 2, 3 та висновки спільного загального звіту (окремий документ).

ВСТУП

Актуальність. Медичний персонал сучасних медичних закладів має можливість отримувати надзвичайно великий обсяг даних через масове використання сучасних методів дослідження (комп'ютерна томографія - КТ, магнітно-резонансна томографія - МРТ та інші інструменти), які використовуються для діагностики стану здоров'я пацієнтів, а також завдяки поширенню цих даних серед медичних установ і навіть країн світу. Звісно, що надвеликий обсяг медичних даних, наприклад JSRT [1], LIDS [2], MCSH [3], [4], ChestX-ray14 [5], CheXpert [6], значно ускладнює персоналу роботу з даними. В рамках багатьох ініціатив по всьому світу наразі спостерігається широке застосування сучасних "розумних" інструментів на основі методів штучного інтелекту (ШІ) (artificial intelligence — AI). Як показують попередні дослідження, вони здатні значно покращити аналіз медичних даних, виявлення відхилень та ознак захворювань [5]—[7].

Запропонована в рамках проєкту платформа штучного інтелекту для дистанційного автоматизованого виявлення (computer-aided detection – CADe) та автоматизованої діагностики (computer-aided diagnosis – CADx) захворювань людини дозволить кінцевим споживачам (працівникам медичних закладів) створювати, розгортати та керувати спеціалізованими ШІ-додатками у гібридних обчислювальних середовищах (на основі хмарних та периферійних обчислень) для доступу до медичних даних надвеликого обсягу, їх обробки, аналізу та звітності в робочому процесі, виявлення відхилень та ознак захворювань.

У сучасному контексті різке зростання захворюваності на коронавірусну хворобу 2019 року (COronaVIrus Disease 2019 – COVID-19) спричинило навантаження на системи охорони здоров'я у всьому світі. Наразі існує нагальна потреба в ефективному ранньому виявленні COVID-19 у широкому загалі, діагностиці COVID-19 у пацієнтів із підозрою на захворювання та прогнозуванні перебігу COVID-19 у пацієнтів із підтвердженим захворюванням. ПЛР-тестування (полімеразна ланцюгова реакція – ПЛР) (polymerase chain reaction – PCR) та КТ грудної клітки наразі є стандартними методами діагностики COVID-19, але забирають багато часу.

У більш загальному контексті та часовому діапазоні захворювання легень (не лише на COVID-19, але також на рак, туберкульоз, пневмонію тощо) є зна-

чним тягарем для нинішньої популяції, особливо для дорослих людей, які є нинішніми або колишніми курцями. Більше того, значне забруднення повітря додатково сприяє захворюванням. Наприклад, за оцінками Всесвітньої організацій охорони здоров'я [8] за 2020 рік, рак легень у всьому світі посідає перше місце за кількістю летальних випадків (18.0% від загальної кількості смертей від раку) та друге місце за кількістю діагностованих нових випадків (11.4% від загальної кількості діагностованих нових випадків на рак). Ці дані свідчать про дуже високу захворюваність та смертність від раку легень, і теперішнім рішенням є проведення раннього скринінгу, що часто призводить до хороших результатів лікування з низькою вартістю. Обстеження КТ з низькою дозою опромінення (low-dose CT – LDCT) було визнано у всьому світі як один з ефективних підходів до зниження смертності від раку легень, але ці скани виявляють багато вузликів, які часто виявляються доброякісними, а непотрібні біопсії та операції можуть бути небезпечними для багатьох пацієнтів.

Завдяки порівняно дешевій ціні та легкій доступності, рентгенографія грудної клітки (chest X-ray – CXR) широко застосовується для моніторингу здоров'я та діагностики багатьох захворювань легень (пневмонія, туберкульоз, рак тощо). "Ручний" аналіз ознак цих захворювань, виявлених за допомогою CXR, проводиться експертами-рентгенологами, що є часто тривалим і складним процесом. Тим не менш, сучасна еволюція апаратних засобів та програмного забезпечення для графічної обробки (GPGPU – обчислення загального призначення на графічних процесорах), особливо технік глибинного навчання, включаючи зусилля авторів цієї пропозиції [9]—[12], дозволяє вченим автоматично виявляти багато захворювань легень за зображеннями CXR на рівні, що перевищує сертифікованих рентгенологів [5], [6]. Незважаючи на ці успіхи, серед експертів існує переконання, що методи глибинного навчання стають ефективними у випадку використання відносно великих наборів даних ($> 10^4$ зображень (навчальних прикладів)), оскільки моделі, які навчались на малих наборах ($< 10^3$ зображень) дають погані прогнози і як наслідок, такі моделі погано узагальнюють на нових даних, а тому володіють дуже низькою продуктивністю за різними метриками оцінок. Проте, методи штучного збільшення різноманітності та зменшення розмірності даних, які запропоновані авторами цієї пропозиції (див. нижче), можуть дозволити уникнути пастки малого обсягу даних.

Незважаючи на безліч значних результатів, яких досягли сучасні CADe/CADx системи, все ще є певні проблеми з якими сьогодні вони стикаються. Через велику доступність даних та необхідність аналізу таких даних, однією з найбільших проблем є згадана проблема великих даних, з якою сьогодні стикаються системи CADe/CADx. Часто медичні дані є складними і можуть бути неструктурованими, напівструктурованими, неправильно маркованими або немаркованими даними. Виникає потреба у високорозвинених підходах до їх зберігання, пошуку та аналізу в стислі строки. Величезні зусилля були присвячені створенню інноваційних методик для основних етапів обробки медичних даних (сегментація, вилучення / відбір ознак та класифікація захворювань) у наявних системах [7]. Але все ще не існує єдиного найкращого алгоритму для кожного етапу обробки з меншим обчислювальним часом, тому додаткові дослідження у створенні інноваційних AI-алгоритмів для всіх аспектів доступних систем є надзвичайно важливими. "Ручний" аналіз цих зображень може бути трудомістким і ненадійним.

Об'єкт дослідження. В цьому контексті в якості об'єкту дослідження було обрано процеси дистанційного автоматизованого виявлення (Computer-Aided Detection – CADe) і дистанційної автоматизованої діагностики (Computer-Aided Diagnostics – CADx) хвороб людини.

Метою роботи є створення платформи штучного інтелекту (ШІ) для розробки, втілення та інтеграції методів ШІ для CADe/CADx на основі обробки наборів медичних даних великого обсягу (датасетів).

Методи дослідження. Для досягнення цієї мети було обрано наступні методи дослідження, які є складовою методів ШІ, а саме: методи машинного навчання (machine learning – ML) та методи глибинного навчання (deep learning – DL) для автоматизованого аналізу медичних даних різного типу і розмірності, включаючи одновимірні акустичні, двовимірні рентгенографічні, тривимірні комп'ютерної томографії (КТ/МРТ), багатоканальної електроенцефалографії (ЕЕГ), тощо.

Завдяки їх використанню запропонована ШІ-платформа може покращити основні показники (точність, чутливість, специфічність, тощо) методів CADe/CADx та замінити людей у безпечний та економічно ефективний спосіб. Крім того, це дозволить використовувати запропоновану ШІ-методологію для аналізу інших різних медичних даних, включаючи одновимірні (1D) акустичні,

двовимірні (2D) рентгенографічні, тривимірні (3D) КТ/МРТ томографії, багатоканальні електроенцефалографічні (ЕЕГ), тощо.

Пропонована платформа штучного інтелекту для дистанційного автоматизованого виявлення (computer-aided detection - CADe) та автоматизованої діагностики (computer-aided diagnosis - CADx) захворювань людини дозволить кінцевим споживачам (працівникам медичних закладів) створювати, розгортати та керувати спеціалізованими ШІ-додатками у гібридних обчислювальних середовищах (на основі хмарних та периферійних обчислень) для доступу до медичних даних надвеликого обсягу, їх обробки, аналізу та звітності в робочому процесі, виявлення відхилень та ознак захворювань.

Широко відомі існуючі рішення [7] характеризуються деякими недоліками, які пов'язані з: відхиленням даних (biased datasets), які часто серйозно зменшують показники розроблених DL-моделей [13], [14], недбалим або непрофесійним (автоматичним) маркуванням даних [15], [16], дисбалансом даних (imbalanced datasets) при наявності слабо представлених класів у датасетах [17], особливо для COVID-19, та обмеженістю даних взагалі по рідкісних хворобах. На відміну від існуючих рішень в даному проєкті пропонується використати кілька перспективних підходів на основі методів, які були розроблені авторами цієї пропозиції для уникнення цих недоліків.

Відповідно до календарного плану під час підготовчого етапу було поставлено за **мету**:

1. Виконати попередню обробку датасетів.
2. Сегментувати найбільш цікаві ділянки (region of interest – ROI).
3. Зменшити розмірність даних та штучно збільшити різноманітність даних.
4. Випробувати DL-методи для класифікації захворювань.
5. Застосувати методи для обробки надвеликих даних (Big Data).

Для досягнення поставленої мети на підготовчого етапі було поставлено для Субвиконавця наступні **завдання**:

1. Налаштування придбаного обладнання (для залученого робочого вузла 1 (WU1) та вузла 2 (WU2) на встановленій робочій станції).
2. Встановлення програмного забезпечення (на базі хмарного сервера та на стороні клієнта: ОС, бібліотеки-оболонки, бібліотеки тощо) (WU1 / WU2).

3. Виконати загальні тести обладнання та програмного забезпечення (WU1 / WU2).

Відповідно до календарного плану під час Етапу №1: "Адаптація нового набору даних рентгенографії грудної клітки (CXR) для Covid 19 (Covid-датасет)" в поточному році було поставлено за **мету**:

1. Встановити та навчити DTL-19 моделі.

Для досягнення поставленої мети на Етапі №1: "Адаптація нового набору даних рентгенографії грудної клітки (CXR) для Covid 19 (Covid-датасет)" було поставлено для Субвиконавця наступні **завдання**:

1. Зробити вибір/налаштування стандартних моделей глибинного навчання (DL) на конфігурації віртуального прототипу обладнання та програмного забезпечення на основі платформи медичного застосування Nvidia Clara та моделей для периферійних обчислень (включно із Google Coral, Inel Movidius, тощо). Для кількох прикладів практичного застосування підготовити доповіді для представлення на конференціях.
2. Дослідити, розробити та протестувати: міні-модель DTL-19 з перенавчанням щільних шарів (WU1), міді-модель DTL-19 з перенавчанням деяких згорткових CNN-шарів + щільних шарів (WU1), максі-модель DTL-19 з перенавчанням усіх шарів (WU2).

РОЗДІЛ 1. ОРГАНІЗАЦІЯ АПАРАТНОГО І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для досягнення головної мети проєкту на підготовчому етапі були сформульовані завдання щодо налаштування апаратного та програмного забезпечення для організації технічної бази досліджень. Ці завдання виконувались у тісній співпраці із субвиконавцем, а саме компанією ТОВ "ТАНХОСТ", яка має багатий досвід і глибокі експертні знання в контексті встановлення, налаштування, супроводження апаратного і програмного забезпечення та дистанційного постачання сервісів для високопродуктивних обчислень.

Протягом останніх років прогрес машинного навчання (ML), особливо глибинного навчання (DL) та глибоких нейронних мереж (deep learning neural network – DNN) [18]—[20], тісно пов'язаний з недавніми досягненнями у високопродуктивних обчисленнях (high-performance computing – HPC), розподілених обчисленнях та конкретних обчислювальних архітектурах.

Вимоги до більшої точності, швидшого навчання та меншого часу передбачення також тісно пов'язані з вимогами до більшої обчислювальної потужності, більшої пам'яті, більшої пропускної здатності мережі тощо. На даний момент для цього використовуються різні IT-інфраструктури: від настільних комп'ютерів (модель спільної пам'яті) до паралельних апаратних архітектур, таких як HPC (наприклад, кластери з моделями спільної та розподіленої пам'яті), хмарних обчислень (систем із розподіленими моделями пам'яті), туманних обчислень [21] та росяних обчислень [22], [23], які особливо популярні та широко використовуються сьогодні [22]—[24] для периферійних обчислювальних пристроїв у рамках парадигми периферійних обчислень (Edge Computing – EC) [25], [26].

Еволюція цієї інфраструктури відповідає сучасній тенденції апаратно прискорених програм DL. За останнє десятиліття це було тісно пов'язано з розробкою графічних процесорів (GPU) або графічних процесорів загального призначення (GPGPU). Однак нещодавно з'явилося широке розмаїття старих та нових альтернативних платформ, починаючи від процесорів цифрових сигналів (digital signal processor – DSP) та програмовані користувачем вентильні матриці (field-programmable gate array – FPGA) [27], [28] до абсолютно нової архітектури прикладних інтегральних схем специфічного застосування (application-specific

integrated circuit – ASIC) [29], включаючи нейроморфне обладнання [30], [31], архітектури тензорної обробки (tensor processing architecture – TPA), такі як тензорні ядра (Tensor Cores – TC) від NVIDIA [32], тензорні блоки обробки або тензорні процесори (tensor processing unit – TPU) від Google [33], блоки для нейронних обчислень або нейронні процесори (neural processing unit – NPU), призначені для спеціалізованих задач обробки тензорів, які широко використовуються в програмах машинного навчання.

В контексті парадигми ЕС з'явився новий тип специфічних пристроїв типу "система на кристалі" (System on a Chip – SoC) для ефективного запуску моделей глибокого навчання на крайніх обчислювальних пристроях. Ці прискорювачі периферійних обчислень (Edge Computing Accelerators – ECA) мають очевидні переваги, такі як низька затримка, висока енергоефективність, безпека, місцевість тощо, що надає численні можливості в реальних додатках [34].

Протягом останніх років на ринку з'явилися численні прискорювачі обчислювальної техніки з відповідним SDK, включаючи NVidia Jetson Nano [35], Google Coral [36], Intel NPU та зорові процесори (Vision Processing Unit – VPU) [37] та їх похідні, такі як Horned Sungem (на базі Intel Movidius NPU) [38], HiSilicon Kirin 970 SoC з архітектурою HiAI та виділеним NPU [39].

Зазвичай вони дозволяють використовувати висновок глибинного навчання на базі CNN на крайовому шарі, підтримують неоднорідне виконання в різних прискорювачах комп'ютерного зору (включаючи GPU, NPU, NPU, VPU тощо), застосовують певну внутрішню оптимізацію для отримання максимальної продуктивності виводу та швидко розгортати програми та рішення для передбачення зору, використовуючи відповідні бібліотеки.

У цьому дослідженні ми коротко обговоримо деякі спеціалізовані ECA, висвітлимо їх впровадження в контексті ЕС, представимо результати досліджень щодо ефективності деяких популярних апаратних реалізацій у контексті ЕС та запропонуємо метод оцінки для оцінки їх ефективного використання в цей контекст.

Основною метою цієї роботи є вивчення продуктивності висновків та ефективності існуючого спеціалізованого обладнання на популярному варіанті медичного застосування для оцінки фізичної активності людей (наприклад для догляду за літніми людьми) на основі розпізнавання пози людини, що дозволяє

прогнозувати ефективність з огляду на приховані накладні витрати навіть при використанні закритих комерційних рішень.

У контексті розподілених обчислень туманні обчислення (fog computing – FC) зазвичай розглядаються як архітектура, яка використовує обчислювальні пристрої периферійного рівня, щоб виконувати значну частину обчислень, зберігання та зв'язку локально та направляти узагальнені дані через Інтернет з'єднання. Така парадигма може ефективно підтримувати величезний обсяг нових додатків інтернету речей (Internet of Things – IoT), що вимагають режиму роботи в режимі реального часу, передбачуваної продуктивності та затримки [21].

Парадигма росяних обчислень (dew computing – DC) виходить за рамки концепції обчислень Cloud-Fog до концепції мікросервісу у вертикально розподіленій обчислювальній ієрархії. У порівнянні з FC парадигма DC додатково зміщує межі обчислювальних додатків, даних та служб низького рівня з централізованих віртуальних вузлів (навіть з рівня FC) на кінцевих користувачів. Тип архітектури Cloud-Fog-Dew дозволяє розповсюджувати та обмінюватися інформацією між пристроями кінцевих користувачів (наприклад, ноутбуками, смартфонами, розумними годинниками та іншими типами розумних пристосувань, особливо носимою електронікою (wearable electronics – WE) в додатках охорони здоров'я), таким чином забезпечуючи доступ до даних навіть коли глобальне підключення до мережі недоступне. Парадигма DC також використовує локальні ресурси, які постійно підключені до мережі, такі як різні розумні гаджети та датчики. Насправді DC охоплює широкий спектр технологій, що дозволяють повсякденне використання загального призначення в робочих та побутових умовах, аж до рівня простого та спеціального обладнання та мікропослуг [22]—[24].

Останнім часом парадигма EC стає дуже популярною. FC-DC та EC забезпечують майже однакові функціональні можливості, використовуючи обчислювальні можливості в локальній мережі (local area network – LAN) для виконання обчислень та зберігання даних, які зазвичай виконувались у хмарі. Основна різниця між FC-DC та EC пов'язана з місцем, де відбувається обробка даних: EC, як правило, відбувається безпосередньо на крайніх пристроях, до яких підключені датчики / виконавчі механізми, або деяких локальних шлюзових пристроях, що знаходяться поруч із датчиками / виконавчими механізмами. Але

FC-DC переносить обробку даних на блоки обробки, які підключені до локальної мережі або до самого обладнання LAN, тому зазвичай вони віддаленіші від датчиків / виконавчих механізмів.

Парадигма EC дозволяє зберігати та обробляти дані на крайніх пристроях без завантаження на платформу CC-FC-DC із наступними перевагами: швидша обробка та аналіз даних (у реальному часі); вища безпека; низька вартість володіння та підтримки; низьке споживання енергії; низька вартість мережі; краща масштабованість; вища незалежність та інші [25], [26].

Завдяки цим перевагам, а також тому, що ресурси інтелектуального обладнання EC, як правило, обмежені, а алгоритми ШІ на базі DL часто потребують величезних обчислювальних та сховищних ресурсів, дорогого живлення та підтримки, тому легкі рішення ШІ для програм EC зараз активно досліджуються [40]. Впровадження DL-програм для пристроїв EC відрізняється від DL-програм на хмарних обчисленнях, де інтенсивні обчислення для DL можуть бути надані як послуга. Кілька досліджень були присвячені дослідженню комбінованих підходів EC / CC, так званих гібридних архітектурних крайових хмар [40].

Слід зазначити, що обчислювальні вимоги DL для пристроїв EC з обмеженими ресурсами також пов'язані з різними додатковими операціями та відповідними накладними витратами (вхід / вихід, підтримка операційної системи (ОС), підключення, спільний зв'язок тощо), які рідко розглядаються, вимірюються та обговорюються щодо реального додатку (за винятком деяких простих ілюстративних прикладів у стандартних наборах даних або за спрощених умов). У цьому контексті ефективність цих систем EC повинна бути оцінена або визначена для правильного розгортання додатків, оптимізації ресурсів та ефективного прийняття рішень. Незважаючи на наявність численних тестів пристроїв EC (як правило, надаються виробниками пристроїв EC), інфраструктур, підходів [41] та тестів реальних додатків [34], внесок накладних витрат для різних наявних конфігурацій та фактичної продуктивності EC на даний момент ретельно не вивчено.

Незважаючи на нещодавній прогрес у галузі AI, DL та EC, значні проблеми залишаються при розгортанні моделей та програм DL на інфраструктурі EC (прискорювачі, контролери, сервери та їх поєднання). Зазвичай вони пов'язані з часом затримки, часом передбачення, енергією, міграцією, оптимізацією. Не-

зважаючи на багато чудових досліджень, описаних вище, ретельний розгляд питання зменшення часу затримки та часу передбачення, проблема залишається особливо для обробки високоякісних вхідних зображень та відео. Наприклад, мінімальне споживання енергії є бажаним для програм DL на ECA, оскільки вони часто живляться від батареї [42]. Цього можна досягти, зменшивши кількість обчислень, пов'язаних з DL, але це може зменшити ефективність прогнозування. Оптимізація конкретних апаратних чіпів (наприклад, GPU і TPU) за рахунок зменшення споживання енергії без шкоди для точності є однією з найгарячіших тем у цій галузі [40], [41], [43]. Через різноманітність доступних та майбутніх ECA проблема великої міграції виникає для великих моделей DL, націлених на інфраструктуру EC. Цим проблемам було присвячено багато досліджень, і запропоновано деякі ефективні рішення на основі віртуальних машин, контейнерів та конкретних мережевих методів [44]—[47]. Для вирішення цих проблем потрібні системні вимірювання та експерименти, щоб отримати емпіричне розуміння проблем міграції. Але вплив накладних витрат на навколишнє середовище на ефективність ECA для конкретних випадків використання ще не досліджено, і пошук оптимальних конфігурацій для низьких накладних витрат та масштабного використання ECA представляє великий практичний інтерес.

У логістиці, транспорті, телекомунікаціях проблема "останньої милі" означає складність доставки товару від швидкісних сполучень до кінцевих пунктів призначення через високошвидкісні останні сполучення, коли ця ефективність страждає від цього уповільнення. Подібна ситуація може виникнути, коли високоефективне обладнання (наприклад, ECA останніх випусків) може бути застосовано до високооптимізованих моделей DL, але всі ці прискорення можуть бути зменшені накладними витратами на "останній милі", користувачі використовують неефективну підтримку, яку надає ОС за замовчуванням, застосовність режимів повільного підключення та незнання деяких конкретних рекомендацій виробника, таких як режими розгону тощо. Проблема посилюється використанням деяких застарілих компонентів в екосистемі EC, наприклад, неефективних ОС, повільного підключення через USB, незнання режимів розгону, запропонованих виробниками. Вплив належної оптимізації на продуктивність інтуїтивно очевидний, але конкретний аналіз прискорення раніше не проводився для конкретних типів ECA, моделей DL та додатків.

1.1 Конфігурація серверу із графічними прискорювачами

Для завдань проєкту було заплановано придбання високопродуктивного серверу із графічними прискорювачами широкого застосування (general purpose graphic processing unit – GPGPU, далі GPU). Для організації 2-х (і більше в перспективі подальшого розвитку проєкту на наступних етапах) робочих місць розробника було заплановано створення віртуальних машин різної конфігурації. Така організація технічної бази високопродуктивних обчислень на основі віртуалізації дозволяє забезпечити високу відмовостійкість внаслідок збоїв елементів програмного/апаратного забезпечення і енергоживлення, легку переносимість на інші фізичні платформи при оновленні фізичних компонентів апаратного забезпечення, надійну відтворюваність робочого оточення при розширенні кола розробників і переносу усієї платформи на робочі місця потенційних споживачів.

Сучасні GPU не обмежені звичайним виводом зображення або образу на екран, вони мають вбудований графічний процесор, який здатен виконувати додаткову обробку, беручи на себе задачу центрального процесору (central processing unit – CPU). GPU було розроблено для прискорення обробки графіки, сьогодні GPU здатні сильно прискорити обчислювальні процеси для глибинного навчання. Вони є невід'ємною частиною сучасної сфери машинного навчання, а нові GPU часто розробляються спеціально із розрахунку на використання їх для сфери машинного, і в особливості глибинного, навчання.

Принцип обчислень за допомогою GPU. GPU складаються з спеціальних обчислювальних ядер, які використовуються для прискорення обчислювальних процесів. Спочатку вони були призначені виключно для обробки зображень та різного роду графічних та візуальних даних.

Однак сьогоднішня тенденція полягає у впровадженні для прискорення процесів обчислення, таких як глибинне навчання. Така тенденція пов'язана з тим, що GPU здатні ефективно виконувати паралельні обчислення для масових розподілених обчислень.

Головною перевагою GPU є здатність паралельно виконувати обчислення та одночасно обробляти частини одного цілого набору даних. Для реалізації паралельних обчислень використовуються чотири основні типи архітектур:

1. Одна інструкція, один елемент даних (SISD – single instruction, single data).
2. Одна інструкція, багато елементів даних (SIMD – single instruction, multiple data).
3. Багато інструкцій, один елемент даних (MISD – multiple instructions, single data).
4. Багато інструкцій, багато елементів даних (MIMD - multiple instructions, multiple data).

Більшість процесорів мають багатоядерну архітектуру MIMD. У свою чергу GPU використовують архітектуру SIMD, що робить GPU найкращим варіантом вибору для підходів глибинного навчання, яким необхідно виконання однакових процесів для великої кількості різних даних.

Такий підхід дозволяє розподіляти процеси машинного навчання і здатний значно прискорити виконання операцій. За допомогою архітектури GPU є можливість накопичувати багато ядер, які будуть використовувати набагато менше ресурсів не знижуючи при цьому ефективність та обчислювальні можливості машини загалом.

На хмарних інфраструктурах Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського" і субвиконавця ТОВ "ТАНХОСТ" були створені кілька версій віртуальних машин (прототипів запланованого фізичного обладнання) із кількома центральними процесорами, різним обсягом оперативної пам'яті і дискових накопичувачів та різними типами GPU.

Типова послідовність дій по встановленню та налаштуванню серверу включала наступні операції (приклад серверу #NVClara):

1. На сервері було встановлено ОС Debian GNU/Linux 10 з системою віртуалізації Proxmox VE 6.2-1.
2. Були внесені зміни в налаштуваннях ОС Debian GNU/Linux 10 для передачі в віртуальну машину пристрою GPU без шару віртуалізації (GPU Passthrough). Такий спосіб організації віртуальних ресурсів дозволив отримати таку саму продуктивність, як на апаратному забезпеченні без використання віртуалізації.
3. Була створена віртуальна машина з наступними параметрами:
    (a) тип процесора: Intel(R) Xeon(R) CPU E5-2620 v4

(б) обсяг оперативної пам'яті 32GB
(в) GPU: PNY GTX 1080 TI, 11GB GDDR5x
(г) обсяг диску: 100GB

4. На віртуальній машині була встановлена ОС Ubuntu 18.04 LTS
5. Було встановлене наступне програмне забезпечення:
   (а) NVIDIA Driver release 440
   (б) NVIDIA CUDA 11.0.194
   (в) Kubernetes 1.15.4
   (г) Docker 19.03.1
   (д) nvidia-docker 2.2
   (е) Helm 2.15.2
6. Були встановлені наступні компоненти Clara Deploy SDK:
   (а) dicom
   (б) render
   (в) monitor
   (г) console
7. Був запущений еталонний приклад для перевірки роботи компонент Clara Deploy SDK.

На іншому сервері (#162) була виконана аналогічна послідовність дій і встановлена аналогічна конфігурація (із деякими варіаціями версій):

1. На сервері було встановлено ОС Debian GNU/Linux 10 з системою віртуалізації Proxmox VE 6.2-1.
2. Були внесені зміни в налаштуваннях ОС Debian GNU/Linux 10 для передачі в віртуальну машину пристрою GPU без шару віртуалізації (GPU Passthrough). Такий спосіб організації віртуальних ресурсів дозволив отримати таку саму продуктивність, як на апаратному забезпеченні без використання віртуалізації.
3. Була створена віртуальна машина з наступними параметрами:
   (а) тип процесора: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, 8 ядер
   (б) обсяг оперативної пам'яті 12GB
   (в) GPU: GeForce GTX 1080 Ti, 11GB GDDR5x
   (г) обсяг диску: 100GB
4. На віртуальній машині була встановлена ОС Ubuntu 18.04.4 LTS

5. Було встановлене наступне програмне забезпечення:
    (а) NVIDIA Driver release 450.51.06
    (б) NVIDIA CUDA 10.1.243
    (в) Tensorflow 2.0.2

На додатковому сервері (#163) була також виконана аналогічна послідовність дій і встановлена аналогічна конфігурація (із деякими варіаціями версій):

1. На сервері було встановлено ОС Debian GNU/Linux 10 з системою віртуалізації Proxmox VE 6.2-1.
2. Були внесені зміни в налаштуваннях ОС Debian GNU/Linux 10 для передачі в віртуальну машину пристрою GPU без шару віртуалізації (GPU Passthrough). Такий спосіб організації віртуальних ресурсів дозволив отримати таку саму продуктивність, як на апаратному забезпеченні без використання віртуалізації.
3. Була створена віртуальна машина з наступними параметрами:
    (а) тип процесора: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, 8 ядер
    (б) обсяг оперативної пам'яті 20GB
    (в) GPU: GeForce GTX 1080 Ti, 11GB GDDR5x
    (г) обсяг диску: 100GB
4. На віртуальній машині була встановлена ОС Ubuntu 16.04.6 LTS
5. Було встановлене наступне програмне забезпечення:
    (а) NVIDIA Driver release 440.64.00
    (б) NVIDIA CUDA 10.2.89
    (в) Tensorflow 1.14.0

Для попередньої обробки наборів даних і розробки DL моделей було встановлено, налаштовано і протестовано сукупність допоміжних Python-бібліотек.

Усі ці потужності було використано для виконання завдань даного етапу із результатами, які описані у наступних розділах. Загалом протестовані віртуальні конфігурації на загаданих вище віртуальних машинах цілком відповідають вимогам запланованого фізичного обладнання і будуть перенесені на них після закінчення тендерних процедур і отримання їх на наступних етапах проєкту.

1.2 Конфігурація серверу із прискорювачами на основі тензорних обчислень

В цьому розділі описані принципи використання прискорювачів на основі тензорних ядер, які були використані на даному етапі в конфігурації хмарної послуги на майданчику Colaboratory і платформі Kaggle TPUv3 компанії Google на безкоштовній основі в якості запланованого тестового прогону описаних в наступних розділах завдань.

Принцип обчислень за допомогою TPU. Tensor Processing Unit (TPU) – тензорний процесор, розроблений компанією Google для використання з бібліотекою машинного навчання TensorFlow. Являє собою спеціалізовану інтегральну схему, що відноситься до нейронних процесорів. В порівнянні з графічними процесорами, TPU розрахований на більшу кількість обчислень з пониженою 8-розрядною точністю при менших енергозатратах.

Технологія реалізована у вигляді матричного помножувача для 8-розрядних чисел, що керується CISC-інструкціями CPU по шині PCI Express 3.0. Принцип у тому, що у TPU використовується менше обчислювальних бітів, ніж у GPU та CPU. Він оброблює тільки ті біти, які необхідно оброблювати у той час, коли це необхідно, тобто ККД значно більший, ніж у інших нейронних пристроїв.

Тензорний процесор включає в себе наступні обчислювальні ресурси:

1. Матричний помножувач (MXU). 65536 8-бітних модулів множення і додавання для матричних операцій.
2. Уніфікований буфер (UB). 24 мегабайти SRAM, які виступають в якості регістрів.
3. Блок активації (AU) – набір функцій активації.

Щоб мати можливість контролювати роботу компонентів були розроблені високо рівневі інструкції для нейронних мереж (табл. 1.1).

Таблиця 1.1 — Список основного апаратного забезпечення TPU

TPU інструкція	**Функція**
Read_Host_Memory	Зчитування даних з пам'яті
Read_Weights	Зчитування вагів з пам'яті
MatrixMultiply / Convolve	Множення або згортка даних і вагів, накопичування результатів
Activation	Застосування функції активації
Write_Host_Memory	Запис результатів у пам'ять

Даний набір команд фокусується на основних математичних операціях, які необхідні для виводу нейронної мережі: матричного множення між вхідними даними і вагами та застосування функцій активації. Тобто, архітектура TPU інкапсулює суть обчислень нейронної мережі і може бути запрограмована для широкого спектру моделей нейронних мереж. Для того щоб його запрограмувати було створено компілятор і програмний стек, який переводить виклики API з графів TensorFlow в інструкції TPU (рис. 1.1).

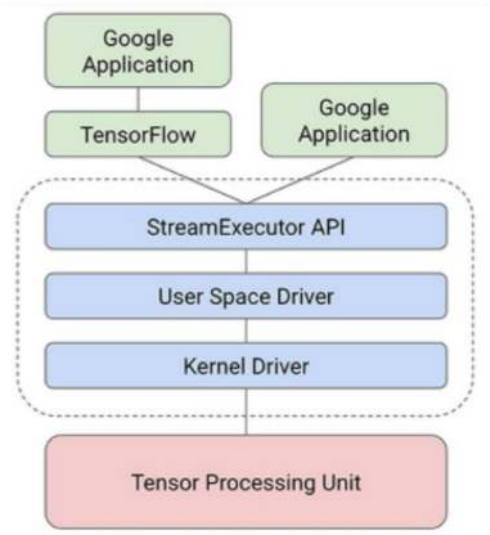


Рисунок 1.1 — Схема взаємодії TPU з програмним середовищем

Механізм множення матриць у TPU реалізований так, що можна виконувати 65536 операцій множення і додавання 8-бітних цілих чисел за кожний цикл, що значно перевищує показники GPU і CPU. Для порівняння RISC процесор без векторної архітектури здатен виконувати декілька операцій на інструкцію,

а GPU, в залежності від архітектури, біля тисячі операцій. TPU, в порівнянні з CPU та GPU, має показники на кілька порядків краще і складає сотні тисяч операцій за один цикл.

Внаслідок цих переваг TPU над традиційними CPU/GPU в даному проекті було зроблено наголос на використанні TPU у серверному і клієнтському оточенні для дистанційного і локального доступу відповідно. Результати цих досліджень в контексті медичних застосувань для деяких практичних ситуацій наведено у наступних розділах даного звіту.

На підготовчому етапі виконання проєкту використовувались потужності платформи Kaggle із апаратним забезпеченням Cloud TPUv3-8 компанії Google із наступною конфігурацією: модулі TPUv3-8 були оснащені 8 ядрами (4 процесори по 2 ядра в кожному), 128 ГБ високошвидкісної пам'яті, що загалом дозволяє досягти обчислювальну потужність у 420 TFLOPS.

1.3 Конфігурація клієнтського робочого місця із прискорювачами на основі тензорних обчислень

Зараз організації охорони здоров'я можуть надавати кращі послуги з автоматизованої діагностики на місці (CADe / CADx) завдяки поєднанню нових IT-технологій, таких як ШІ (включаючи DL [6], [48]—[50]), децентралізовану інфраструктуру (включаючи блокчейн [51], [52]) та розподілених обчислень (включаючи CC-FC-DC). За допомогою передових обчислень кінцеві користувачі охорони здоров'я (наприклад, пацієнти, лікарі, лікарні тощо) можуть використовувати потенціал поєднання IT на своїх робочих місцях.

Ця робота мотивована роботами, присвяченими медичним застосуванням, де підходи DL використовувались для вимірювання та оцінки стану здоров'я CADe / CADx різних захворювань (таких як рак [9], [53]—[55], туберкульоз [4], [56], [57], кілька інших легеневих аномалій [6], [12], [58] та ін.) і можуть бути легко перенесені в EC для кінцевих користувачів в галузі охорони здоров'я. Це стало очевидним під час поточної пандемії COVID-2019, коли з'явився величезний обсяг відповідного DL-контенту, який був ретельно розглянутий та критикований [59].

Більше того, EC з DL стає все більш популярним завдяки використанню WE для моніторингу стану здоров'я на місці та в режимі реального часу не тіль-

ки в медицині, але і в інших галузях, таких як спорт, транспорт, виробництво, відпочинок тощо [60]—[65], де використовуються численні датчики, які можна носити на тілі людини для отримання деякої інформації про стану здоров'я. В даному проєкті задача моніторингу пози людини розглядається як змодельований варіант оцінки продуктивності деяких доступних пристроїв ЕС в різних конфігураціях середовищ кінцевих користувачів.

Зараз багато персональних гаджетів можуть робити передбачення на основі багатьох моделей DL (після належної адаптації) на пристроях ЕС, таких як IoT-контролери, смартфони, WE та ін. Крім них, малопотужні та застарілі ноутбуки та настільні комп'ютери без потужних процесорних мікросхем або вбудованих апаратних прискорювачів, такі як GPU-карти, можуть бути оснащені новою категорією мікросхем (ECA [18]), спрямованих на висновки на рівні ЕС, завдяки своїй здатності виконувати обчислення для моделей DL з високим часом продуктивність (латентність, умовивід, ітерація та інші часи) і без шкоди для продуктивності прогнозування (точність, втрата тощо). Незважаючи на кілька спроб виміряти експлуатаційні характеристики таких ECA [34], [41], [66], їх характеристики часто невідомі в умовах реальних застосувань, а вплив накладних витрат при різних наявних конфігураціях і фактична ефективність ECA на даний момент не була ретельно досліджена.

Серед різних раніше згаданих ECA та відповідних SDK тут були досліджені дві широко використовувані ECA (а саме, Google Coral [36] та Horned Sungem на базі Intel Movidius NPU [38]) через їх придатність не лише для пристроїв IoT та WE, а також для ПК низького класу, як-от ноутбуки та настільні ПК.

Щоб забезпечити належну основу для порівняння, це дослідження було проведено для декількох різних програмно-апаратних конфігурацій, які є найбільш широко доступними для кінцевих користувачів з ПК низького рівня, як-от ноутбуки та настільні комп'ютери без потужних чіпів процесора або вбудованих апаратних прискорювачів, таких як GPU-карти. Ми повідомляємо про їхні результати щодо часу передбачення та часу ітерації, який є часом передбачення та всім пов'язаним із цим накладним часом (введення даних та вихід результатів). Ці результати та подальші аналітичні дані можуть бути використані для передбачуваної та дешевої оптимізації програмного / апаратного забезпечення для кінцевого користувача, щоб отримати кращу продуктивність висновків за

допомогою дешевих та широко доступних ECA на базі Google Coral, Intel Movidius та інші рішення.

Серед багатьох апаратних реалізацій, запропонованих різними виробниками для прискорення виконання моделей глибокого навчання на рівні EC, було обрано два популярних типи ECA (Google Coral та Horned Sungem) та досліджені в різних конфігураціях середовища з порівнянням їх використання на платформах CPU та GPU. У цій роботі мотивація їх вибору була продиктована їх широкою доступністю, низькою ціною, підтримкою та наявним використанням додатків з нашого фактичного досвіду: в галузі охорони здоров'я на основі рентгенологічного зображення грудної клітки та прогнозування захворювань легень [54]—[58], у спорті та повсякденному житті для розпізнавання активності [62], [63] та догляду за похилими людьми за позою людини та розпізнавання діяльності [24], [65], [66].

Google TPU. Протягом останніх років Google випустив кілька версій їх платформи Tensor Processing Unit (TPU), які можна використовувати в хмарному середовищі, відомі як Google Cloud TPU v1, v2 і v3 (https://cloud.google.com/tpu/) [33] та в середовищі EC, відоме як EdgeTPU під торговою маркою Google Coral (https://www.coral.ai).

TPU стають популярними для підвищення ефективності та швидкості DNN, оскільки згідно з доступними тестами, TPU може обробляти DNN до 30 разів швидше і може бути до 80 разів енергоефективнішим, ніж процесори або графічні процесори [28], [67]. Це можливо, оскільки TPU спеціально пристосований для обробки DNN з більшою кількістю інструкцій за цикл, ніж CPU та GPU. Більше того, TPU є унікальним TPA завдяки своїй поточній доступності як хмарна послуга лише в наступних конфігураціях: Cloud TPU v2 (180 терафлопс, 64 Гб пам'яті з високою пропускною здатністю (HBM)), Cloud TPU v3 (420 терафлопс, 128 Гб HBM), Cloud TPU v2 Pod Alpha (11,5 петафлоп, 4 ТБ HBM, 2-D тороїдальна сітчаста мережа) [28]. EdgeTPU – це специфічна інтегральна схема (ASIC), яка підтримує лише цілочисельні операції (8/16 біт) та моделі TensorFlow Lite із специфічними вимогами [4]. Серед двох варіантів EdgeTPU (Coral Dev Board і як Coral Accelerator) була використана автономна версія на базі USB (відома як Coral Accelerator).

Intel VPU. Intel випустила кілька модифікацій VPU, які фактично є малопотужними мікросхемами SoC, призначеними для прискорення розгорта-

ння глибокого навчання та програм для комп'ютерного зору. Наприклад, Intel Movidius Myriad X містить кілька процесорів та обчислювальних блоків, оптимізованих для високого паралелізму та виводу DNN. VPU доступний у різних конфігураціях. У контексті поточної роботи представляє інтерес версія на базі USB-донглів (відома як Intel Neural Compute Sticks – NCS). Наприклад, NCS розміщує Movidius Myriad X VPU з 4 Гбіт оперативної пам'яті. Цей USB-накопичувач можна підключити до будь-якого обчислювального блоку, оснащеного USB (наприклад, до окремого ноутбука чи ПК), як співпроцесор, щоб пришвидшити висновок DNN. Кілька виробників пропонують подібне рішення, оснащене одним і тим же VPU, наприклад, у цій роботі розглядається популярне рішення Horned Sungem [38], яке базується на чіпі Intel Movidius MA245x (таблиця 1.2).

Таблиця 1.2 — Параметри апаратних засобів, використаних у роботі (рис. 1.2)

Тип	**Процесор**	**Операції**	**Пам'ять**	**Інтерфейс I/O (швидкість)**	**ОС**	**Розгін**
GPU	NVIDIA 1080	0.6 TFLOPs	11 GB	PCIe	Linux	Немає
Horned Sangem / usb2	Intel Movidius MA245x	0.1 TOPs	512 MB	USB 2.0 (480 Mbps)	Linux	Немає
Horned Sangem / usb3	Intel Movidius MA245x	0.1 TOPs	512 MB	USB 3.0 (5 Gbps)	Linux	Немає
Coral Stick / usb2 / standard	Google Edge TPU	4 TOPs	8 MB	USB 2.0 (480 Mbps)	Linux / Windows	Немає / Max
Coral Stick / usb3 / standard	Google Edge TPU	4 TOPs	8 MB	USB 3.0 (5 Gbps)	Linux / Windows	Немає / Max

а) Horned Sungem б) Google Coral

Рисунок 1.2 — Приклади апаратних засобів, використаних у роботі: Horned Sungem а) та Google Coral б)

Нещодавно хмарну версію Google TPU [67] і версію Coral Dev Board EdgeTPU [34] було протестовано та ретельно проаналізовано, і ця робота присвячена USB Coral Accelerator в різних конфігураціях середовища, включаючи різні ОС, підключення, і можливий розгін. Незважаючи на наявність деяких

тестів продуктивності та великих оглядів доступних TPA, таких як Coral Dev Board версія EdgeTPU [34], Google TPU [33], TPUv2 [67]—[72], систематичні дослідження щодо їх продуктивності щодо різних середовища кінцевих користувачів (ОС, тип підключення, розгін тощо) відсутні.

Ось чому в цій роботі особлива увага була приділена різним конфігураціям середовища (ОС, тип підключення, розгін тощо) (рис. 1.2, табл. 1.2), які можуть бути легко змінені кінцевими користувачами для отримання кращої продуктивності, як це показано тут для пропонованого випадку використання завдання розпізнавання пози людини для спорту, охорони здоров'я, виробництва та інших застосувань.

Були розглянуті дві найпопулярніші ОС (Windows 10 та Ubuntu 18.0) разом із двома популярними інтерфейсами підключення вводу / виводу, а саме USB 2.0 (з максимальною швидкістю 480 мегабіт в секунду) та USB 3.0 (з максимальною швидкістю 5 гігабіт в секунду), а також розглядаються два режими розгону (стандартний - фактично відсутність розгону та максимальний), запропоновані Horned Sungem SDK.

1.4 Висновки до РОЗДІЛ 1

Налаштовано обладнання у конфігурації віртуального прототипу запланованого обладнання із графічним прискорювачем (віртуальна машина із дистанційним мережевим доступом).

Встановлено програмне забезпечення (на основі хмарних серверів та клієнтів): операційної системи Ubuntu 18, бібліотеки-оболонки Tensorflow 2.0 і необхідних додаткових бібліотек та компонентів програмного забезпечення для розробки DL-моделей, зберігання стандартних наборів даних (JSRT, ChestXray, CheXpert, ISIC, ЕЕГ, та інші), які описані у наступних розділах.

Усі конфігурації віртуального прототипу запланованого обладнання із графічним прискорювачем (віртуальна машина із дистанційним мережевим доступом) було використано для повного виконання завдань даного етапу із результатами, які описані у наступних розділах, у повній відповідності із календарним планом і технічним завданням. Загалом протестовані віртуальні конфігурації на загаданих вище віртуальних машинах цілком відповідають вимогам запла-

нованого фізичного обладнання і будуть перенесені на них після закінчення тендерних процедур і отримання їх на наступних етапах проєкту.

РОЗДІЛ 2. ВСТАНОВЛЕННЯ ТА НАВЧАННЯ МОДЕЛЕЙ ГЛИБИННОГО НАВЧАННЯ (DTL-19)

У даному розділі наводяться результати виконання роботи відповідно до наступних завдань, які були заплановані для виконання під час проєкту:

1. Зробити вибір/налаштування стандартних моделей глибинного навчання (DL) на конфігурації віртуального прототипу обладнання та програмного забезпечення на основі платформи медичного застосування Nvidia Clara та моделей для периферійних обчислень (включно із Google Coral, Inel Movidius, тощо). Для кількох прикладів практичного застосування підготовити доповіді для представлення на конференціях.
2. Дослідити, розробити та протестувати: міні-модель DTL-19 з перенавчанням щільних шарів (WU1), міді-модель DTL-19 з перенавчанням деяких згорткових CNN-шарів + щільних шарів (WU1), максі-модель DTL-19 з перенавчанням усіх шарів (WU2).

Для виконання цих завдань та інтенсифікації досліджень було сформовано кілька паралельних потоків для виконання окремих складових частин роботи, які не є залежними і не пов'язані причинно-наслідковими зв'язками, а саме вибір/налаштування стандартних моделей глибинного навчання (DL) для:

1. Фільтрації 2D рентгенограм легень.
2. Ідентифікації ознак COVID-19 із візуалізацією теплових карт.
3. Оцінки пропускної здатності моделей глибинного навчання на наявному апаратному забезпеченні.

Ці паралельні робочі потоки виконувалися на створеному на підготовчому етапі робочому вузлі 1 (work unit 1 - WU1) та вузлі 2 (work unit 2 - WU2) на встановленій робочій станції, опис яких було дано у звіті (РОЗДІЛ 1).

2.1 Вибір/налаштування моделей глибинного навчання для фільтрації 2D рентгенограм легень

2.1.1 Вступ

Завдання фільтра, який буде отримано у цій роботі – класифікувати зображення на дві категорії: chest X-ray та other.

Зображення є рентгенівським знімком грудної клітки людини (chest X-ray), якщо на ньому відображено дві легені людини (ліва та права) та було отримано за допомогою рентгенівських променів (звичайний рентген, не КТ).

Набір даних, який використовувався для створення фільтра рентгенівських знімків грудної клітки людини:

1. `train` – 10739 зображень chest X-ray та other (для навчання та валідації моделі).
2. `test`– 1800 зображень chest X-ray та other (для тестування моделі).
3. `train.csv` – файл містить назви зображень разом з розширеннями та відповідними мітками для кожного зображення з папки `train`.
4. `sample_submission.csv` – зразок файлу у правильному форматі для тестування прогнозу на зображеннях з папки `test`.

2.1.2 Підготовка оточення

Тут і далі наводяться приклади програмної реалізації окремих важливих частин компонентів платформи ШІ, які дають уявлення про спосіб імплементації запропонованих методів та способів обробки медичних зображень.

Імпортування бібліотек

На даному етапі виконується встановлення бібліотек Python (рис. 2.1), які є необхідними для реалізації та виконання наступних фрагментів коду обробки медичних даних (фрагменти коду наводяться нижче із відповідними контекстними коментарями).

```
import numpy as np
import cv2
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import os

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from sklearn.model_selection import train_test_split
from keras.callbacks import ReduceLROnPlateau
from keras import optimizers

from tensorflow.keras.applications import VGG16
```

Рисунок 2.1 — Імпорт бібліотек на основі оточення Tensorflow/Keras

Підготовка та перевірка наявної інфраструктури (рис. 2.2).

```
# Check GPUs available
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

Рисунок 2.2 — Перевірка наявної інфраструктури

2.1.3 Попередня обробка даних

Зчитування та перевірка контенту зібраних даних (рис. 2.3).

```
train_df = pd.read_csv('../input/cxray/train.csv')
train_df
```

```
                   id  label
0       xray_3029.jpeg      1
1        xray_824.jpeg      1
2        im_17241.jpeg      0
3         im_4272.jpeg      0
4       xray_4090.jpeg      1
...                ...    ...
10734     im_888.jpeg      0
10735  dif_11629.jpeg      0
10736   im_33989.jpeg      0
10737  xray_2544.jpeg      1
10738  xray_5674.jpeg      1

[10739 rows x 2 columns]
```

Рисунок 2.3 — Перевірка контенту зібраних даних

Отримання інформації про кількість медичних знімків(рис. 2.4).

```
len(train_df)
```

```
10739
```

Рисунок 2.4 — Перевірка кількості навчальних знімків

Підготовка та перевірка метаданих із інформацією про маркування медичних зображень (рис. 2.5).

```
#map training metadata
train_df2 = pd.read_csv('../input/cxray/train.csv')
mapper = {1 : "Chest X-ray", 0 : "Other"}    # map labels to readable format
train_df2["label"] = train_df2["label"].map(mapper)
train_df2
```

```
                  id        label
0      xray_3029.jpeg  Chest X-ray
1       xray_824.jpeg  Chest X-ray
2       im_17241.jpeg        Other
3        im_4272.jpeg        Other
4      xray_4090.jpeg  Chest X-ray
...               ...          ...
10734     im_888.jpeg        Other
10735  dif_11629.jpeg        Other
10736   im_33989.jpeg        Other
10737  xray_2544.jpeg  Chest X-ray
10738  xray_5674.jpeg  Chest X-ray

[10739 rows x 2 columns]
```

Рисунок 2.5 — Перевірка контенту зібраних даних

Підготовка даних до навчання: з метаданих прибирається стовпчик із ідентифікаторами знімків (id) перед передачею маркерів (label) в чисельному форматі до DL-моделі (рис. 2.6).

```
# remove id column from train_df to pass labels into model
train_label = train_df.drop(['id'], axis=1)
train_label
```

```
       label
0          1
1          1
2          0
3          0
4          1
...      ...
10734      0
10735      0
10736      0
10737      1
10738      1

[10739 rows x 1 columns]
```

Рисунок 2.6 — Підготовка даних до навчання

Підготовка даних до візуалізації результатів: з метаданих прибирається стовпчик із ідентифікаторами знімків (id) перед передачею маркерів (label) в текстовому форматі до засобів візуалізації (рис. 2.7).

```
# remove id column from train_df2 to pass labels for visualization
train_label2 = train_df2.drop(['id'], axis=1)
train_label2
```

```
             label
0      Chest X-ray
1      Chest X-ray
2            Other
3            Other
4      Chest X-ray
...            ...
10734        Other
10735        Other
10736        Other
10737  Chest X-ray
10738  Chest X-ray

[10739 rows x 1 columns]
```

Рисунок 2.7 — Підготовка даних до візуалізації результатів

Додаємо нульовий відступ по краях зображення для того, щоб зробити зображення квадратним та зберегти його пропорції(рис. 2.8).

```python
# function that add zero padding to image
# used to make image square to save picture proportions
def pad_img(img):
    ht, wd, cc= img.shape

    # min image size is 150x150
    # select bigger axle
    size = 150
    if (wd >= ht and wd > 150):
        size = wd
    if (ht >= wd and ht > 150):
        size = ht

    ww = size
    hh = size
    color = (0,0,0) #black color

    result = np.full((hh,ww,cc), color, dtype=np.uint8)

    # compute center offset
    xx = (ww - wd) // 2
    yy = (hh - ht) // 2

    # copy initial image into center of result image
    result[yy:yy+ht, xx:xx+wd] = img
    return result
```

Рисунок 2.8 — Додавання відступу по краях зображення

2.1.4 Завантаження та візуалізація даних

Завантаження тренувальної підмножини датасету (рис. 2.9).

```python
# load training set
train_set = []
for index, row in train_df.iterrows():
    # read image as numpy array
    im = cv2.imread(f"../input/cxray/train/train/{row.id}")
    # add zero padding than resize image to 150x150 using interpolation
    train_set.append(cv2.resize(pad_img(im), dsize=(150, 150), interpolation=cv2.INTER_CUBIC))
train_set = np.array(train_set) # conver train_set to numpy array
```

Рисунок 2.9 — Завантаження тренувальної підмножини датасету

```
not_transformed_images = []
indexes = []
num_random_images = 6
# read num_random_images not transformed
for index, row in train_df2.sample(num_random_images).iterrows():
    indexes.append(index)
    im = cv2.imread(f"../input/cxray/train/train/{row.id}")
    not_transformed_images.append(im)
```

Рисунок 2.10 — Обираємо зображення для візуалізації

Візуалізуємо обрані зображення до і після їх обробки (рис. 2.11).

```
# show images on plot
f, ax = plt.subplots(2,num_random_images)
f.set_size_inches(3*num_random_images, num_random_images)

for i in range(num_random_images):
    ax[0][i].set_title(train_label2['label'][indexes[i]])
    ax[0][i].imshow(not_transformed_images[i])

for i in range(num_random_images):
    ax[1][i].set_title(train_label2['label'][indexes[i]])
    ax[1][i].imshow(train_set[indexes[i]])
plt.tight_layout()
```

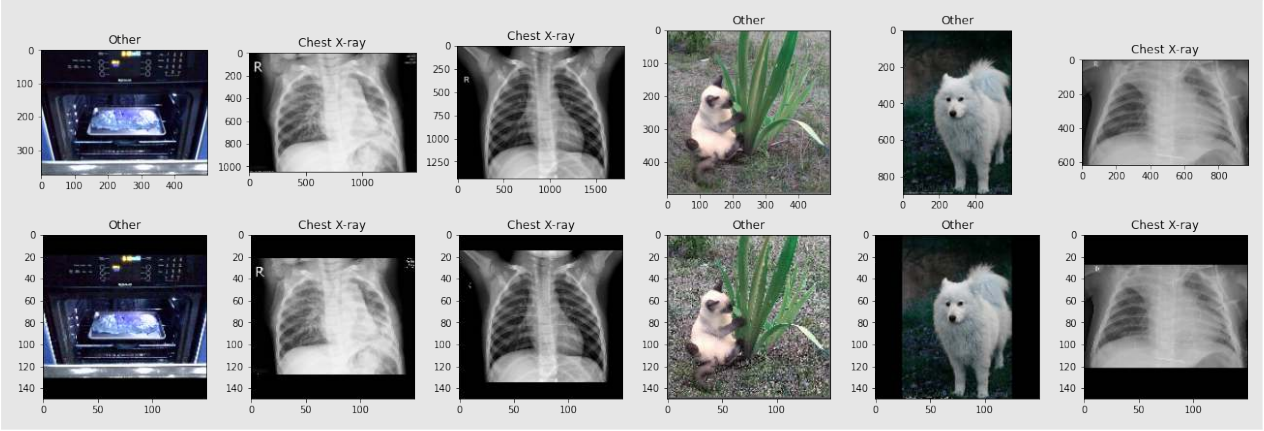


Рисунок 2.11 — Приклади візуалізації до і після обробки

Розбиття даних на навчальну та валідаційну частини:

```
# split our data into train and test, used stratify to split data more stable
X_train, X_test, Y_train, Y_test = train_test_split(train_set, train_label,
                                                    stratify=train_label,
                                                    test_size=0.2)
X_train.shape, Y_train.shape
```

```
((8591, 150, 150, 3), (8591, 1))
```

2.1.5 Підготовка моделі глибинного навчання (DL)

Вказуємо, що швидкість навчання нашої моделі буде змінюватись адаптивно (рис. 2.12).

```
# learning rate reduction callback to prevent overfitting
# it will reduce learning rate if validation accuracy stagnates
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience = 2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

Рисунок 2.12 — Адаптація швидкості навчання моделі

```
# init VGG16 model
# we will use the model with pretrained weights on ImageNet
vgg_base = VGG16(weights='imagenet',  # pretrained weights on ImageNet
                 include_top=False,   # drop the Dense layers
                 input_shape=(150, 150, 3))


model = Sequential([
    # vgg16_base model added as a layer
    vgg_base,
    # add our prediction layers
    Flatten(),
    Dropout(0.50),
    Dense(1024, activation='relu'),
    Dropout(0.20),
    Dense(512, activation='relu'),
    Dropout(0.10),
    Dense(1, activation='sigmoid')
])

# mark vgg_base as non-trainable, so training updates
# weights and biases of just our newly added layers
vgg_base.trainable = False

# use adam optimizer and accuracy as metric
model.compile(optimizer=optimizers.Adam(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

Рисунок 2.13 — Ініціалізація моделі

2.1.6 Навчання моделі глибинного навчання (DL)

```
# fit our model
model.fit(
    X_train,
    Y_train,
    epochs = 10,
    validation_data = (X_test, Y_test), # specify validation data
    callbacks = [learning_rate_reduction] # apply learning rate reduction callback
)
```

```
Epoch 1/10
269/269 [==============================] - 22s 49ms/step - loss: 0.3702 - accuracy: 0.9605 - val_loss: 9.3871e-05 - val_accuracy: 1.0000
Epoch 2/10
269/269 [==============================] - 11s 43ms/step - loss: 0.0029 - accuracy: 0.9996 - val_loss: 7.9381e-06 - val_accuracy: 1.0000
Epoch 3/10
269/269 [==============================] - 12s 43ms/step - loss: 0.0059 - accuracy: 0.9993 - val_loss: 1.5808e-04 - val_accuracy: 1.0000

Epoch 00003: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
Epoch 4/10
269/269 [==============================] - 11s 43ms/step - loss: 0.0051 - accuracy: 0.9989 - val_loss: 5.9843e-07 - val_accuracy: 1.0000
Epoch 5/10
269/269 [==============================] - 11s 43ms/step - loss: 0.0046 - accuracy: 0.9997 - val_loss: 2.4395e-06 - val_accuracy: 1.0000

Epoch 00005: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
Epoch 6/10
269/269 [==============================] - 11s 43ms/step - loss: 1.0029e-06 - accuracy: 1.0000 - val_loss: 2.2941e-06 - val_accuracy: 1.0000
Epoch 7/10
269/269 [==============================] - 11s 42ms/step - loss: 1.0480e-04 - accuracy: 1.0000 - val_loss: 7.6266e-07 - val_accuracy: 1.0000

Epoch 00007: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
Epoch 8/10
269/269 [==============================] - 11s 43ms/step - loss: 6.7274e-04 - accuracy: 0.9998 - val_loss: 6.7580e-08 - val_accuracy: 1.0000
Epoch 9/10
269/269 [==============================] - 12s 43ms/step - loss: 1.1632e-05 - accuracy: 1.0000 - val_loss: 5.2026e-08 - val_accuracy: 1.0000

Epoch 00009: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 10/10
269/269 [==============================] - 11s 43ms/step - loss: 6.7976e-04 - accuracy: 0.9998 - val_loss: 1.8685e-07 - val_accuracy: 1.0000
```

Рисунок 2.14 — Тренування мережі та вивід результатів після кожної епохи

2.1.7 Тестування моделі глибинного навчання (DL)

Підготовка файлу прогнозів DL-моделі на тестових зображеннях (рис. 2.15).

```
# read sample submision to extract photo filenams from there
sample_df = pd.read_csv('../input/cxray/sample_submission.csv')
submit_set = []
for index, row in sample_df.iterrows():
    # read image
    im = cv2.imread(f"../input/cxray/test/test/{row.id}")
    # add zero padding than resize image to 150x150 using interpolation
    submit_set.append(cv2.resize(pad_img(im), dsize=(150, 150),
                                 interpolation=cv2.INTER_CUBIC))
submit_set = np.array(submit_set) # convert to numpy array
```

Рисунок 2.15 — Підготовки файлу для збереження прогнозів

Здійснюємо тестування DL-моделі на тестових зображеннях (рис. 2.16).

```
# make prediction
predictions = model.predict(submit_set)
# round predictions to integer
predictions_float = np.rint(predictions)
predictions_string = []

# prepare string labels for submission
for i in range(0, len(predictions_float)):
    if (predictions_float[i] == 0.0):
        predictions_string.append('0')
    elif (predictions_float[i] == 1.0):
        predictions_string.append('1')

# create result pandas dataframe
result = pd.DataFrame()
result['id'] = sample_df['id']
result['label'] = predictions_string

# save submission csv
result.to_csv('submission.csv', index=False)
```

Рисунок 2.16 — Тестування DL-моделі

Перевірка контенту прогнозів моделі на тестових даних (файл + маркування) (рис. 2.17, 2.18).

```
# display submission data
result
```

```
                  id label
0         16040.jpeg     0
1         42711.jpeg     1
2         86482.jpeg     0
3         92093.jpeg     0
4          6064.jpeg     1
...              ...   ...
1795    5811795.jpeg     1
1796  218081796.jpeg     0
1797    8051797.jpeg     1
1798   21711798.jpeg     1
1799  285101799.jpeg     0

[1800 rows x 2 columns]
```

Рисунок 2.17 — Прогнози моделі на тестових даних

```python
#map predicted metadata
predictions = pd.read_csv('./submission.csv')
mapper = {1 : "Chest X-ray", 0 : "Other"}  # map labels to readable format
predictions["label"] = predictions["label"].map(mapper)
predictions
```

```
                  id        label
0         16040.jpeg        Other
1         42711.jpeg  Chest X-ray
2         86482.jpeg        Other
3         92093.jpeg        Other
4          6064.jpeg  Chest X-ray
...              ...          ...
1795    5811795.jpeg  Chest X-ray
1796  218081796.jpeg        Other
1797    8051797.jpeg  Chest X-ray
1798   21711798.jpeg  Chest X-ray
1799  285101799.jpeg        Other

[1800 rows x 2 columns]
```

Рисунок 2.18 — Представлення прогнозів моделі на тестових даних

Обираємо декілька тестових зображень для візуальної перевірки прогнозів моделі (рис. 2.19).

```python
test_images = []
test_index = []
show_test_images = 25

# read num_random_images not transformed
for index, row in predictions.sample(show_test_images).iterrows():
    test_index.append(index)
    im = cv2.imread(f"../input/cxray/test/test/{row.id}")
    test_images.append(im)
```

Рисунок 2.19 — Обираємо декілька тестових зображень для візуальної перевірки прогнозів моделі

Кількісна оцінка продуктивності фільтру на тестових даних. Метрика, яка використовувалась для оцінювання продуктивності отриманого фільтра – mean F1-Score. Внаслідок дуже характерних ознак легенів і застосування потужних мереж, фільтр показав дуже високу ефективність на рівні 100% за метрикою mean F1-Score, тобто усі 1800 тестових зображень було правильно класифіковано на дві категорії: chest X-ray та other.

2.1.8 Візуалізація результатів тестування моделі глибинного навчання (DL)

```
fig, ax = plt.subplots(show_test_images // 5, 5, figsize=(25,show_test_images))

for i in range(show_test_images):
    ax[i // 5, i % 5].set_title(f"Predicted: {predictions['label'][test_index[i]]}")
    ax[i // 5, i % 5].imshow(test_images[i])
```

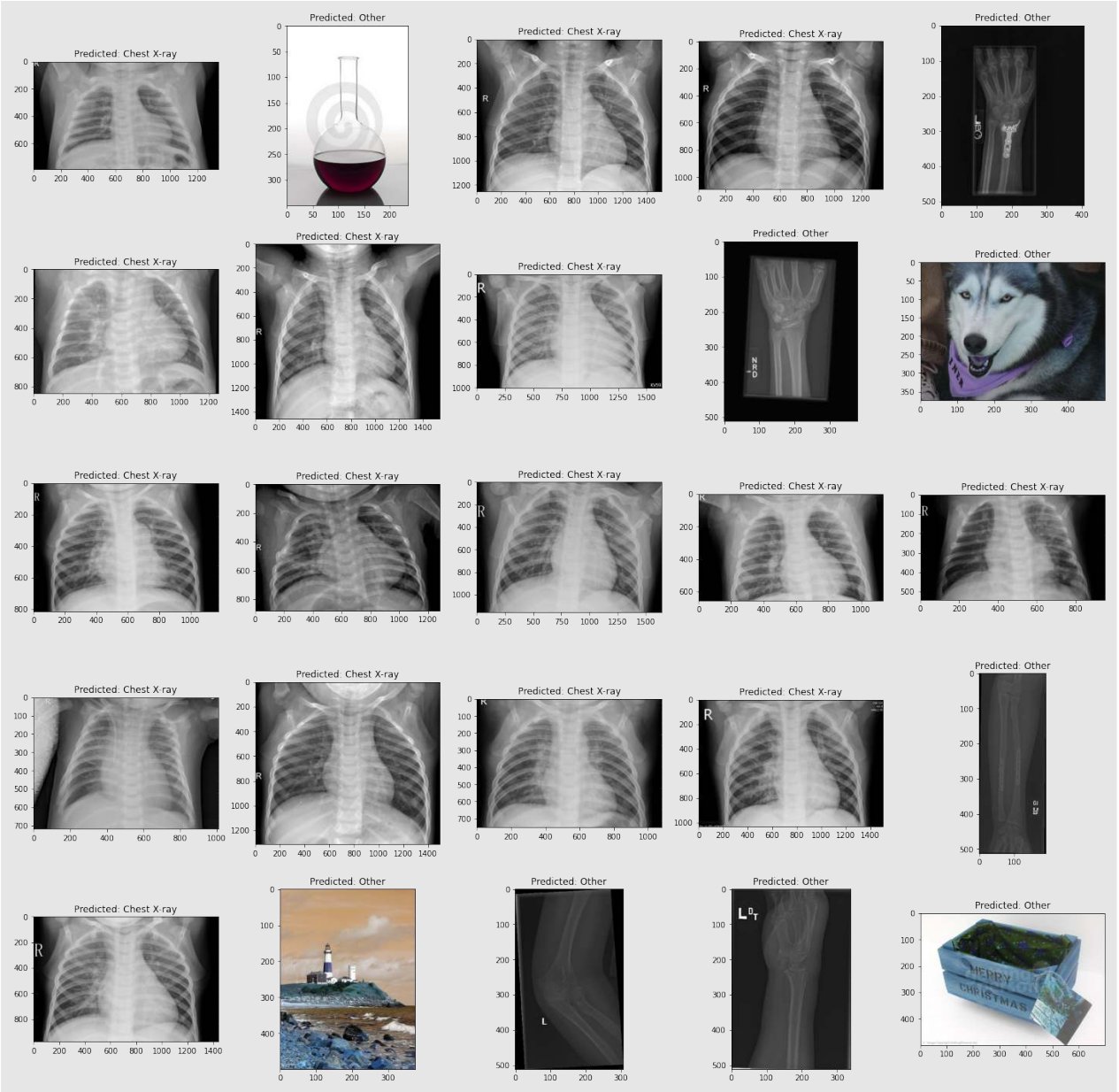


Рисунок 2.20 — Візуалізація результатів тестування моделі глибинного навчання (DL)

На рисунку 2.20 подана візуалізація декількох тестових зображень, які були обрані на попередньому кроці (див. 2.19) для візуальної перевірки прогнозів. Збереження отриманої DL-моделі фільтру (фактично на сервері проєкту):

```
model.save('chestX-rayFilter.h5')
```

2.1.9 Створення інфраструктури для використання у навчальному процесі

Для подальшого розвитку методів та способів обробки медичних зображень було створено навчальний майданчик на платформі Kaggle (рис. 2.21), який в даний момент використовується в якості навчального матеріалу (підготовки і проведення лабораторних робіт по темі даного підрозділу) на кафедрі обчислювальної техніки факультету інформатики та обчислювальної техніки КПІ імені Ігоря Сікорського.

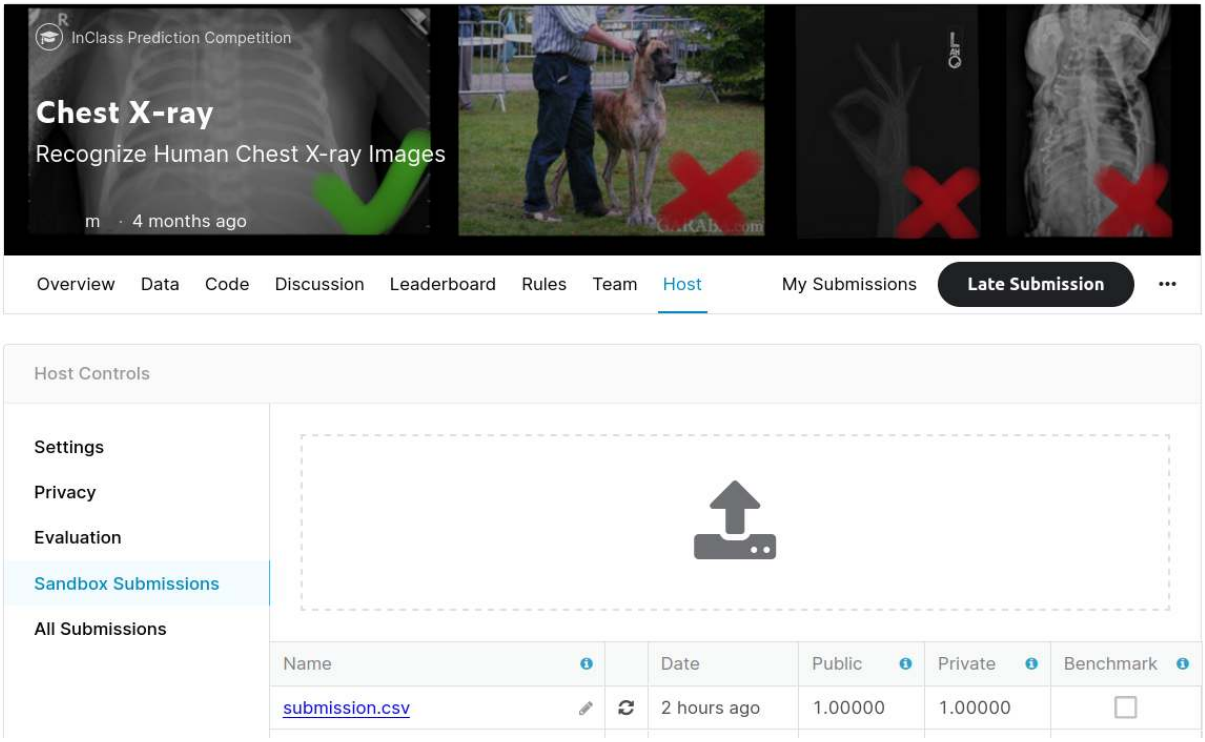


Рисунок 2.21 — Веб-сторінка навчального майданчику на платформі Kaggle із датасетом, кодами та результатами їх виконання

2.2 Вибір/налаштування моделей глибинного навчання (DL) для ідентифікації ознак COVID-19 із візуалізацією теплових карт

2.2.1 Вступ

У цьому розділі продемонстровано ідентифікацію уражених на COVID-19 легень, а також виявлення областей легень, які відповідають цьому патологічному стану. Також продемонстровано спосіб обробки та використання незбалансованого набору даних на основі розглянутого вище датасету і методу теплових карт [73].

Підготовка оточення. Імпортування бібліотек на основі оточення Torch (рис. 2.22).

```
import os
import sys
import random
import subprocess
import cv2

import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from collections import OrderedDict

import scipy
import scipy.ndimage as ndimage
import scipy.ndimage.filters as filters
from scipy.ndimage import binary_dilation
import matplotlib.patches as patches

import torch
import torch.nn as nn
import torch.backends.cudnn as cudnn
from torch.nn import functional as F
import torchvision
import torchvision.transforms as transforms
from torch.autograd import Variable
from sklearn.metrics import roc_auc_score
import torch.optim as optim
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.data import Dataset

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
from PIL import Image
```

Рисунок 2.22 — Імпортування бібліотек на основі оточення Torch

Ініціалізація генератора псевдо-випадкових чисел та інших параметрів (рис. 2.23).

```
random_seed= 42
np.random.seed(random_seed)
torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmark = True
batch_size = 16
validation_split = .34
shuffle_dataset = True
```

Рисунок 2.23 — Встановлення параметрів

Організація виконання операцій командного рядка (рис. 2.24).

```
def run_cmd(cmd, stderr=subprocess.STDOUT):
    out = None
    try:
        out = subprocess.check_output(
            [cmd],
            shell=True,
            stderr=subprocess.STDOUT,
            universal_newlines=True,
        )
    except subprocess.CalledProcessError as e:
        print(f'ERROR {e.returncode}: {cmd}\n\t{e.output}',
              flush=True, file=sys.stderr)
        raise e
    return out
```

Рисунок 2.24 — Функція для виконання операцій командного рядка

Організація копіювання (клонування) оригінального датасету (рис. 2.25).

```
def clone_data(data_root):
    clone_uri = 'https://github.com/ieee8023/covid-chestxray-dataset.git'
    if os.path.exists(data_root):
        assert os.path.isdir(data_root), \
        f'{data_root} should be cloned from {clone_uri}'
    else:
        print(
            'Cloning the covid chestxray dataset. It may take a while\n...\n',
            flush=True
        )
        run_cmd(f'git clone {clone_uri} {data_root}')
```

Рисунок 2.25 — Скачування оригінального датасету

2.2.2 Попередня обробка даних

Організація шляхів до даних локального сховища (рис. 2.26).

```
data_root = "./data"
mgpath=f'{data_root}/images',
csvpath=f'{data_root}/metadata.csv',
```

```
csvpath
```

```
('./data/metadata.csv',)
```

Рисунок 2.26 — Організація шляхів до даних локального сховища

Підвантаження набору даних (рис. 2.27).

```
clone_data(data_root)
```

```
Cloning the covid chestxray dataset. It may take a while
...
```

Рисунок 2.27 — Підвантаження набору даних

Зчитування та запис метаінформації до змінної `meta` (рис. 2.28).

```
meta = pd.read_csv('data/metadata.csv')
# pd.set_option("display.max_columns", None)
# meta.head()
```

Рисунок 2.28 — Зчитування та запис метаінформації до змінної `meta`

Вивід метаінформації про режими зйомки рентгенограм (рис. 2.29).

```
meta['view'].value_counts(dropna=False)
```

```
PA           344
AP Supine    234
AP           203
L             84
Axial         68
Coronal       16
AP Erect       1
Name: view, dtype: int64
```

Рисунок 2.29 — Проєкції у яких подані рентгенограми

Надалі увесь робочий процес навчання DL-моделі буде виконуватись на рентгенограмах, отриманих у проекції Posterior-Anterior (PA).

Вивід метаінформації про наявні типи хвороб у наборі даних[1] (рис. 2.30).

```
meta['finding'].unique()
```

```
array(['Pneumonia/Viral/COVID-19', 'Pneumonia', 'Pneumonia/Viral/SARS',
       'Pneumonia/Fungal/Pneumocystis',
       'Pneumonia/Bacterial/Streptococcus', 'No Finding',
       'Pneumonia/Bacterial/Chlamydophila', 'Pneumonia/Bacterial/E.Coli',
       'Pneumonia/Bacterial/Klebsiella', 'Pneumonia/Bacterial/Legionella',
       'Unknown', 'Pneumonia/Lipoid', 'Pneumonia/Viral/Varicella',
       'Pneumonia/Bacterial', 'Pneumonia/Bacterial/Mycoplasma',
       'Pneumonia/Viral/Influenza', 'todo', 'Tuberculosis',
       'Pneumonia/Viral/Influenza/H1N1', 'Pneumonia/Fungal/Aspergillosis',
       'Pneumonia/Viral/Herpes ', 'Pneumonia/Aspiration',
       'Pneumonia/Bacterial/Nocardia', 'Pneumonia/Viral/MERS-CoV',
       'Pneumonia/Bacterial/Staphylococcus/MRSA'], dtype=object)
```

Рисунок 2.30 — Вивід метаінформації про наявні типи хвороб у наборі даних

Підготовка датасету:

```
for x in meta['filename']:
    if x.split('.')[-1]=='gz':
        meta.drop(meta.index[meta['filename']==x],
                  inplace=True)
```

Обираємо для подальшого аналізу рентгенограми у проєкції PA з наступними патологічними станами або їх відсутністю (рис. 2.31).

```
meta = meta[
    (meta['finding'] == 'Pneumonia/Viral/COVID-19')
    |(meta['finding'] == 'Pneumonia')
    |(meta['finding'] == 'Pneumonia/Viral/SARS')
    |(meta['finding'] == 'Pneumonia/Fungal/Pneumocystis')
    |(meta['finding'] == 'Pneumonia/Bacterial/Streptococcus')
    |(meta['finding'] == 'Pneumonia/Viral/MERS-CoV')
    |(meta['finding'] == 'No Finding')
    |(meta['finding'] == 'Pneumonia/Bacterial/Klebsiella')
    |(meta['finding'] == 'Tuberculosis')]
meta = meta[meta['view']=='PA']
```

Рисунок 2.31 — Вибір для подальшого аналізу рентгенограми у проєкції PA

Аналіз кількості рентгенограм по кожному класу (рис. 2.32).

[1] Детальний розгляд відповідних хвороб та аномалій легень наведено у параграфі **??**

```
meta['finding'].value_counts(dropna=False)
```

```
Pneumonia/Viral/COVID-19              196
Pneumonia                              38
Pneumonia/Fungal/Pneumocystis          21
Pneumonia/Bacterial/Streptococcus      12
No Finding                             10
Tuberculosis                            9
Pneumonia/Viral/SARS                    8
Pneumonia/Bacterial/Klebsiella          7
Pneumonia/Viral/MERS-CoV                4
Name: finding, dtype: int64
```

Рисунок 2.32 — Аналіз кількості рентгенограм по кожному класу

Цей набір даних є незбалнсованими, так як кількість рентгенограм, які відносяться до кожного класу сильно відрізняється між собою. Іншими словами, це випадок, коли екземплярів з набору даних для одного класу значно більше ніж для іншого. Це є типовою ситуацією для медичної практики, оскільки відображає поширеність певних хвороб, локальну спеціалізацію медичних закладів, сезонні та річні коливання. Загалом це дуже сильно впливає на процес навчання моделей і порівняння результатів. Для зменшення зсуву (bias) результатів навчання моделей, зазвичай, загальною практикою є балансування таких даних.

Балансування датасету. Як видно з рисунку 2.32, кількість рентгенограм легень, які уражені COVID, значно переважає кількість рентгенограм інших класів, які представлені у наборі. Тому наступний крок буде полягати у зменшинні кількості зображень у навчальному наборі даних, які відповідають COVID. Надлишки рентгенограм з COVID, які будуть вилучені з набору даних, будуть використовуватися для виявлення областей легень, які постраждали від COVID-19 за допомогою візуалізації теплової карти.

Обираємо випадковим чином рентгенограми з COVID для подальшого використання у процесі навчання, валідації та тестування моделі (рис. 2.33).

```
X_train_val, X_test = train_test_split(
    meta[meta['finding']=='Pneumonia/Viral/COVID-19'],
    test_size=0.85,
    random_state=random_seed)
meta.drop(X_test.index, inplace=True)
meta.reset_index(drop=True, inplace=True)
```

Рисунок 2.33 — Балансування рентгенограм по COVID-19

Перевірка отриманої кількості знімків із ознаками COVID-19 для тренування та валідації:

```
X_train_val['finding'].value_counts(dropna=False)
```

```
Pneumonia/Viral/COVID-19    29
Name: finding, dtype: int64
```

Перевірка отриманої кількості знімків із ознаками COVID-19 для тестування:

```
X_test['finding'].value_counts(dropna=False)
```

```
Pneumonia/Viral/COVID-19    167
Name: finding, dtype: int64
```

Вибірку `X_test` пізніше буде використано для перевірки здатності моделі виявляти уражені зони легень за допомогою теплових карт.
Отриманий набір даних після балансування (рис. 2.34).

```
meta['finding'].value_counts(dropna=False)
```

```
Pneumonia                           38
Pneumonia/Viral/COVID-19            29
Pneumonia/Fungal/Pneumocystis       21
Pneumonia/Bacterial/Streptococcus   12
No Finding                          10
Tuberculosis                         9
Pneumonia/Viral/SARS                 8
Pneumonia/Bacterial/Klebsiella       7
Pneumonia/Viral/MERS-CoV             4
Name: finding, dtype: int64
```

Рисунок 2.34 — Набір даних після балансування

Подальша підготовка датасету – перемішування даних:

```
dataset_size = len(meta)
indices = list(range(dataset_size))
split = int(np.floor(validation_split * dataset_size))
if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]
```

Підготовка вибірок для навчання та валідації моделі:

```
train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)
```

Підготовка маркерів до унітарного кодування (one-hot encoding). Унітарний код, або прямий унітарний код (англ. one-hot) – це група бітів, серед яких дозволеними комбінаціями значень є лише ті, в яких встановлено (1) лише один біт, а всі інші вимкнено (0).

У машинному та глибинному навчанні унітарне кодування часто використовується для задач класифікації. Модель має на вхід приймати змінні, які будуть представлені у числовому форматі, тому маємо перетворити відповідні мітки рентгенограм, що відображають патологічний стан легень або вказують на відсутність ознак захворювання, з текстового формату у числовий.
Підготовка маркерів та їх загальна кількість:

```
Labels = np.array(meta['finding']).reshape(len(meta['finding']), 1)
# Labels
```

```
len(Labels)
```

```
138
```

Унітарне кодування маркерів:

```
encode = OneHotEncoder()
encode.fit(Labels)
labels_enc = encode.transform(Labels).toarray()
```

```
le = LabelEncoder()
classes = pd.DataFrame(le.fit_transform(meta['finding']), columns= ['Target'])
```

Перевірка новостворених маркерів на основі унітарного кодування (рис. 2.35).

```
classes['Target'].unique()
```

```
array([5, 1, 7, 4, 3, 0, 2, 8, 6])
```

Рисунок 2.35 — Отримані маркери унаслідок унітарного кодування

Перевірка оригінальних маркерів, які подані в текстовому форматі (рис. 2.36).

```
meta['finding'].unique()
```

```
array(['Pneumonia/Viral/COVID-19', 'Pneumonia', 'Pneumonia/Viral/SARS',
       'Pneumonia/Fungal/Pneumocystis',
       'Pneumonia/Bacterial/Streptococcus', 'No Finding',
       'Pneumonia/Bacterial/Klebsiella', 'Tuberculosis',
       'Pneumonia/Viral/MERS-CoV'], dtype=object)
```

Рисунок 2.36 — Оригінальні маркери

Перевірка на відповідність унітарних чисельних та оригінальних текстових маркерів (рис. 2.37).

```
encoded_classes = pd.DataFrame({'Actual': meta['finding'].unique(),
                                'Target': classes['Target'].unique()})
encoded_classes
```

	Actual	Target
0	Pneumonia/Viral/COVID-19	5
1	Pneumonia	1
2	Pneumonia/Viral/SARS	7
3	Pneumonia/Fungal/Pneumocystis	4
4	Pneumonia/Bacterial/Streptococcus	3
5	No Finding	0
6	Pneumonia/Bacterial/Klebsiella	2
7	Tuberculosis	8
8	Pneumonia/Viral/MERS-CoV	6

Рисунок 2.37 — Унітарні та оригінальні маркери

Підготовка рентгенограм:

1. Конвертація до єдиного формату (функція `transforms.ToPILImage`).
2. Обрізка до розміру 224×224 (функція `transforms.RandomCrop`).
3. Перетворення у тензорне представлення (функція `transforms.ToTensor`).
4. Нормалізація (функція `transforms.Normalize`).

Текстове представлення класів (категорій) рентгенограм для подальшого використання під час візуалізації даних:

```
class_names = [
    'Pneumonia/Viral/COVID-19',
    'Pneumonia',
    'Pneumonia/Viral/SARS',
    'Pneumonia/Fungal/Pneumocystis',
    'Pneumonia/Bacterial/Streptococcus',
    'Pneumonia/Viral/MERS-CoV',
    'No Finding',
    'Pneumonia/Bacterial/Klebsiella',
    'Tuberculosis',
]
```

Увесь процес підготовки даних для подальшого використання винесено у окремий клас `ChestXrayDataSet`:

```
class ChestXrayDataSet(Dataset):
    def __init__(self, csvpath, mgpath, labels_enc, transform=None):
        self.meta_data = pd.read_csv(csvpath)
        self.root_dir = mgpath
        self.labels = self.meta_data['finding']
        self.transform = transform
        for x in self.meta_data['filename']:
            if x.split('.')[-1]=='gz':
                self.meta_data.drop(self.meta_data.index[self.meta_data['filename']==x],
                                    inplace=True)

        self.meta_data = self.meta_data[(self.meta_data['finding'] =='Pneumonia/Viral/COVID-19')
                                        |(self.meta_data['finding'] == 'Pneumonia')
                                        |(self.meta_data['finding'] == 'Pneumonia/Viral/SARS')
                                        |(self.meta_data['finding'] == 'Pneumonia/Fungal/Pneumocystis')
                                        |(self.meta_data['finding'] == 'Pneumonia/Bacterial/Streptococcus')
                                        |(self.meta_data['finding'] == 'Pneumonia/Viral/MERS-CoV')
                                        |(self.meta_data['finding'] == 'No Finding')
                                        |(self.meta_data['finding'] == 'Pneumonia/Bacterial/Klebsiella')
                                        |(self.meta_data['finding'] == 'Tuberculosis')]

        self.meta_data = self.meta_data[self.meta_data['view']=='PA']
        self.meta_data.drop(X_test.index, inplace=True)
        self.meta_data.reset_index(drop=True, inplace=True)

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()
        img_name = os.path.join(self.root_dir,
                                self.meta_data.loc[idx,'filename'])
        image = Image.open(img_name).convert('RGB')
        image = np.array(image.resize((256,256)))
        image = image[:,:,0]
        image = np.uint8(((np.array(image)/255).reshape(256,256,1))*255*255)
        image = np.tile(image,3)
        label = labels_enc[idx]
        if self.transform is not None:
            image = self.transform(image)
        return image, label, idx
```

Ініціалізація екземпляру класу `ChestXrayDataSet`:

```
dataset = ChestXrayDataSet(csvpath[0], mgpath[0], labels_enc, transform)
```

Створення завантажувачів даних для навчання та валідації моделі:

```
train_loader = torch.utils.data.DataLoader(dataset,
                                           batch_size=batch_size,
                                           sampler=train_sampler)
validation_loader = torch.utils.data.DataLoader(dataset,
                                                batch_size=batch_size,
                                                sampler=valid_sampler)
```

Денормалізація рентгенограм:

```
def img_display(img):
    img = img*0.229+0.485    # unnormalize (inp = inp*std + mean)
```

```
    npimg = img.numpy()[0]
    return npimg
```

2.2.3 Візуалізація даних

Візуалізація декількох екземплярів рентгенограм з набору даних (рис. 2.38).

```
# get some random training images
dataiter = iter(train_loader)
images, labels, id_ = dataiter.next()
# Viewing data examples used for training
fig, axis = plt.subplots(3, 5, figsize=(30, 20))
for i, ax in enumerate(axis.flat):
    with torch.no_grad():
        image, label, _ = images[i], labels[i], id_[i]
        ax.imshow(img_display(image), cmap = 'gray') # add image
        ax.set(title = f"{meta['finding'][_.item()]}") # add label
```

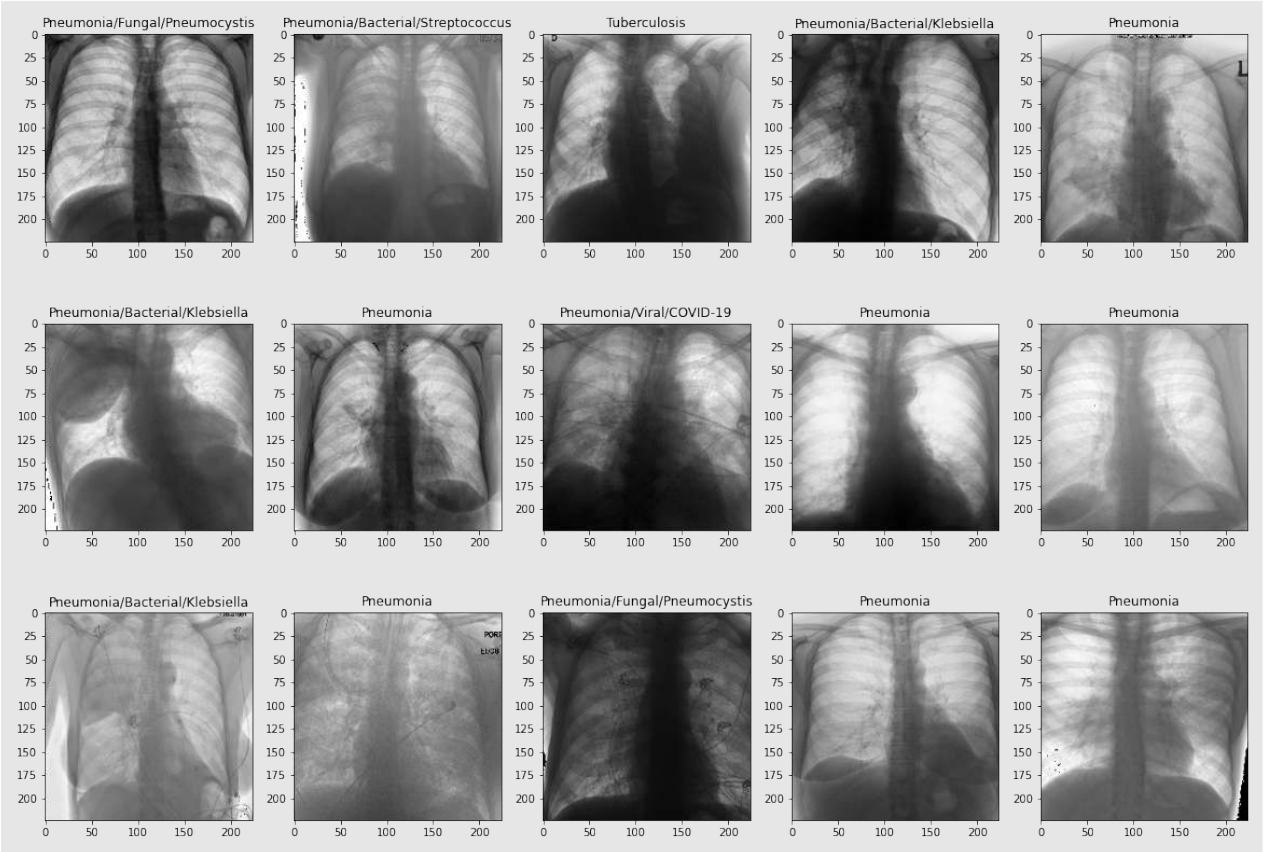


Рисунок 2.38 — Візуальне подання рентгенограм та класів, яким вони відповідають

2.2.4 Підготовка моделі глибинного навчання (DL)

Для навчання моделі, у якості демонстраційного прикладу, використано мережу із стандартною архітектурою DenseNet121 (рис. 2.39).

```
# construct model
class DenseNet121(nn.Module):
    def __init__(self, out_size):
        super(DenseNet121, self).__init__()
        self.densenet121 = torchvision.models.densenet121(pretrained=True)
        num_ftrs = self.densenet121.classifier.in_features
        self.densenet121.classifier = nn.Sequential(
            nn.Linear(num_ftrs, out_size),
            nn.Sigmoid()
        ).cuda()
    def forward(self, x):
        x = self.densenet121(x)
        return x
```

Рисунок 2.39 — Визначення DenseNet121

Визначення констант: активація використання GPU під час навчання моделі та задання кількості класів, які присутні у наборі даних:

```
cudnn.benchmark = True
N_CLASSES = 9
```

Визначення функції для підрахунку площі під кривою (AUC) – робочої характеристики приймача (ROC-крива):

```
def compute_AUCs(gt, pred):
    AUROCs = []
    gt_np = gt.cpu().numpy()
    pred_np = pred.cpu().numpy()
    for i in range(N_CLASSES):
        AUROCs.append(roc_auc_score(gt_np[:, i], pred_np[:, i]))
    return AUROCs
```

ROC-крива (англ. receiver operating characteristic, робоча характеристика приймача), або крива похибок, дозволяє оцінити якість моделі для бінарної або багатокласової класифікації, відображає співвідношення між часткою об'єктів від загальної кількості носіїв ознаки, правильно класифікованих до загальної кількості об'єктів, що не несуть ознаки, помилково класифікованих, як такі, що мають цю ознаку. Метрика AUC (англ. area under ROC curve, площа під ROC-кривою) дає кількісну інтерпретацію ROC, а саме як площа, обмежена ROC-кривою і віссю частки помилкових позитивних класифікацій. Чим вище

показник AUC, тим якісніше діє класифікатор, при цьому значення 0.5 демонструє непридатність обраного методу класифікації (відповідає звичайному вгадуванню).

Перевірка наявності спеціалізованого прискорювача на базі графічного процесора загального призначення (GP GPU). В разі відсутності спеціалізованого прискорювача (gpu) буде використано лише центральний процесор (cpu):

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)
```

```
cuda:0
```

В даному демонстраційному прикладі використовується наявний спеціалізований прискорювач (gpu) з іменем і порядковим номером (cuda:0).
Ініціалізація моделі та її завантаження з відкритого хмарного сховища:

```
# initialize and load the model
model = DenseNet121(N_CLASSES)
model = model.cuda(device)
```

```
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to /root/.cache/torch/checkpoints/densenet121-a639ec97.pth
HBox(children=(FloatProgress(value=0.0, max=32342954.0), HTML(value='')))
```

Вибір оптимізатора, задання темпу навчання та визначення функції втрат:

```
optimizer = optim.Adam(model.parameters(), lr=0.0006)
criterion = nn.BCEWithLogitsLoss()
criterion = criterion.cuda(device)
```

2.2.5 Навчання DenseNet121

Навчання моделі протягом 40 епох із підрахунком метрики AUC:

```
save_best = 0.0
for epoch in range(40):
    print("Epoch:",epoch)
    running_loss = 0.0
    for batch_idx, (data_, target_,_) in enumerate(train_loader):
        target_ = target_.type(torch.float)
        data_, target_ = data_.cuda(device), target_.cuda(device)
        optimizer.zero_grad()
        outputs = model(data_)
        loss = criterion(outputs, target_)
```

```
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        # ======== validation ========
        # switch to evaluate mode
        # initialize the ground truth and output tensor
        with torch.no_grad():
            model.eval()
            gt = torch.FloatTensor()
            gt = gt.cuda()
            pred = torch.FloatTensor()
            pred = pred.cuda()
            for i, (data_t, target_t,_t) in enumerate(validation_loader):
                target_t = target_t.type(torch.float)
                data_t, target_t = data_t.cuda(device), target_t.cuda(device)
                gt = torch.cat((gt, target_t), 0)
                input_var = Variable(data_t.view(-1, 3, 224, 224).cuda())
                output = model(input_var)
                pred = torch.cat((pred, output.data), 0)
            AUROCs = compute_AUCs(gt, pred)
            AUROC_avg = np.array(AUROCs).mean()
            print('The average AUROC is {AUROC_avg:.3f}'.format(AUROC_avg=AUROC_avg))
            if AUROC_avg>save_best:
                save_best=AUROC_avg
                torch.save(model.state_dict(), 'model_Covid-19_detection.pt')
                print('Detected network improvement, saving current model')
            for i in range(N_CLASSES):
                print('The AUROC of {} is {}'.format(class_names[i], AUROCs[i]))
        model.train()
        # print statistics
        #print('[%d] loss: %.3f' % (epoch + 1, running_loss / 715 ))
        print("================================================================================\n")
print('Finished Training')
```

Навчання моделі відбувається із підрахунком метрики AUC для кожного класу, а також середнього значення метрики AUC для усіх класів. Нижче подано результати лише першої та останньої епох навчання (рис. 2.40).

```
 Epoch: 0
The average AUROC is 0.426
Detected network improvement, saving current model
The AUROC of Pneumonia/Viral/COVID-19 is 0.7321428571428572
The AUROC of Pneumonia is 0.4361111111111111
The AUROC of Pneumonia/Viral/SARS is 0.06818181818181818
The AUROC of Pneumonia/Fungal/Pneumocystis is 0.47619047619047616
The AUROC of Pneumonia/Bacterial/Streptococcus is 0.5675675675675675
The AUROC of Pneumonia/Viral/MERS-CoV is 0.3972222222222222
The AUROC of No Finding is 0.8666666666666667
The AUROC of Pneumonia/Bacterial/Klebsiella is 0.03875968992248061
The AUROC of Tuberculosis is 0.24806201550387594
======================================================================

                                                                    ⋮


Epoch: 39
The average AUROC is 0.859
The AUROC of Pneumonia/Viral/COVID-19 is 0.7678571428571428
The AUROC of Pneumonia is 0.7944444444444445
The AUROC of Pneumonia/Viral/SARS is 1.0
The AUROC of Pneumonia/Fungal/Pneumocystis is 0.8988095238095237
The AUROC of Pneumonia/Bacterial/Streptococcus is 0.9369369369369369
The AUROC of Pneumonia/Viral/MERS-CoV is 0.7277777777777777
The AUROC of No Finding is 1.0
The AUROC of Pneumonia/Bacterial/Klebsiella is 0.9689922480620154
The AUROC of Tuberculosis is 0.6356589147286822
======================================================================
```

Рисунок 2.40 — Фрагмент виводу результатів навчання моделі після першої (Epoch: 0) та останньої епохи навчання (Epoch: 39)

Фрагмент лістингу завантаження найкращої моделі:

```
model = DenseNet121(N_CLASSES)
model = model.cuda(device)
model.load_state_dict(torch.load('model_Covid-19_detection.pt'))
model.eval()
```

2.2.6 Візуалізація результатів тестування і створення теплових мап для тестової підмножини даних

Метод Grad-CAM. Grad-CAM – це метод створення теплових карт, який застосовується до вже навченої нейронної мережі після завершення навчання та встановлення параметрів.

Grad-CAM працює шляхом:

1. Пошуку остаточного згорткового шару в мережі.
2. Вивчення інформації про градієнт, що надходить у цей шар.

Створення класу для візуалізації знімків із тепловими мапами:

```
class ChestXrayDataSet_plot(Dataset):
    def __init__(self, input_X, transform=None):
        self.data = input_X#np.uint8(test_X*255)
        self.transform = transform
        self.root_dir = mgpath[0]
        self.transform = transform

    def __getitem__(self, index):
        if torch.is_tensor(index):
            index = index.tolist()
        img_name = os.path.join(self.root_dir,self.data.loc[index,'filename'])
        image = Image.open(img_name).convert('RGB')
        image = np.array(image.resize((256,256)))
        image = np.uint8(image*255)
        image = self.transform(image)
        return image

    def __len__(self):
        return len(self.data)
```

Підготовка тестової підмножини даних. Для візуалізації теплових карт за допомогою методу Grad-CAM, буде використовуватися тестова підмножина даних `X_test`, яка була визначена раніше:

```
X_test.reset_index(drop=True, inplace=True)
test_dataset = ChestXrayDataSet_plot(input_X = X_test,
                                     transform=transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.ToTensor(),
                                        transforms.Normalize([0.485, 0.456, 0.406],
                                                             [0.229, 0.224, 0.225])
                                        ]))
```

Перевірка кількості знімків у тестовій підмножині:

```
X_test['finding'].value_counts(dropna=False)
```

```
Pneumonia/Viral/COVID-19    167
Name: finding, dtype: int64
```

Перевірка знімків тестової підмножини даних: файли та маркери:

```
X_test[['filename', 'finding']]
```

```
                                              filename  \
0                                         bb0e626a.jpg
1          88de9d8c39e946abd495b37cd07d89e5-2ee6-0.jpg
2                      ryct.2020200034.fig5-day4.jpeg
3    1.CXRCTThoraximagesofCOVID-19fromSingapore.pdf...
4                                         1052b0fe.jpg
..                                                 ...
162                                   16672_1_1.jpg
163          FE9F9A5D-2830-46F9-851B-1FF4534959BE.jpeg
164                                   16953_4_1.jpg
165          2966893D-5DDF-4B68-9E2B-4979D5956C8E.jpeg
166                                   gr1_lrg-a.jpg

                      finding
0    Pneumonia/Viral/COVID-19
1    Pneumonia/Viral/COVID-19
2    Pneumonia/Viral/COVID-19
3    Pneumonia/Viral/COVID-19
4    Pneumonia/Viral/COVID-19
..                        ...
162  Pneumonia/Viral/COVID-19
163  Pneumonia/Viral/COVID-19
164  Pneumonia/Viral/COVID-19
165  Pneumonia/Viral/COVID-19
166  Pneumonia/Viral/COVID-19

[167 rows x 2 columns]
```

PropagationBase для методу Grad-CAM:

```
print("generate heatmap ..........")
# ======= Grad CAM Function =========
class PropagationBase(object):
    def __init__(self, model, cuda=True):
        self.model = model
        self.model.eval()
        if cuda:
            self.model.cuda()
        self.cuda = cuda
        self.all_fmaps = OrderedDict()
        self.all_grads = OrderedDict()
        self._set_hook_func()
        self.image = None

    def _set_hook_func(self):
        raise NotImplementedError

    def _encode_one_hot(self, idx):
```

```
        one_hot = torch.FloatTensor(1, self.preds.size()[-1]).zero_()
        one_hot[0][idx] = 1.0
        return one_hot.cuda() if self.cuda else one_hot

    def forward(self, image):
        self.image = image
        self.preds = self.model.forward(self.image)
        #self.probs = F.softmax(self.preds, dim=1)[0]
        #self.prob, self.idx = self.preds[0].data.sort(0, True)

        return self.preds.cpu().data.numpy()

    def backward(self, idx):
        self.model.zero_grad()
        one_hot = self._encode_one_hot(idx)
        self.preds.backward(gradient=one_hot, retain_graph=True)
```

Фрагмент лістингу класу GradCAM:

```
class GradCAM(PropagationBase):
    def _set_hook_func(self):
        def func_f(module, input, output):
            self.all_fmaps[id(module)] = output.data.cpu()

        def func_b(module, grad_in, grad_out):
            self.all_grads[id(module)] = grad_out[0].cpu()

        for module in self.model.named_modules():
            module[1].register_forward_hook(func_f)
            module[1].register_backward_hook(func_b)

    def _find(self, outputs, target_layer):
        for key, value in outputs.items():
            for module in self.model.named_modules():
                if id(module[1]) == key:
                    if module[0] == target_layer:
                        return value
        raise ValueError('Invalid layer name: {}'.format(target_layer))

    def _normalize(self, grads):
        l2_norm = torch.sqrt(torch.mean(torch.pow(grads, 2))) + 1e-5
        return grads / l2_norm.item()

    def _compute_grad_weights(self, grads):
        grads = self._normalize(grads)
        self.map_size = grads.size()[2:]
        return nn.AvgPool2d(self.map_size)(grads)

    def generate(self, target_layer):
```

```
        fmaps = self._find(self.all_fmaps, target_layer)
        grads = self._find(self.all_grads, target_layer)
        weights = self._compute_grad_weights(grads)
        gcam = torch.FloatTensor(self.map_size).zero_()
        for fmap, weight in zip(fmaps[0], weights[0]):
            gcam += fmap * weight.data

        gcam = F.relu(Variable(gcam))
        gcam = gcam.data.cpu().numpy()
        gcam -= gcam.min()
        gcam /= gcam.max()
        gcam = cv2.resize(gcam, (self.image.size(3), self.image.size(2)))
        return gcam

    def FinalImage(self, gcam, raw_image):
        raw_image = raw_image*0.229+0.485
        gcam = cv2.applyColorMap(np.uint8(gcam * 255.0), cv2.COLORMAP_JET)
        gcam = np.float32(gcam) / (600)
        gcam = gcam.astype(np.float) + raw_image.numpy()[0].astype(np.float).reshape(256,256,1)
        gcam = gcam / gcam.max()

        return np.uint8(gcam * 255.0)
```

Фрагмент лістингу підготовки моделі для побудови теплових мап:

```
with torch.no_grad():
            model.eval()
            gt = torch.FloatTensor()
            gt = gt.cuda()
            pred = torch.FloatTensor()
            pred = pred.cuda()
            for i, (data_t, target_t,_t) in enumerate(validation_loader):
                target_t = target_t.type(torch.float)
                data_t, target_t = data_t.cuda(device), target_t.cuda(device)
                gt = torch.cat((gt, target_t), 0)
                input_var = Variable(data_t.view(-1, 3, 224, 224).cuda())
                output = model(input_var)
```

Створення теплових мап для тестової підмножини даних:

```
# ======== Create heatmap ===========

heatmap_output = []
image_id = []
output_class = []

thresholds = 0.15
dataiter = iter(validation_loader)
images = dataiter.next()
gcam = GradCAM(model=model, cuda=True)
for index in range(len(test_dataset)):
```

```
    input_img = Variable((test_dataset[index]).unsqueeze(0).cuda(), requires_grad=True)
    probs = gcam.forward(input_img)
    activate_classes = np.where((probs > thresholds)[0]==True)[0] # get the activated class
    for activate_class in activate_classes:
        gcam.backward(idx=activate_class)
        output = gcam.generate(target_layer="densenet121.features.denseblock4.denselayer16.conv2")
        #### this output is heatmap ####
        if np.sum(np.isnan(output)) > 0:
            print("fxxx nan")
        img = gcam.FinalImage(output, test_dataset[index])
        heatmap_output.append(img)
        image_id.append(index)
        output_class.append(activate_class)
    print("test image", str(index), "finished ", activate_class) #, activate_class, "--", "expected
```

2.2.7 Візуалізація теплових мап

Усі зображення, які подані нижче є рентгенівськими зображеннями легень з Covid-19 для яких подано візуалізацію теплових мап на основі прогнозів, зроблених моделлю.
Візуалізація теплових мап для вибірки з 50 рентгенограм:

```
heatmap_output_1 = heatmap_output[:50]
```

Нижче представлено теплові мапи та прогноз моделі для перших 50 рентгенограм з тестового набору.
Лістинг для візуалізації теплових мап для перших 50 рентгенограм з тестового набору:

```
fig, axis = plt.subplots(10,5, figsize=(26, 60))
for i, ax in enumerate(axis.flat):
    ax.imshow(heatmap_output_1[i]) # add image
    ax.set(title = f"Predicted class: {output_class[i]} \nExpected class: {encoded_classes['Target']
```

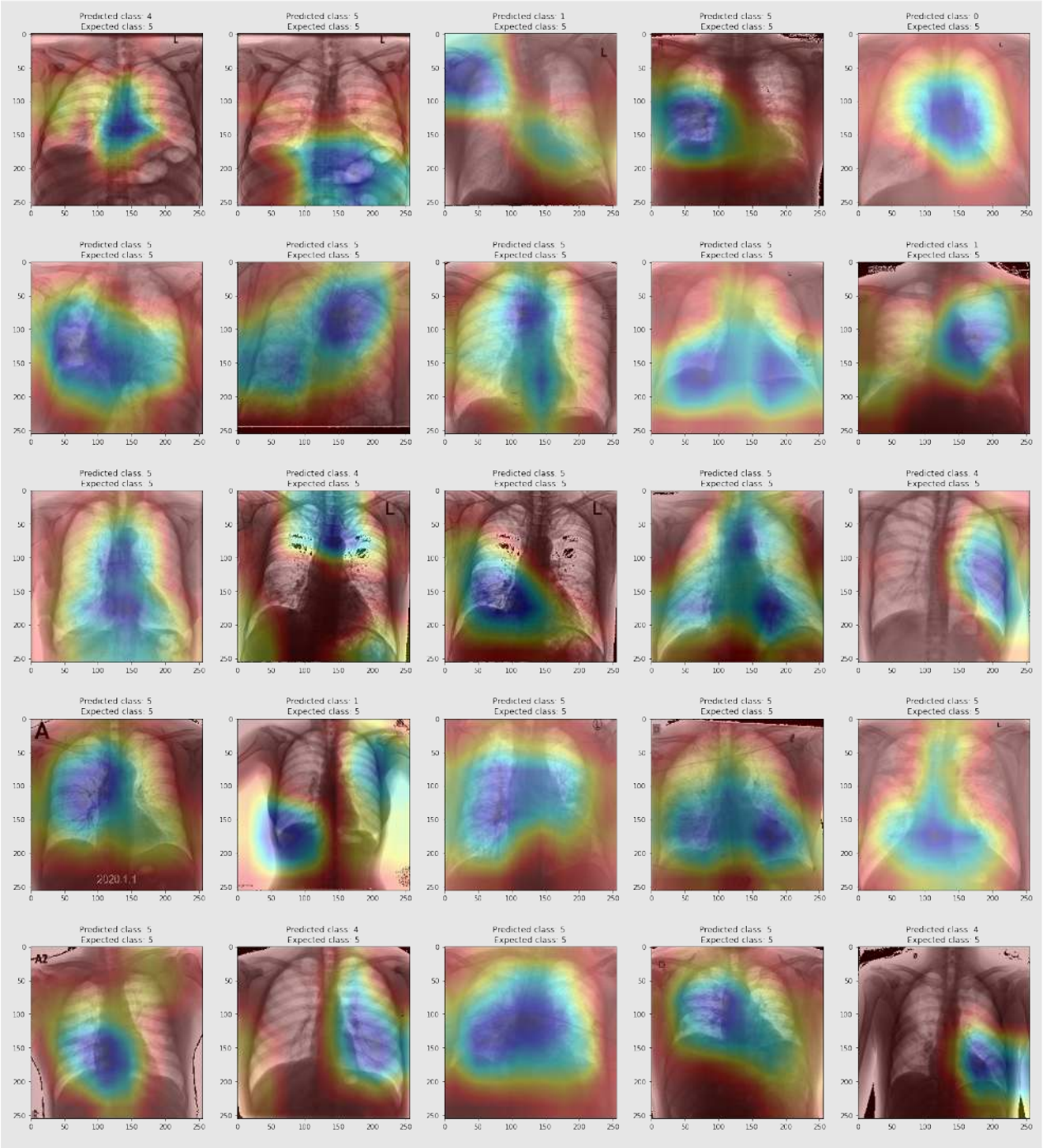


Рисунок 2.41 — Візуалізація теплових мап для перших 25 (з 50) тестових зображень

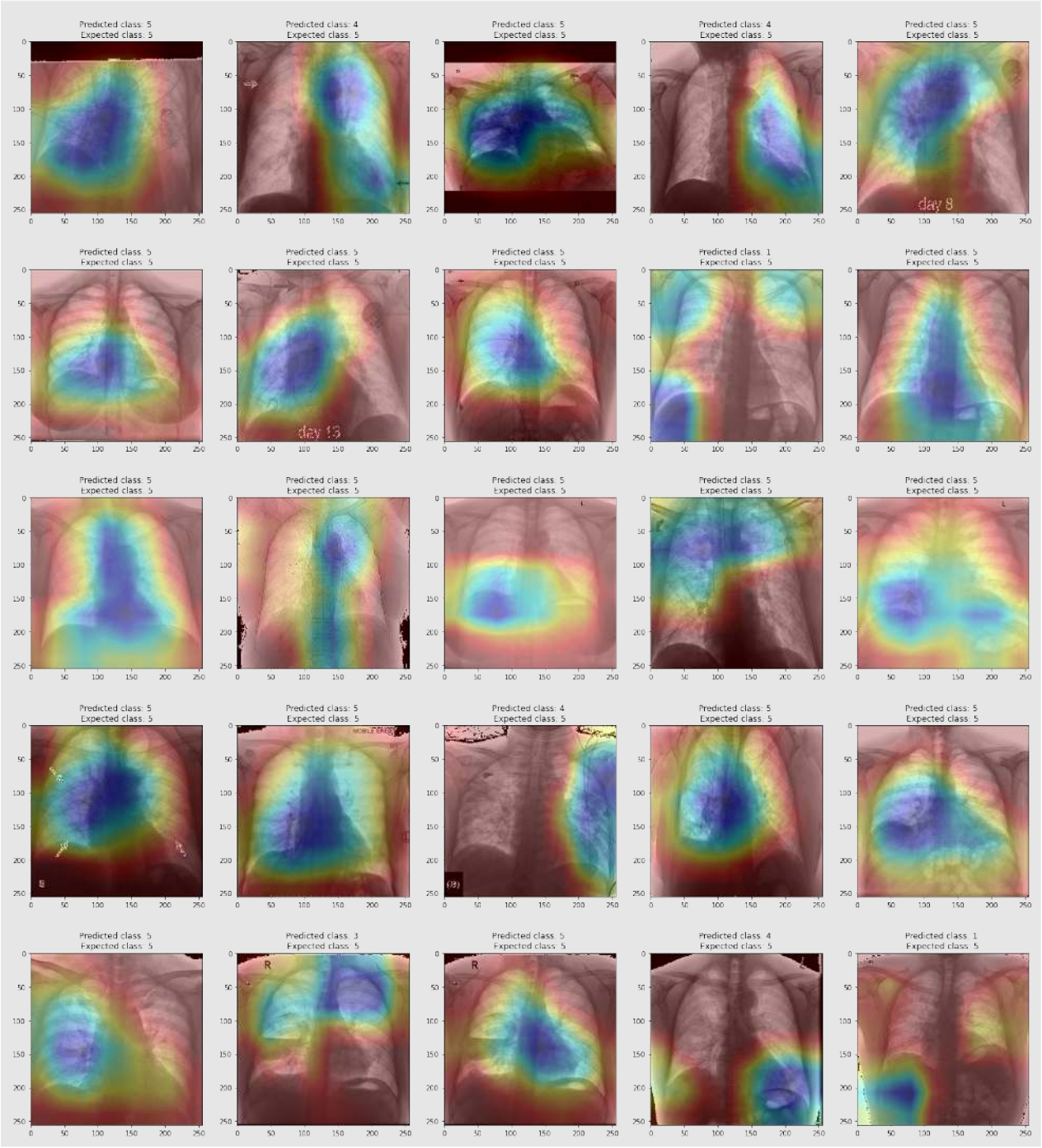


Рисунок 2.42 — Візуалізація теплових мап для наступних 25 (з 50) тестових зображень

2.3 Вибір/налаштування DL-моделей для оцінки їх пропускної здатності на наявному апаратному забезпеченні

2.3.1 Оцінка пропускної здатності моделей глибинного навчання - WU1 (Titan1080/HDD)

Для оцінки пропускної здатності моделей глибинного навчання на наявному апаратному забезпеченні (WU1), а саме сервері із графічним прискорювачем (NVIDIA GPU Titan 1080) і жорстким диском для збереження даних (HDD) було виконано вибір/налаштування/тестування зазначених вище моделей глибинного навчання (DL).

Імпорт бібліотек на основі оточення Tensorflow/Keras:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow
# Important ! For TF2!
tensorflow.compat.v1.disable_eager_execution()

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.densenet import DenseNet121
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras import preprocessing

from tensorflow.keras import models

from tqdm import tqdm

import random

import cv2
from tensorflow.keras.preprocessing import image

from sklearn.metrics import roc_auc_score, roc_curve

import time
import statistics
```

Попередня обробка знімків для візуалізації:

```
random.seed(a=None, version=2)

def get_mean_std(image_path, H=320, W=320):
    image_np = np.array(image.load_img(image_path, target_size=(H, W)))
    mean = np.mean(image_np)
    std = np.std(image_np)
    return mean, std


def load_image(img, image_dir, preprocess=True, H=320, W=320):
    """Load and preprocess image."""
    img_path = image_dir + img
    mean, std = get_mean_std(img_path, H=H, W=W)
    x = image.load_img(img_path, target_size=(H, W))
    if preprocess:
        x -= mean
        x /= std
        x = np.expand_dims(x, axis=0)
    return x
```

Метод Grad-CAM. Створення функції для методу Grad-CAM:

```
def grad_cam(input_model, image, cls, layer_name, H=320, W=320):
    """GradCAM method for visualizing input saliency."""
    y_c = input_model.output[0, cls]
    conv_output = input_model.get_layer(layer_name).output
    grads = K.gradients(y_c, conv_output)[0]

    gradient_function = K.function([input_model.input], [conv_output, grads])

    output, grads_val = gradient_function([image])
    output, grads_val = output[0, :], grads_val[0, :, :, :]

    weights = np.mean(grads_val, axis=(0, 1))
    cam = np.dot(output, weights)

    # Process CAM
    cam = cv2.resize(cam, (W, H), cv2.INTER_LINEAR)
    cam = np.maximum(cam, 0)
    cam = cam / cam.max()
    return cam
```

Функція обчислення передбачень:

```
def compute_predictions(model, img, image_dir, labels):
    preprocessed_input = load_image(img, image_dir)

    # Start timer
    ts_eval = time.time()
```

```
    predictions = model.predict(preprocessed_input)
    # End timer
    te_eval = time.time()
    test_time = (te_eval-ts_eval)

    labels_probabilities_df = pd.DataFrame({'Diseases': labels,
                                            'Probabilities': list(predictions[0])})
    print(labels_probabilities_df)
    print('Prediction time = %.3f seconds.'%(test_time))

    return test_time
```

Фрагмент лістингу для функції обчислення теплових мап:

```
def compute_gradcam(model, img, image_dir, labels, selected_labels,
                    layer_name='bn'):
    preprocessed_input = load_image(img, image_dir)
    predictions = model.predict(preprocessed_input)
    #print(predictions.shape)
    #print(list(predictions[0]))
    #print(list(predictions))

    print("Loading original image")
    fig = plt.figure(figsize=(15, 10))
    ax0 = fig.add_subplot(3,5,1)
    plt.title("Original")
    plt.axis('off')
    plt.imshow(load_image(img, image_dir, preprocess=False), cmap='gray')

    extent = ax0.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
    image_core_filename = img.split('.')[0]
    print(image_core_filename)
    plt.savefig(image_core_filename + '_original.png', bbox_inches=extent)

    jh = 2
    jv = 0
    for i in tqdm(range(len(labels))):
        if labels[i] in selected_labels:
            print("Generating gradcam for class " + labels[i])
            gradcam = grad_cam(model, preprocessed_input, i, layer_name)

            axN = fig.add_subplot(3,5,jv + jh)
            pred_round = round(predictions[0][i],3)
            plt.title( labels[i] + ": p=" + str(pred_round))
            plt.axis('off')
            plt.imshow(load_image(img, image_dir, preprocess=False),
                       cmap='gray')
            plt.imshow(gradcam, cmap='jet', alpha=min(0.5, predictions[0][i]))

```

```
            extent = axN.get_window_extent().transformed(fig.dpi_scale_trans.inverted())
            plt.savefig(image_core_filename + '_' + labels[i] + '.png', bbox_inches=extent)
            if jh>=5:
                jv += 6
                jh = 0
            else:
                jh += 1

    labels_probabilities_df = pd.DataFrame({'Diseases': labels,
                                            'Probabilities': list(predictions[0])})

    return labels_probabilities_df
```

Перелік маркерів для ознак певних аномалій або хвороб легень:

```
labels = ['Cardiomegaly',
          'Emphysema',
          'Effusion',
          'Hernia',
          'Infiltration',
          'Mass',
          'Nodule',
          'Atelectasis',
          'Pneumothorax',
          'Pleural_Thickening',
          'Pneumonia',
          'Fibrosis',
          'Edema',
          'Consolidation']
```

Визначення шляхів до моделей та тестових знімків:

```
#IMAGE_DIR = "/home/nvidia-user/ChestXray-NIHCC/DATA/images/"
IMAGE_DIR = "./images/"
IMAGE_FILENAME = '00008270_015.png'
MODEL_BASE_FILENAME = 'densenet.hdf5'
MODEL_PRETRAINED_FILENAME = 'pretrained_model.h5'

# to show the labels with the highest 4 AUC-values only:
#labels_to_show = np.take(labels, np.argsort(util_my.auc_rocs)[::-1])[:4]

# to show ALL of the labels:
labels_to_show = labels
```

Завантаження попередньо навченої моделі:

```
%%time
# create the base pre-trained model
#base_model = DenseNet121(weights='./densenet.hdf5', include_top=False)
#base_model = DenseNet121(weights=None, include_top=False)
base_model = DenseNet121(weights=MODEL_BASE_FILENAME, include_top=False)
```

```

x = base_model.output

# add a global spatial average pooling layer
x = GlobalAveragePooling2D()(x)

# and a logistic layer
predictions = Dense(len(labels), activation="sigmoid")(x)

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

# Let's load our pre-trained weights into the model now:
model.load_weights(MODEL_PRETRAINED_FILENAME)
```

Тестування з “холодного” старту оточення та апаратного забезпечення:

```
%%time
predict_time_list = []

test_time = compute_predictions(model, IMAGE_FILENAME, IMAGE_DIR, labels)
predict_time_list.append(test_time)
```

```
/home/yura/.local/lib/python3.6/site-packages/tensorflow/python/keras/engine/training.py:2325: UserWarning: `Model.state_up
  warnings.warn('`Model.state_updates` will be removed in a future version. '
             Diseases  Probabilities
0         Cardiomegaly       0.004283
1            Emphysema       0.029357
2              Effusion       0.020301
3                Hernia       0.190042
4         Infiltration       0.531233
5                 Mass       0.077701
6               Nodule       0.254761
7          Atelectasis       0.071555
8         Pneumothorax       0.031860
9   Pleural_Thickening       0.125375
10           Pneumonia       0.259928
11            Fibrosis       0.171520
12               Edema       0.005386
13       Consolidation       0.079036
Prediction time = 6.958 seconds.
CPU times: user 6.41 s, sys: 538 ms, total: 6.95 s
Wall time: 7.01 s
```

Рисунок 2.43 — Тестування з “холодного” старту оточення та апаратного забезпечення із виводом результатів на Titan1080

Тестування з “розігрітого” старту оточення та апаратного забезпечення:

```
%%time
for i in tqdm(range(100)):
    print('Image = %d'%i)
    test_time = compute_predictions(model, IMAGE_FILENAME, IMAGE_DIR, labels)
    predict_time_list.append(test_time)
```

```
  1%|          | 1/100 [00:00<00:11,  8.54it/s]
Image = 0
             Diseases  Probabilities
0         Cardiomegaly       0.004283
1            Emphysema       0.029357
2              Effusion       0.020301
3                Hernia       0.190042
4         Infiltration       0.531233
5                 Mass       0.077701
6               Nodule       0.254761
7           Atelectasis       0.071555
8         Pneumothorax       0.031860
9   Pleural_Thickening       0.125375
10           Pneumonia       0.259928
11            Fibrosis       0.171520
12               Edema       0.005386
13       Consolidation       0.079036
Prediction time = 0.077 seconds.

Image = 99
100%|      | 100/100 [00:10<00:00,  9.59it/s]
             Diseases  Probabilities
0         Cardiomegaly       0.004283
1            Emphysema       0.029357
2              Effusion       0.020301
3                Hernia       0.190042
4         Infiltration       0.531233
5                 Mass       0.077701
6               Nodule       0.254761
7           Atelectasis       0.071555
8         Pneumothorax       0.031860
9   Pleural_Thickening       0.125375
10           Pneumonia       0.259928
11            Fibrosis       0.171520
12               Edema       0.005386
13       Consolidation       0.079036
Prediction time = 0.064 seconds.
CPU times: user 14.1 s, sys: 2.01 s, total: 16.1 s
Wall time: 10.4 s
```

Рисунок 2.44 — Тестування з “розігрітого” старту оточення та апаратного забезпечення із виводом результатів на Titan1080

Підрахунок середнього значення для часу тестування одного знімку та стандартного відхилення:

```
# Calculate the mean for testing time values to list
predict_time_mean = statistics.mean(predict_time_list[1:]) # mean
# Calculate the standard deviation for testing time values
predict_time_std = statistics.stdev(predict_time_list[1:]) # standard devition
print('#### Testing time: mean = %.6f, stdev = %.6f seconds.'%(predict_time_mean,predict_time_std))
```

```
#### Testing time: mean = 0.066087, stdev = 0.005706 seconds.
```

Побудова графіку залежності часу тестування одного знімку для всієї послідовності знімків:

```
# Plot the testing time values - from 1st to the last image

plt.plot(predict_time_list)
plt.xlabel('Images')
```

```
plt.ylabel('Time (seconds)')
plt.show()
```

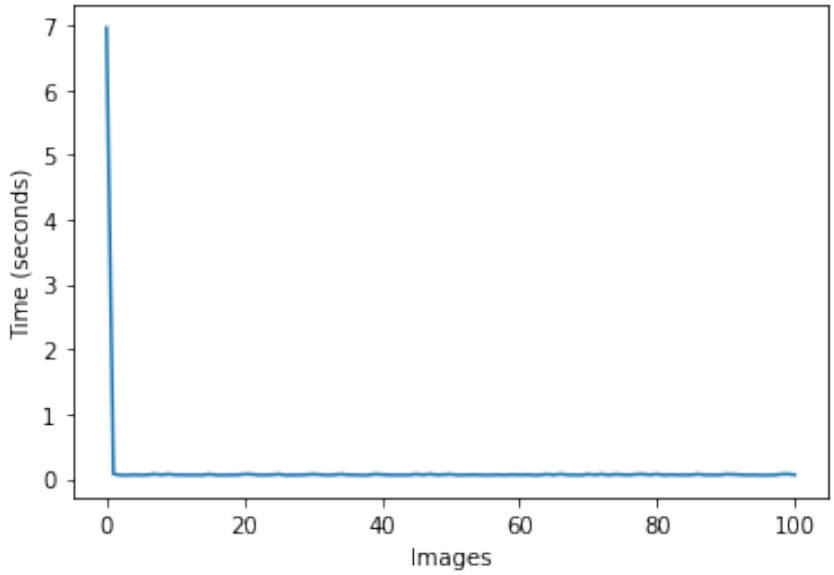


Рисунок 2.45 — Графік залежності часу тестування одного знімку для всієї послідовності знімків (“холодний” + “розігрітий” старт) на Titan1080

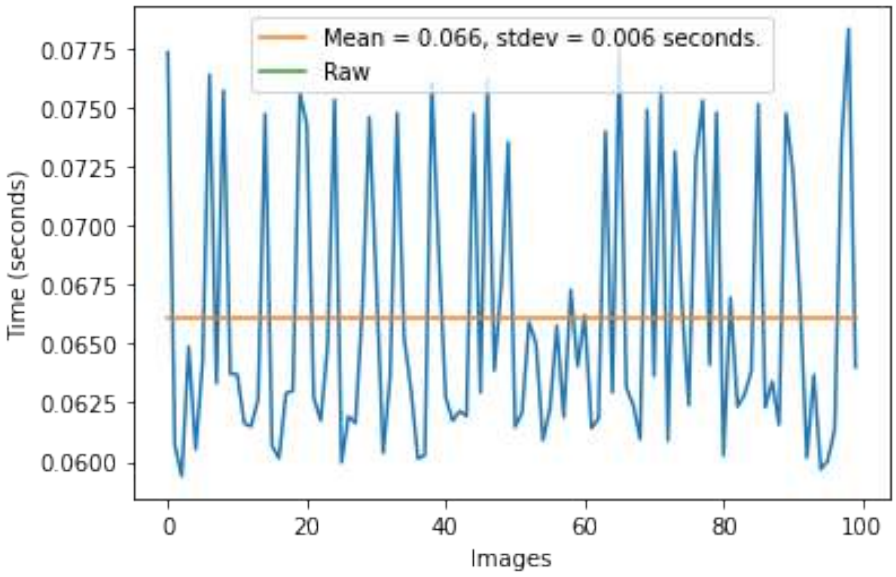


Рисунок 2.46 — Графік залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту” на Titan1080

Побудова графіку (рис. 2.46) залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту”:

```
plt.plot(predict_time_list[1:])
x_coordinates = [0, 99]
y_coordinates = [predict_time_mean, predict_time_mean]
plt.plot(x_coordinates, y_coordinates, label='Mean = %.3f,
         stdev = %.3f seconds.'%(predict_time_mean,predict_time_std))
plt.plot(predict_time_mean, label='Raw')
plt.xlabel('Images')
plt.ylabel('Time (seconds)')
plt.legend(loc='upper center')
plt.show()
```

Побудова теплових мап:

```
%%time

labels_probabilities_df = compute_gradcam(model, IMAGE_FILENAME, IMAGE_DIR,
                                          labels, labels_to_show)
```

```
100%|      | 14/14 [01:15<00:00,  5.43s/it]
CPU times: user 1min 16s, sys: 975 ms, total: 1min 17s
Wall time: 1min 16s
```

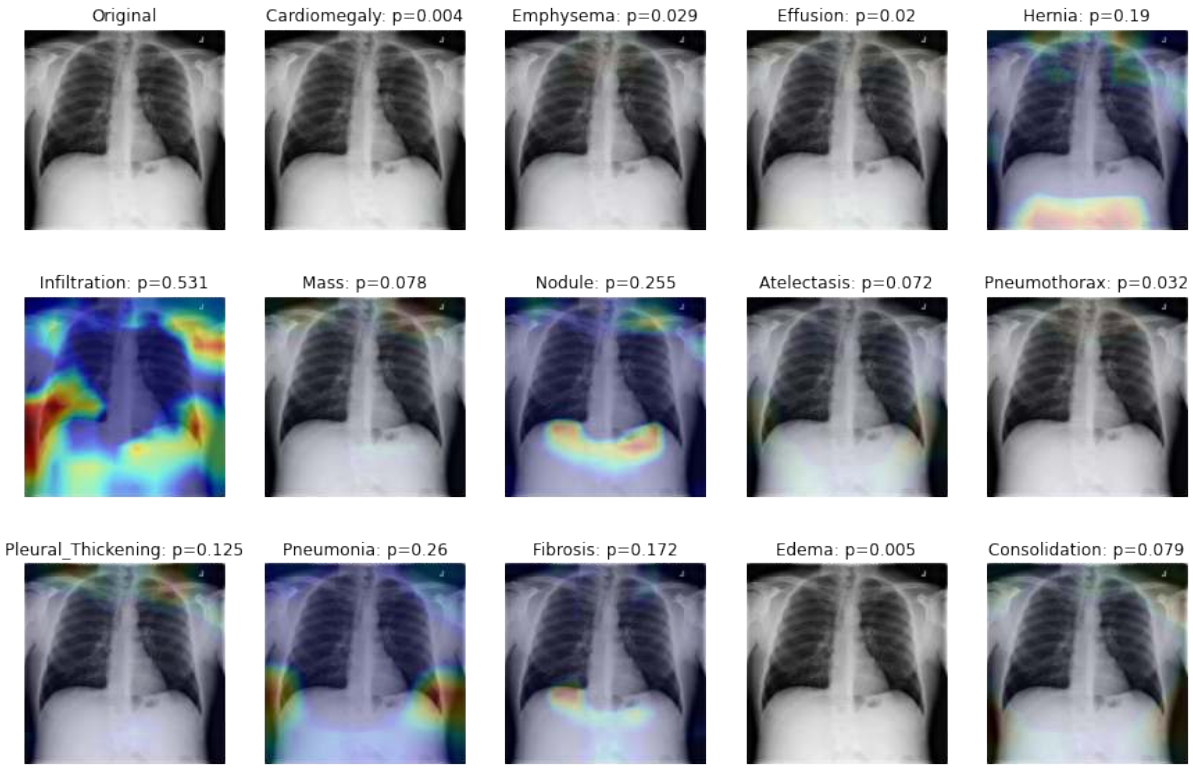


Рисунок 2.47 — Результатами візуалізації теплових мап на Titan1080

Тестування для міні-пакетів (mini-batch) різного розміру (BATH_SIZE):

```
%%time

predict_time_batch_list = []

for i in tqdm(range(1,129)):
    BATCH_SIZE = i
    print('BATCH_SIZE = %d'%BATCH_SIZE)
    test_time = compute_predictions_batch(model,
                                          IMAGE_FILENAME,
                                          IMAGE_DIR,
                                          labels)
    predict_time_batch_list.append(test_time)
```

```
  0%|          | 0/128 [00:00<?, ?it/s]
BATCH_SIZE = 1
  1%|          | 1/128 [00:07<16:26,  7.77s/it]
Batch size = 1.
Prediction time = 0.0607 seconds.
Images (total) = 128.
...
BATCH_SIZE = 128
100%|      | 128/128 [16:28<00:00,  7.73s/it]
Batch size = 128.
Prediction time = 0.0599 seconds.
Images (total) = 128.
CPU times: user 26min 32s, sys: 5min 37s, total: 32min 9s
Wall time: 16min 28s
```

Підрахунок середнього значення для часу тестування одного знімку та стандартного відхилення:

```
# Calculate the mean for testing time values to list
predict_time_mean = statistics.mean(predict_time_batch_list) # mean
# Calculate the standard deviation for testing time values
predict_time_std = statistics.stdev(predict_time_batch_list) # standard devition
print('#### Testing time: mean = %.6f, stdev = %.6f seconds.'%(predict_time_mean,predict_time_std))
```

```
#### Testing time: mean = 0.060337, stdev = 0.000493 seconds.
```

Побудова графіку залежності часу тестування одного знімку для різних значень розміру міні-пакету:

```
plt.plot(predict_time_batch_list)
x_coordinates = [0, 128]
y_coordinates = [predict_time_mean, predict_time_mean]
plt.plot(x_coordinates, y_coordinates, label='Mean = %.4f,
         stdev = %.4f seconds.'%(predict_time_mean,predict_time_std))
plt.plot(predict_time_mean, label='Raw')
plt.xlabel('Images')
```

```
8  plt.ylabel('Time (seconds)')
9  plt.legend(loc='upper center')
10 plt.title('GPU Titan 1080, HDD')
11 plt.show()
```

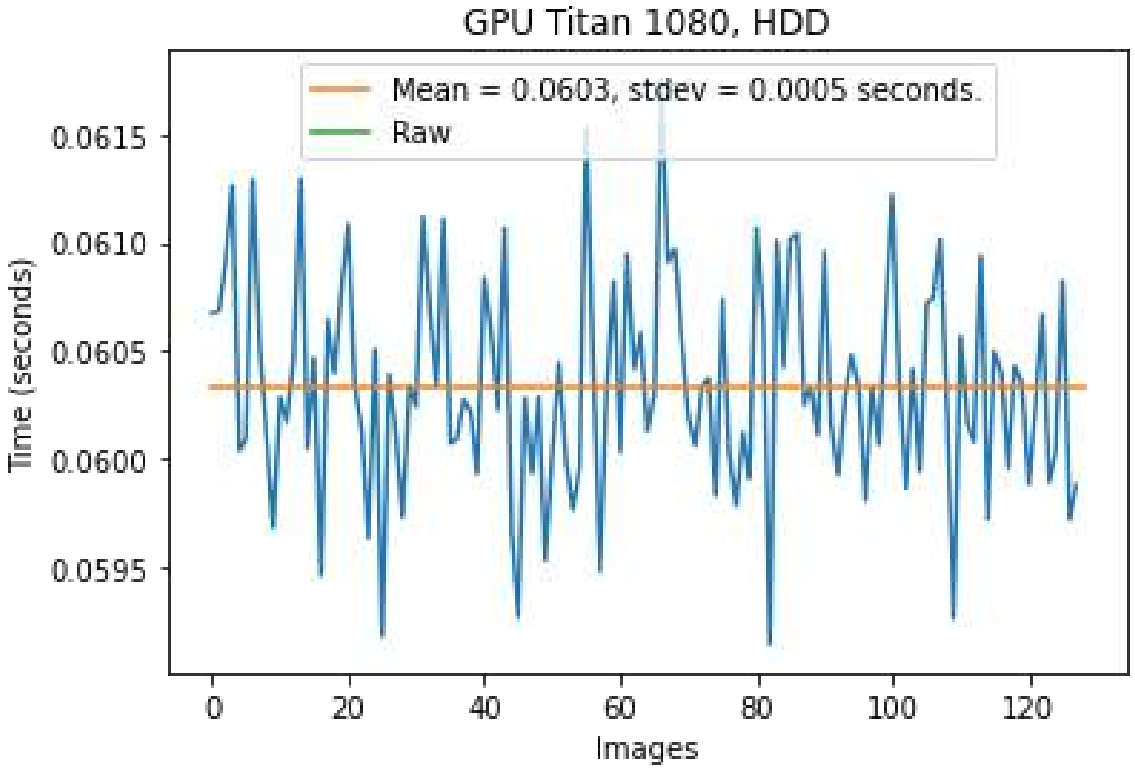


Рисунок 2.48 — Графік залежності часу тестування одного знімку для різних значень розміру міні-пакету на Titan1080

Таким чином в результаті тестування виявлено практичну можливість отримувати прогнози щодо наявності чи відсутності ознак певних легеневих хвороб або аномалій на рівні $\sim 14-16$ зображень в секунду на даному програмному оточенні та апаратному забезпеченні (GPU Titan 1080, HDD). На даному етапі роботи це відповідає поставленій фінальній меті проєкту обробляти від 10 до 100 зображень в секунду. Варто зауважити, що ці результати отримані без застосування інших додаткових засобів та способів оптимізації обрахунку моделей, які планується застосувати на наступному етапі проєкту. Додатковий потенціал покращення передбачається отримати за рахунок використання нового обладнання (детально про це описано в наступному підрозділі).

2.3.2 Оцінка пропускної здатності моделей глибинного навчання – WU2 (A100/SSD)

Апаратне забезпечення – карта NVIDIA GPU A100 + SSD

Встановлене та протестоване на підготовчому етапі апаратне забезпечення було використано на даному етапі для вимірів часу тестування знімків в залежності від розміру міні-пакету.

```
! nvidia-smi
```

```
Mon Aug 23 11:55:02 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 465.19.01    CUDA Version: 11.3     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-PCI...  On   | 00000000:01:00.0 Off |                    0 |
| N/A   29C    P0    33W / 250W |   2926MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1251      G   /usr/lib/xorg/Xorg                  4MiB |
|    0   N/A  N/A     24990      C   /usr/bin/python3                 2919MiB |
+-----------------------------------------------------------------------------+
```

Рисунок 2.49 — Конфігурація нового апаратного забезпечення (NVIDIA GPU A100)

Наступні етапи є аналогічними етапам попередньо розглянутої конфігурації апаратного забезпечення (карта NVIDIA GPU Titan 1080 + HDD) для яких в лістингах подана програмна реалізація, тому далі будуть розглянуті лише блоки, які відповідають з продуктивність моделі на карті NVIDIA GPU A100 + SSD.

Тестування з “холодного” старту оточення та апаратного забезпечення:

```
%%time
predict_time_list = []
```

```
3
4  test_time = compute_predictions(model, IMAGE_FILENAME, IMAGE_DIR, labels)
5  predict_time_list.append(test_time)
```

```
/home/yura/.local/lib/python3.6/site-packages/tensorflow/python/keras/engine/training.py:2325: UserWarning: `Model.state_up
  warnings.warn('`Model.state_updates` will be removed in a future version. '
             Diseases  Probabilities
0         Cardiomegaly       0.007493
1            Emphysema       0.137789
2             Effusion       0.061149
3               Hernia       0.214248
4         Infiltration       0.379196
5                 Mass       0.112850
6               Nodule       0.318557
7          Atelectasis       0.138213
8         Pneumothorax       0.171010
9   Pleural_Thickening       0.269736
10           Pneumonia       0.323991
11            Fibrosis       0.507190
12               Edema       0.041926
13       Consolidation       0.178292
Prediction time = 21.305 seconds.
CPU times: user 6.61 s, sys: 1.67 s, total: 8.28 s
Wall time: 21.3 s
```

Рисунок 2.50 — Тестування з “холодного” старту оточення та апаратного забезпечення із виводом результатів на A100

Тестування з “розігрітого” старту оточення та апаратного забезпечення:

```
1  %%time
2  for i in tqdm(range(100)):
3      print('Image = %d'%i)
4      test_time = compute_predictions(model, IMAGE_FILENAME, IMAGE_DIR, labels)
5      predict_time_list.append(test_time)
```

```
  3%|         | 3/100 [00:00<00:03, 24.94it/s]
Image = 0
             Diseases  Probabilities
0         Cardiomegaly       0.007493
1            Emphysema       0.137789
2             Effusion       0.061149
3               Hernia       0.214248
4         Infiltration       0.379196
5                 Mass       0.112850
6               Nodule       0.318557
7          Atelectasis       0.138213
8         Pneumothorax       0.171010
9   Pleural_Thickening       0.269736
10           Pneumonia       0.323991
11            Fibrosis       0.507190
12               Edema       0.041926
13       Consolidation       0.178292
Prediction time = 0.036 seconds.
```

```
Image = 99
             Diseases  Probabilities
0         Cardiomegaly       0.007493
1            Emphysema       0.137789
2             Effusion       0.061149
3               Hernia       0.214248
4         Infiltration       0.379196
5                 Mass       0.112850
6               Nodule       0.318557
7          Atelectasis       0.138213
8         Pneumothorax       0.171010
9   Pleural_Thickening       0.269736
10           Pneumonia       0.323991
11            Fibrosis       0.507190
12               Edema       0.041926
13       Consolidation       0.178292
Prediction time = 0.032 seconds.
CPU times: user 4.05 s, sys: 268 ms, total: 4.32 s
Wall time: 3.71 s
```

Рисунок 2.51 — Тестування з “розігрітого” старту оточення та апаратного забезпечення із виводом результатів на A100

Підрахунок середнього значення для часу тестування одного знімку та стандартного відхилення:

```
# Calculate the mean for testing time values to list
predict_time_mean = statistics.mean(predict_time_list[1:]) # mean
# Calculate the standard deviation for testing time values
predict_time_std = statistics.stdev(predict_time_list[1:]) # standard devition
print('#### Testing time: mean = %.6f, stdev = %.6f seconds.'%(predict_time_mean,predict_time_std))
```

```
#### Testing time: mean = 0.030049, stdev = 0.001946 seconds.
```

Побудова графіку залежності часу тестування одного знімку для всієї послідовності знімків:

```
# Plot the testing time values - from 1st to the last image

plt.plot(predict_time_list)
plt.xlabel('Images')
plt.ylabel('Time (seconds)')
plt.show()
```

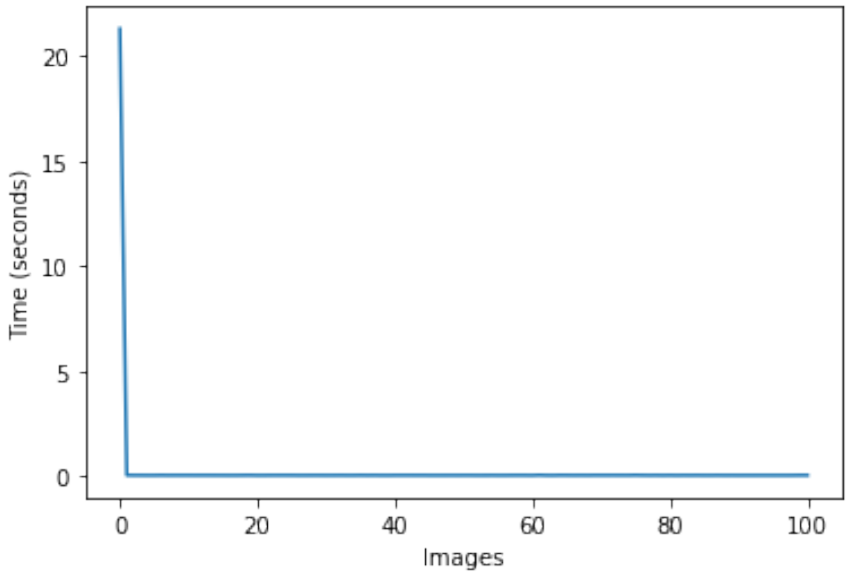


Рисунок 2.52 — Графік залежності часу тестування одного знімку для всієї послідовності знімків (“холодний” + “розігрітий” старт) на A100

Побудова графіку (рис. 2.53) залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту”:

```
plt.plot(predict_time_list[1:])
x_coordinates = [0, 99]
y_coordinates = [predict_time_mean, predict_time_mean]
plt.plot(x_coordinates, y_coordinates, label='Mean = %.3f, stdev = %.3f seconds.'%(predict_time_mean
```

```
plt.plot(predict_time_mean, label='Raw')
plt.xlabel('Images')
plt.ylabel('Time (seconds)')
plt.legend(loc='upper center')
plt.show()
```

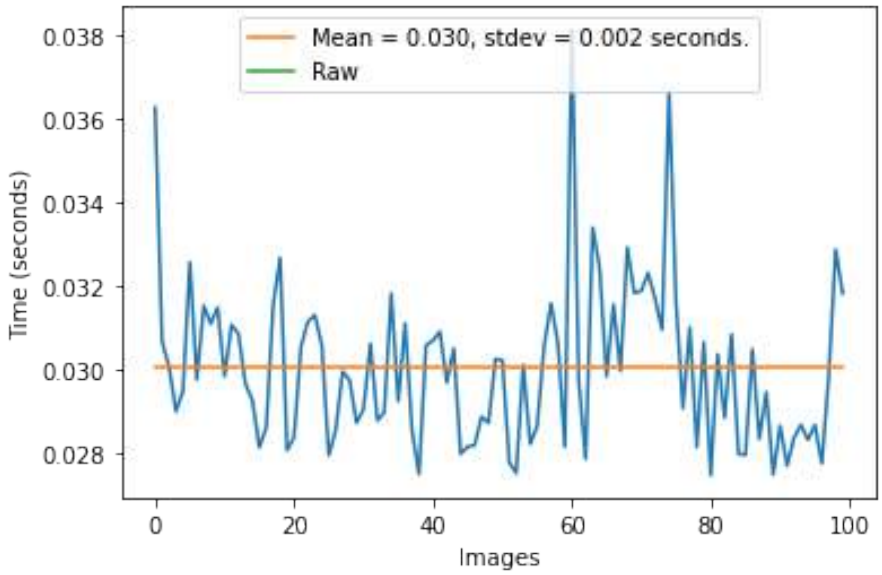


Рисунок 2.53 — Графік залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту” на A100

Побудова теплових мап:

```
%%time

labels_probabilities_df = compute_gradcam(model, IMAGE_FILENAME, IMAGE_DIR,
                                          labels, labels_to_show)
```

```
100%|      | 14/14 [00:58<00:00,  4.20s/it]
CPU times: user 59 s, sys: 1.14 s, total: 1min
Wall time: 58.9 s
```

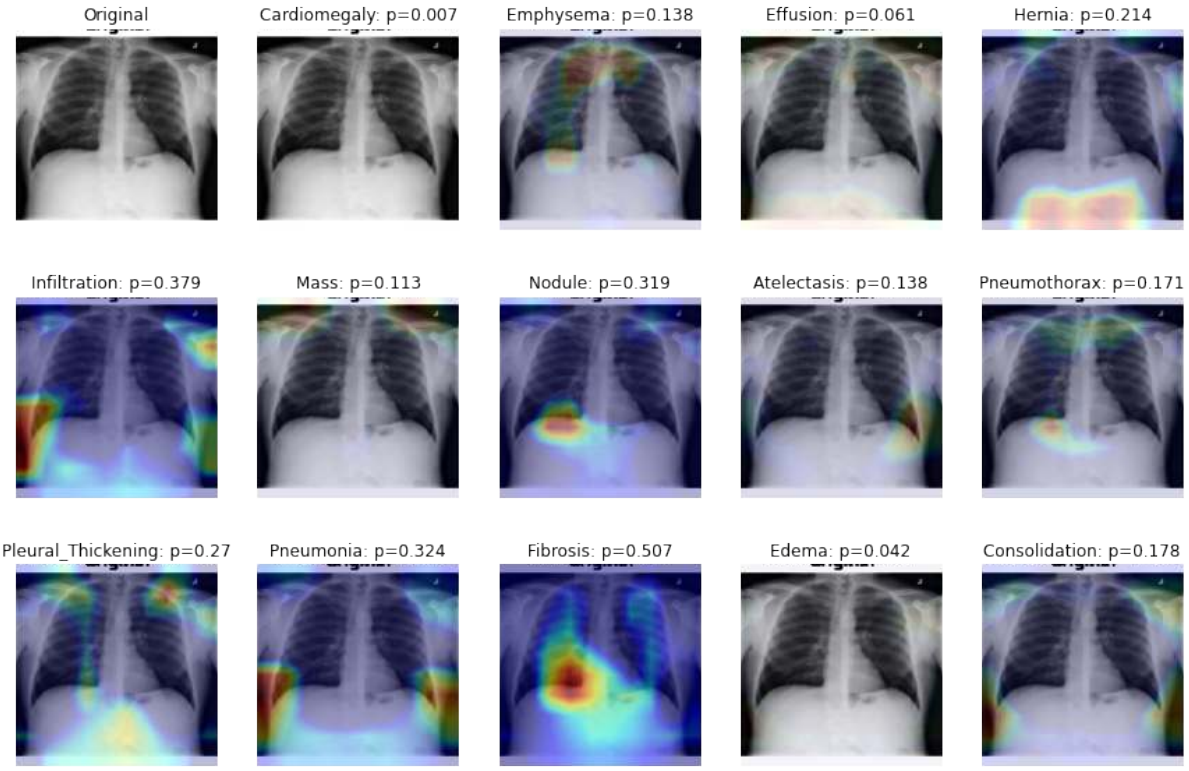


Рисунок 2.54 — Результатами візуалізації теплових мап на A100

Тестування для міні-пакетів (mini-batch) різного розміру (BATH_SIZE):

```
%%time

predict_time_batch_list = []

for i in tqdm(range(1,129)):
    BATCH_SIZE = i
    print('BATCH_SIZE = %d'%BATCH_SIZE)
    test_time = compute_predictions_batch(model, IMAGE_FILENAME, IMAGE_DIR, labels)
    predict_time_batch_list.append(test_time)
```

```
  0%|          | 0/128 [00:00<?, ?it/s]
BATCH_SIZE = 1
  1%|          | 1/128 [00:03<07:25,  3.51s/it]
Batch size = 1.
Prediction time = 0.0274 seconds.
Images (total) = 128.


BATCH_SIZE = 128
100%|      | 128/128 [07:35<00:00,  3.56s/it]
Batch size = 128.
Prediction time = 0.0275 seconds.
Images (total) = 128.
CPU times: user 9min 11s, sys: 49.8 s, total: 10min 1s
Wall time: 7min 35s
```

Підрахунок середнього значення для часу тестування одного знімку та стандартного відхилення:

```
# Calculate the mean for testing time values to list
predict_time_mean = statistics.mean(predict_time_batch_list) # mean
# Calculate the standard deviation for testing time values
predict_time_std = statistics.stdev(predict_time_batch_list) # standard devition
print('#### Testing time: mean = %.6f, stdev = %.6f seconds.'%(predict_time_mean,predict_time_std))
```

```
#### Testing time: mean = 0.027765, stdev = 0.000471 seconds.
```

Побудова графіку залежності часу тестування одного знімку для різних значень розміру міні-пакету:

```
plt.plot(predict_time_batch_list)
x_coordinates = [0, 128]
y_coordinates = [predict_time_mean, predict_time_mean]
plt.plot(x_coordinates, y_coordinates, label='Mean = %.4f, stdev = %.4f seconds.'%(predict_time_mean
plt.plot(predict_time_mean, label='Raw')
plt.xlabel('Images')
plt.ylabel('Time (seconds)')
plt.legend(loc='upper center')
plt.title('GPU A100, SSD')
plt.show()
```

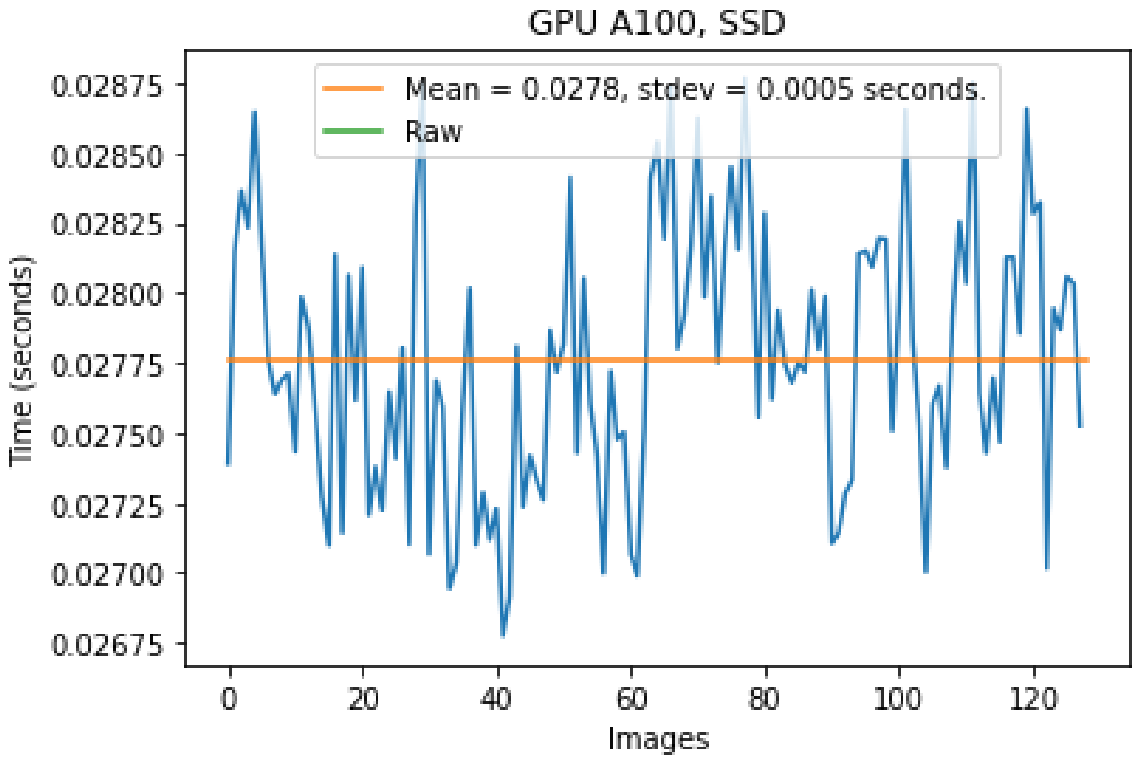


Рисунок 2.55 — Графік залежності часу тестування одного знімку для різних значень розміру міні-пакету на A100

Таким чином в результаті тестування виявлено практичну можливість отримувати прогнози щодо наявності чи відсутності ознак певних легеневих хвороб або аномалій на рівні ~ 36 зображень в секунду на даному програмному оточенні та апаратному забезпеченні (GPU NVIDIA A100, SDD).

На даному етапі роботи це значно (більше ніж в 2 рази) покращує результат на менш потужному оригінальному обладнанні і ще кращою мірою відповідає поставленій фінальній меті проєкту: обробляти від 10 до 100 зображень в секунду.

Ще раз варто зауважити, що ці результати також були отримані без застосування інших додаткових засобів та способів оптимізації обрахунку моделей, які планується застосувати на наступному етапі проєкту.

2.3.3 Створення інфраструктури (демо-версія) для тестування знімків від інших зовнішніх джерел

Для подальшої перевірки наявних моделей глибинного навчання для 14 легеневих захворювань та налаштування та оновлення цих моделей для COVID-2019 на базі серверу Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського" було розгорнуто технічний тестовий майданчик (демо-версія): http://77.47.193.185/covid/.

Рисунок 2.56 — Приклад виводу інформації на тестовому майданчику про ймовірність наявності аномалій, виявлених на рентгенограмах легень, та побудовою теплових мап цих аномалій

Наразі тестовий майданчик (демо-версія) використовується в загальному сенсі для вирішення поточної проблеми виявлення COVID-2019 на основі найбільш промаркованих наборів даних (CheXNet, CheXpert, $\cdots$ + обмежені набори даних COVID-2019, $\cdots$).

В більш конкретному сенсі, цей тестовий майданчик використовується для відпрацювання окремих компонентів платформи, які перераховані нижче в наступних підрозділах цього звіту.

2.4 Вибір/налаштування моделей глибинного навчання (DL) для підвищення якості зображень

В останні роки ми стали свідками значного прогресу в області надвисокої роздільної здатності зображень із використанням методів глибинного навчання. Надвисока роздільна здатність (Super resolution, SR) – це важливий клас методів обробки зображень для підвищення роздільної здатності зображень і відео в комп'ютерному баченні, що активно використовується для медичних даних також. Дана частина робота демонструє спроби дослідження останніх досягнень у цій області із використанням підходів глибинного навчання для подальшого використання в якості компонентів пропонованої платформи.

Результати цієї частини роботи опубліковані під час виконання етапу у статті та доповіді із відповідним згадуванням грантової підтримки НФДУ [74]:

"The work was partially supported by the National Research Foundation of Ukraine by the grant 2020.01/0490 "Artificial Intelligence Platform for Distant Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."

2.4.1 Вступ

У загальному випадку, існуючі дослідження методів SR можна згрупувати за трьома основним категоріями: контрольовані, неконтрольовані та домен-специфічні SR. Метод SR, який відноситься до процесу відновлення зображень до високої роздільної здатності (High resolution, HR) із зображень з низькою (Low Resolution, LR), є важливим класом методів для обробки зображень у комп'ютерному баченні. Він має широкий спектр реальних додатків, такі як медична візуалізація, спостереження та безпека та інші. Крім покращення якості сприйняття, цей метод також допомагає вирішити інші завдання комп'ютерного бачення. Загалом, ця проблема дуже складна і по своїй суті некоректна, оскільки завжди є кілька результуючих зображень HR, що відповідають одному зображенню LR.

З швидким розвитком методів глибинного навчання в останні роки SR-моделі, засновані на глибокому навчанні, активно досліджувалися і часто до-

сягали високого рівня продуктивності в різних тестах. Різноманітність глибоких методів навчання застосовувалися для вирішення завдань SR, починаючи з ранніх згорткових нейронних мереж (CNN) заснованих на методі, наприклад, SRCNN, та багатообіцяючих підходів SR з використанням генеративних змагальних мереж (GAN), наприклад, метод фото-реалістичної надвисокої роздільної здатності одного зображення за допомогою генеративної змагальної мережі (SRGAN) [14]. В цілому, сімейство алгоритмів SR, що використовують методи глибинного навчання, відрізняються один від одного в наступних основних аспектах: різні типи мережевої архітектури, різні типи втрат функцій, різні типи принципів навчання і стратегії і т. д.

Тому дослідження існуючих методів SR з використанням глибинного машинного навчання є надзвичайно актуальною та спрямоване на покращення результату виконання обробки зображень, а саме на використання більш сучасних та розповсюджених видів нейронних мереж.

У результаті виконаної роботи:

1. Було визначено програмні та апаратні вимоги для реалізації спроектованого методу. Найбільш доцільним для виконання поставлених задач було обрано використання локального ПК, характеристики якого були приведені.
2. У якості програмного забезпечення середовищ для виконання та запуску програмного коду були обрані Anaconda та JupyterNotebook, оскільки вони повністю відповідають вимогам.
3. Для тренування нейромереж було досліджено та використано набір даних DIV2K, оскільки він є одним з найкращих в плані кількості та якості тестових зображень.
4. Для покращення існуючих методів збільшення реалістичності сприйняття, де використовується нейромережа VGG19, було досліджено та реалізовано нейромережі MobileNetV2 та EfficientNetB7, оскільки вони мають одні з найкращих показників у порівнянні з аналогами, які було описано у даному розділі.

2.4.2 Метрики

Для об'єктивної оцінки в якості параметрів для порівняння реалізованого та базового методу, виступають якісні та кількісні значення, які ми отримали в процесі та результаті навчання та в результаті виконання роботи нейромереж. Кількісні параметри:

1. Втрата дискримінатора (Discriminator Loss) – найнижче значення коефіцієнта дискримінації для всіх етапів навчання.
2. Втрата сприйняття (Perceptual Loss) – найнижче значення коефіцієнта втрат сприйняття навчання для всіх етапів навчання.
3. Час на 1 крок навчання – середній час навчання 1 крок даної моделі в секундах.
4. Загальний час навчання – час навчання з конфігурацією, описаною нижче для спроектованої моделі.
5. Розмір моделі – скільки пам'яті займає конфігурація.
6. Кількість навчальних параметрів нейромережі – сума усіх параметрів шару для даної моделі.

Якісні параметри:

1. Опрацьоване зображення з надвисокою роздільною здатністю використовуючи піксельну втрату (Pixel Loss).
2. Опрацьоване зображення з надвисокою роздільною здатністю використовуючи втрату сприйняття.

2.4.3 Результати

Для проведених експериментів були досить добре налаштовані моделі EDSR та WDSR, оскільки SRGAN показує найкращі результати для цього підходу. Для отримання загальних необхідних результатів, кількість кроків для навчальних моделей була встановлена 30000 (300 епох по 100 кроків) для EDSR та WDSR та 2000 для SRGAN, оскільки нам потрібно отримати результат, який указує на ефективність моделі, швидкість його навчання та швидкість обробки зображень.

Так як для повного навчання системи для всіх моделей потрібно 30000 кроків, потрібно багато часу та ресурсів. Наше завдання – отримати кількісний результат виконання, і ми сподіваємось, що якість обробленого зображення буде знижено і не відображатиме кінцевий результат, який можна отримати після проходження необхідної кількості навчальних кроків. Порівняння основних параметрів у процесі навчання мереж подано у таблиці 2.1.

Таблиця 2.1 — Порівняння основних параметрів у процесі навчання мереж

Метрики	**VGG19**	**MobileNetV2**	**EfficientNetB7**
Розмір моделі	549 MB	14 MB	256 MB
Кількість параметрів	143 667 240	3 538 984	66 658 687
Час на 1 крок (с)	2.22	1.38	1.42
Загальний час (с)	4317	2815	2986
Perceptual loss	0.0844	0.0041	0.0007
Discriminator loss	0.1266	0.8976	1.2326

Як результат, ми можемо визначити, що порівняно з іншими моделями, базова модель VGG19 має найбільші: розмір, кількість параметрів, час навчання на один крок і для всіх кроків загалом, а також має найбільше значення втрат сприйняття. Але в порівнянні з EfficientNet та MobileNet, VGG19 має найнижчі втрати дискримінатора, що свідчить про те, що модель має більшу ефективність у співвідношенні зображень LR та HR, що, в свою чергу, впливає на якість базової обробки зображень при збільшенні.

EfficientNet має приблизно половину розміру моделі та кількості параметрів у порівнянні з VGG19. Він також має середній час навчання та найбільше значення втрати дискримінатора, але в порівнянні з рештою, значення втрати сприйняття в цій мережі є найменшим, що свідчить про хорошу якість остаточної обробки зображення нейронною мережею, а саме його реалізм.

MobileNet показав себе з боку найлегшої моделі - він має найменший розмір і кількість параметрів, а також час на тренування, який для одного кроку, та і для всього навчання у цілому. Значення дискримінатора та сприйняття є середніми порівняно з іншими мережами, але тим не менше значення втрат сприйняття, порівняно з VGG19, все ще значно нижчі, як для MobileNet, так

і для EfficientNet. Кількісні значення, основні показники точності втрат та час навчання для досліджуваних моделей представлені в таблиці 2.1.

Підсумовуючи значення, отримані в результаті навчання моделей, ми отримуємо відповідну якість обробки зображень. Таким чином, ми бачимо, що при обробці з втратою пікселів ми отримуємо приблизно однаковий візуальний результат, але оскільки MobileNet є найменш вимогливим до ресурсів, у цій категорії він має перевагу серед порівняних моделей, це демонструє час тренування для одного кроку, та усього тренування у цілому, а саме 1.38с та 2815с відповідно, коли у порівнюваних час відповідає 1.42с і 2986с для EfficientNet та 2.22с і 4317с для VGG19. Обробка з втратою сприйняття також показала відповідні результати, визначені під час тренування. Отже, у цій категорії EfficientNet має найкращі результати обробки, оскільки він має найнижчий середній коефіцієнт втрати сприйняття серед порівнянних моделей, а саме 0.0007 у порівнянні з 0.0041 у MobileNet та 0.0844 для VGG19.

Аналізуючи отримані результати у цілому, в даній роботі ми отримали метод, що має більш оптимізовані характеристики, але візуальна частина не дала необхідного результату, оскільки нейромережі, що використовувались не пройшли повний обсяг тренування, як було зазначено у базовому методі.

2.5 Вибір/налаштування моделей глибинного навчання (DL) для підвищення якості стискання зображень

З кожним роком попит на ефективний механізм стиснення зображень зростає з ростом темпів отримання даних, особливо медичних даних великого обсягу (рентгенографія, МРТ, КТ, тощо). Для їх обробки та збереження потрібні нові підходи до механізму стиснення даних. На додачу поліпшення загальних характеристик та якісних характеристик механізмів стиснення необхідно для ефективної передачі медичних зображень через Інтернет від та до периферійного рівня обслуговування пацієнтів.

Результати цієї частини роботи опубліковані під час виконання етапу у статті та доповіді із відповідним згадуванням грантової підтримки НФДУ [75]:

“The work was partially supported by the National Research Foundation of Ukraine by the grant 2020.01/0490 “Artificial Intelligence Platform for Distant

Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."

2.5.1 Моделі для підвищення якості стискання зображень

Поточна модель реалізує мережу із квантуванням змінних на рівні каналу для динамічного розподілу та зняття бітрейтів зі значних і незначних каналів. Його основна специфіка – це використання регулятора квантування змінної, що складається з таких компонентів: модуль важливості каналу, що динамічно дізнається про важливість каналів під час навчання, та модуль розщеплення-злиття, який виділяє канали різних бітрейтів.

Модель створена на основі наступних компонентів (рис. 2.57), які детально описані у відповідній публікації [75]:

1. Ко/Декодувальник.
2. Квантувальник.
3. Контролер квантизації.
4. Модуль важливості каналу.
5. Стискання і збудження блоків.
6. Реконструкція на основі помилок.
7. Модуль Поділу-Об'єднання.

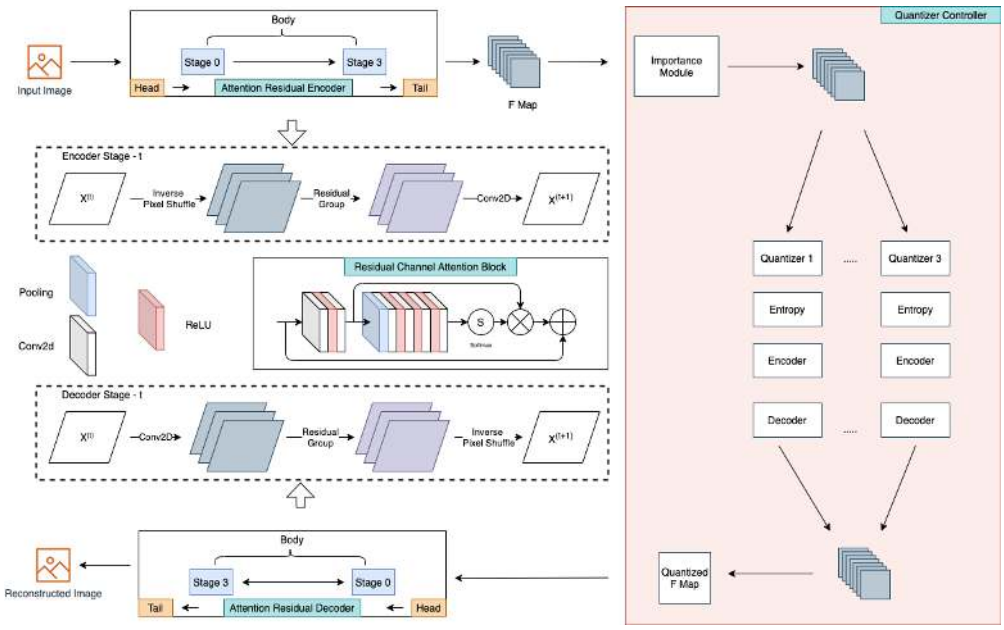


Рисунок 2.57 — Модель методу компресії

2.5.2 Тренування мережі

Для тренування ми розглянули 2 основних провайдери віртуальних машин:

1. Microsoft Azure.
2. Google Colab.

Вибір Microsoft Azure був зроблений через неможливість використання Google Colab для довготривалих процесів навчання, адже час життя сесії максимально може дорівнювати 12 годинам що значно менше ніж наш процес тренування. За основу було обрано віртуальну машину NC6 на Microsoft Azure із наступною конфігурацією:

1. CPU – 6 ядер.
2. RAM – 56 ГБ.
3. GPU – 1 Nvidia K80 з 12 ГБ пам'яті.
4. Hard Drive – 375 ГБ.
5. ОС – Azure OS що базується на Ubuntu.
6. Python 3, автоматично йде в комплекті із Azure OS.

Розмір пакета зменшено через обмеження пам'яті на GPU оскільки загальне використання пам'яті графічного процесора потребує більше 24 ГБ відеопам'яті. Навчання нейронної мережі може бути значно прискорене використання декількох графічних процесорів у комбінації із багато ядерним процесором, найкращим вибором буде процесори типу Ryzen. У майбутніх дослідженнях ми також протестуємо процесори TPU, що має значно прискорити процес навчання.

Виходячи з результатів експерименту (рис. 2.58), на основі метрик MS-SSIM, PNSR, (вище аналогів на 0.012 та 0.84 відповідно) можна зробити висновок про отриману кращу якість зображення. Процес тренування проведено більш ефективно зважаючи на те що використаний набір даних на 45% менший та кількістю тренувальних епох менша в 4 рази.

Якщо розглянути практичну сторону розглянутого питання то вкінці кінців треба обирати що потрібно більше - висока якість зображення та гірша компресія чи навпаки трохи гірше зображення при значно більшому коефіцієнті стискання.

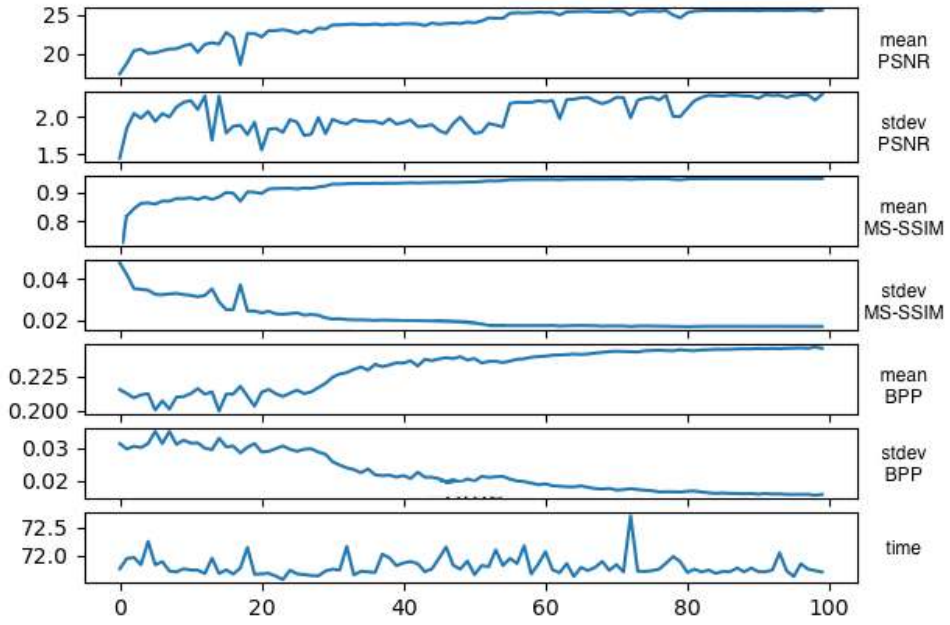


Рисунок 2.58 — Результати тренування

Проведений аналіз існуючих аналогів нейронних мереж для глибокої компресії зображення. Створено оновлену архітектуру методу компресії що спирається на глибокі нейронні мережі. Створений програмний код що написаний на мові програмування Python із використанням відкритої бібліотеки PyTorch та Keras що спеціалізовані на машинному навчанні. Продемонстровано та проаналізовано отримані результати відносно оригінальних даних з подібної архітектури методу компресії. На основі метрик MS-SSIM, PSNR та BPP відображено високу якість реконструйованого зображення та оптимізацію результатів відносно подібного дослідження.

2.6 Вибір/налаштування моделей глибинного навчання (DL) для покращеної сегментації зображень

2.6.1 Вступ

Аналіз медичних зображень за допомогою конвеєрів комп'ютерного зору є актуальним завданням у сфері охорони здоров'я. Раніше виявлення і сегмен-

тування різних медичних зображень тканин було виконано вручну, що може зайняти значний час. В останні роки з'явилися різні автоматизовані підходи розроблені для виконання цих завдань для різних органів людини (мозку, легені, печінку) та для різних цілей. Найпопулярніші методи використовувати нейронні мережі, які використовують різні нейронні мережі архітектури загального призначення (ResNet, VGG, AlexNet) та DNN, призначені спеціально для медичних зображень (U-Net, V-Net). Але поточний попит щодо ефективності сегментації зображень дуже високий в медичних цілях, особливо в контексті даного проєкту. Саме тому окремий напрям досліджень було присвячена вивченню можливостей використання існуючих методів сегментації органів людини. Зокрема, були розглянуті мережі U-Net та mU-Net сімейств.

Результати цієї частини роботи опубліковані під час виконання етапу у статтях та доповідях із відповідним згадуванням грантової підтримки НФДУ [76], [77]: *"The work was partially supported by the National Research Foundation of Ukraine by the grant 2020.01/0490 "Artificial Intelligence Platform for Distant Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."*

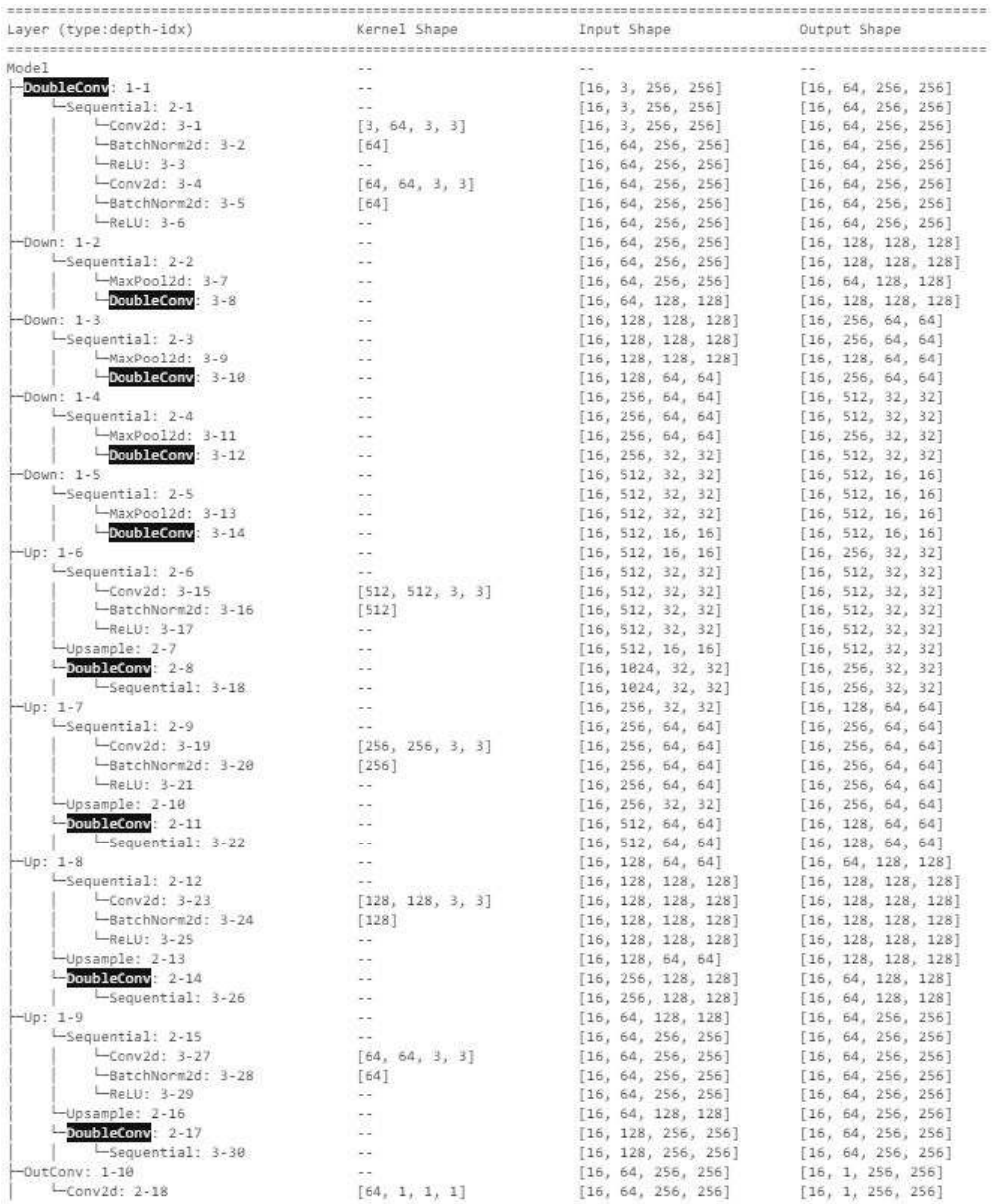
2.6.2 Моделі для покращеної сегментації зображень

Для цього дослідження досліджувалася архітектура U-Net сімейства мереж із метою покращення продуктивності на основі їх модифікацій U-Net. Архітектура mU-Net працює краще для невеликих об'єктів сегментації. Основна відмінність між оригінальною архітектурою U-Net - це змінений спосіб пропускання з'єднань, які містять додаткову пакетну нормалізацію та згорткові шари для посилення пропусків з'єднань. Остаточний шар - це вектор, розмір якого залежить від кількості класів сегментованих об'єктів. Специфікацію шару для чотириступеневої моделі 4 stage mU-Net (4-ступінчаста мережа U-Net) можна знайти нижче (рис. 2.59).

Тренування, валідація та тестування моделей

Для експериментів було використано 3 моделі:

1. 4-ступінчаста мережа U-Net.
2. 3-ступінчаста mU-Net.
3. 4-ступінчаста mU-Net



Layer (type:depth-idx)	Kernel Shape	Input Shape	Output Shape
Model	--	--	--
├─DoubleConv: 1-1	--	[16, 3, 256, 256]	[16, 64, 256, 256]
│ └─Sequential: 2-1	--	[16, 3, 256, 256]	[16, 64, 256, 256]
│ │ └─Conv2d: 3-1	[3, 64, 3, 3]	[16, 3, 256, 256]	[16, 64, 256, 256]
│ │ └─BatchNorm2d: 3-2	[64]	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─ReLU: 3-3	--	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─Conv2d: 3-4	[64, 64, 3, 3]	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─BatchNorm2d: 3-5	[64]	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─ReLU: 3-6	--	[16, 64, 256, 256]	[16, 64, 256, 256]
├─Down: 1-2	--	[16, 64, 256, 256]	[16, 128, 128, 128]
│ └─Sequential: 2-2	--	[16, 64, 256, 256]	[16, 128, 128, 128]
│ │ └─MaxPool2d: 3-7	--	[16, 64, 256, 256]	[16, 64, 128, 128]
│ │ └─DoubleConv: 3-8	--	[16, 64, 128, 128]	[16, 128, 128, 128]
├─Down: 1-3	--	[16, 128, 128, 128]	[16, 256, 64, 64]
│ └─Sequential: 2-3	--	[16, 128, 128, 128]	[16, 256, 64, 64]
│ │ └─MaxPool2d: 3-9	--	[16, 128, 128, 128]	[16, 128, 64, 64]
│ │ └─DoubleConv: 3-10	--	[16, 128, 64, 64]	[16, 256, 64, 64]
├─Down: 1-4	--	[16, 256, 64, 64]	[16, 512, 32, 32]
│ └─Sequential: 2-4	--	[16, 256, 64, 64]	[16, 512, 32, 32]
│ │ └─MaxPool2d: 3-11	--	[16, 256, 64, 64]	[16, 256, 32, 32]
│ │ └─DoubleConv: 3-12	--	[16, 256, 32, 32]	[16, 512, 32, 32]
├─Down: 1-5	--	[16, 512, 32, 32]	[16, 512, 16, 16]
│ └─Sequential: 2-5	--	[16, 512, 32, 32]	[16, 512, 16, 16]
│ │ └─MaxPool2d: 3-13	--	[16, 512, 32, 32]	[16, 512, 16, 16]
│ │ └─DoubleConv: 3-14	--	[16, 512, 16, 16]	[16, 512, 16, 16]
├─Up: 1-6	--	[16, 512, 16, 16]	[16, 256, 32, 32]
│ └─Sequential: 2-6	--	[16, 512, 32, 32]	[16, 512, 32, 32]
│ │ └─Conv2d: 3-15	[512, 512, 3, 3]	[16, 512, 32, 32]	[16, 512, 32, 32]
│ │ └─BatchNorm2d: 3-16	[512]	[16, 512, 32, 32]	[16, 512, 32, 32]
│ │ └─ReLU: 3-17	--	[16, 512, 32, 32]	[16, 512, 32, 32]
│ └─Upsample: 2-7	--	[16, 512, 16, 16]	[16, 512, 32, 32]
│ └─DoubleConv: 2-8	--	[16, 1024, 32, 32]	[16, 256, 32, 32]
│ │ └─Sequential: 3-18	--	[16, 1024, 32, 32]	[16, 256, 32, 32]
├─Up: 1-7	--	[16, 256, 32, 32]	[16, 128, 64, 64]
│ └─Sequential: 2-9	--	[16, 256, 64, 64]	[16, 256, 64, 64]
│ │ └─Conv2d: 3-19	[256, 256, 3, 3]	[16, 256, 64, 64]	[16, 256, 64, 64]
│ │ └─BatchNorm2d: 3-20	[256]	[16, 256, 64, 64]	[16, 256, 64, 64]
│ │ └─ReLU: 3-21	--	[16, 256, 64, 64]	[16, 256, 64, 64]
│ └─Upsample: 2-10	--	[16, 256, 32, 32]	[16, 256, 64, 64]
│ └─DoubleConv: 2-11	--	[16, 512, 64, 64]	[16, 128, 64, 64]
│ │ └─Sequential: 3-22	--	[16, 512, 64, 64]	[16, 128, 64, 64]
├─Up: 1-8	--	[16, 128, 64, 64]	[16, 64, 128, 128]
│ └─Sequential: 2-12	--	[16, 128, 128, 128]	[16, 128, 128, 128]
│ │ └─Conv2d: 3-23	[128, 128, 3, 3]	[16, 128, 128, 128]	[16, 128, 128, 128]
│ │ └─BatchNorm2d: 3-24	[128]	[16, 128, 128, 128]	[16, 128, 128, 128]
│ │ └─ReLU: 3-25	--	[16, 128, 128, 128]	[16, 128, 128, 128]
│ └─Upsample: 2-13	--	[16, 128, 64, 64]	[16, 128, 128, 128]
│ └─DoubleConv: 2-14	--	[16, 256, 128, 128]	[16, 64, 128, 128]
│ │ └─Sequential: 3-26	--	[16, 256, 128, 128]	[16, 64, 128, 128]
├─Up: 1-9	--	[16, 64, 128, 128]	[16, 64, 256, 256]
│ └─Sequential: 2-15	--	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─Conv2d: 3-27	[64, 64, 3, 3]	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─BatchNorm2d: 3-28	[64]	[16, 64, 256, 256]	[16, 64, 256, 256]
│ │ └─ReLU: 3-29	--	[16, 64, 256, 256]	[16, 64, 256, 256]
│ └─Upsample: 2-16	--	[16, 64, 128, 128]	[16, 64, 256, 256]
│ └─DoubleConv: 2-17	--	[16, 128, 256, 256]	[16, 64, 256, 256]
│ │ └─Sequential: 3-30	--	[16, 128, 256, 256]	[16, 64, 256, 256]
├─OutConv: 1-10	--	[16, 64, 256, 256]	[16, 1, 256, 256]
│ └─Conv2d: 2-18	[64, 1, 1, 1]	[16, 64, 256, 256]	[16, 1, 256, 256]

Рисунок 2.59 — Архітектура моделі 4 stage mU-Net

Використані метрики для оцінки продуктивності моделей:

1. Dice Score
2. Precision
3. Recall
4. Accuracy

Моделі навчалися протягом 60 епох, використовуючи оптимізатор Adam зі швидкістю навчання 10^{-4} у середовищі Kaggle IPython з 16 ГБ оперативної пам'яті, процесором Intel (R) Xeon (R) на 2.00 ГГц та графічним процесором NVIDIA Tesla P100-PCIE-16 з 16 ГБ відеопам'яті.

Після кожної епохи навчання проводилась оцінка основних метрик (рис. 2.60).

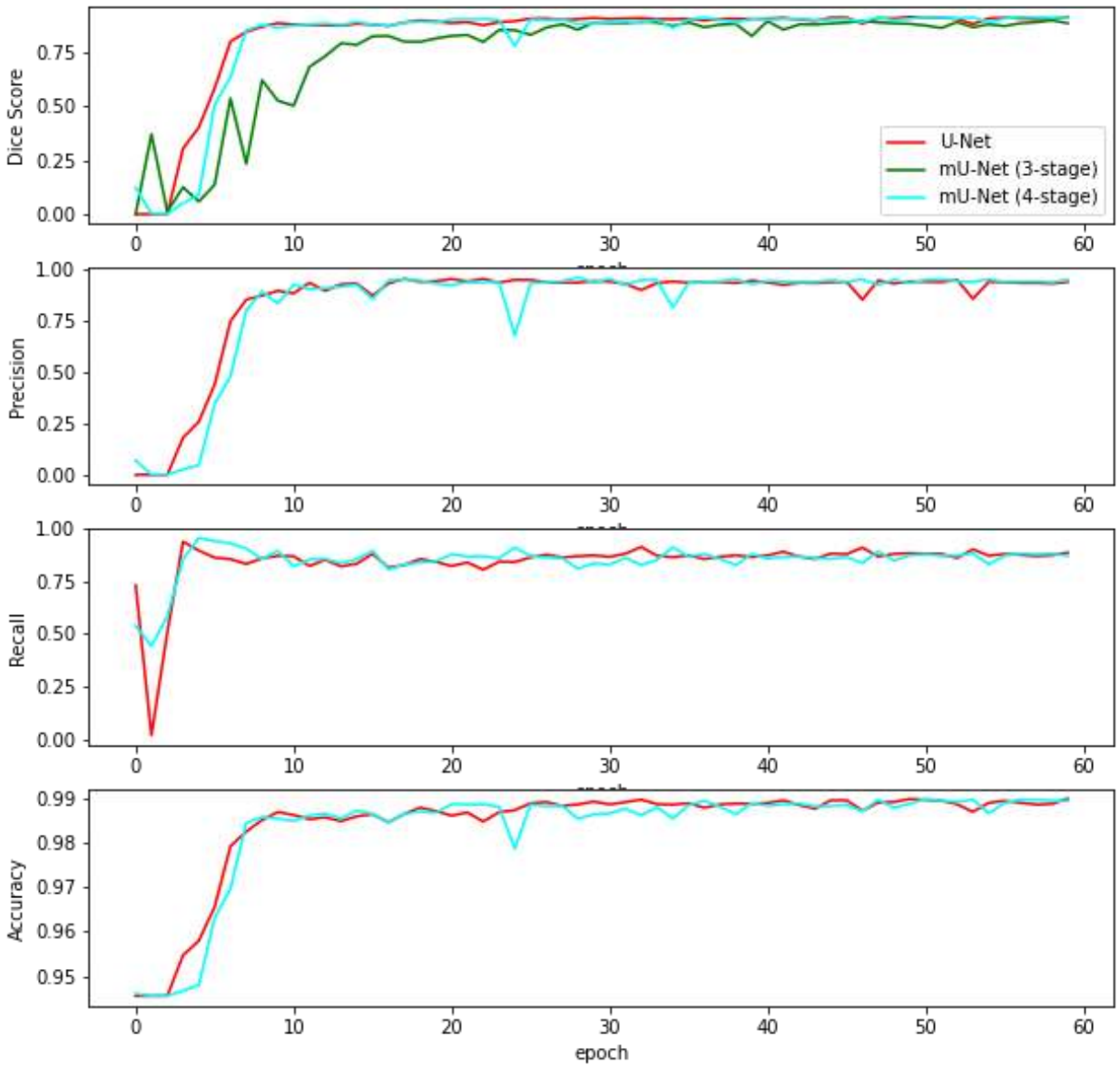


Рисунок 2.60 — Метрики для оцінки продуктивності моделей: 3-stage mU-Net, 4-stage mU-Net та U-Net

У таблиці 2.2 подано метрики для найкращих епох навчання розглянутих моделей.

Таблиця 2.2 — Метрики для найкращих епох навчання моделей

Модель	**Dice Score**	**Precision**	**Recall**	**Accuracy**
U-Net	0.9124	0.9395	0.8866	0.9101
4-stage mU-Net	0.9127	0.9494	0.8786	0.99
3-stage mU-Net	0.8965			

У цій роботі моделі U-Net та mU-Net були застосовані для перевірки способів покращення методів сегментації органів людини. Хоча було застосовано мінімальне збільшення даних та використано базову архітектуру мережі, метод показав можливості підвищення продуктивності близько 90порівняльний аналіз мереж U-Net та mU-Net показав, що архітектура mU-Net не забезпечує значного значення покращення до Dice-Score у цьому конкретному завданні. Серед архітектури, пов'язаних з U-Net, які широко використовуються для сегментація зображення, 3-ступеневу варіація mU-Net продемонструвала досить високу продуктивність, незважаючи на нижчий в 4 рази розмір моделі. В результаті достатньо високу точність можна досягнути навіть шляхом покращення елементарної архітектури мережі та за рахунок обмеженого часу тренування.

Отримані результати є надзвичайно важливими для практичного застосування сегментації зображень у медицині, наприклад, рентгенівських знімків грудної клітки, зображень меланоми, зображень магнітно-резонансної томографії (МРТ), комп'ютерної томографії та ін. Це особливо перспективно у контексті розгортання моделей сегментації на інфраструктурі периферійних обчислень Edge Computing з обмеженою кількістю обчислювальні здібності. Ось чому додаткові дослідження пошуку компромісу між продуктивністю моделі та розміром моделі слід виконати для застосування таких конкретних вимог.

Додатково (на іншому практичному прикладі) було перевірено запропоновано та досліджено кілька мінімальних удосконалень за допомогою різних функцій активації (наприклад, Swish та GELU) та змін базової нейронної мережі. Було виявлено найефективнішу комбінацію згаданих заходів, яка веде до менших DNN без шкоди для їх роботи. Ці запропоновані вдосконалення вважаються дуже перспективними для потенційного використання досліджуваних DNN в додатках Edge Computing на обчислювальних пристроях з обмеженими ресурсами [77].

2.7 Вибір/налаштування гібридних моделей глибинного навчання (DL)

Сучасні глибинні нейронні мережі (DNN) успішно застосовується для аналізу послідовностей часу, як електроенцефалографічні (ЕЕГ) сигнали мозку, зібрані за допомогою нейроінтерфейсу (brain-computer interface – BCI). Ця частина робота присвячена дослідженню аналіз даних ЕЕГ, зібраних BCI з метою класифікації на основі різних гібридних конфігурацій DNN, які можуть бути корисними для застосування для аналізу широкого кола медичних даних включно із даними про стан легенів (наведених у попередніх розділах). Кілька гібридних DNN розглядалися як комбінації згорткових нейронних мереж (CNN), повно зв'язаних мереж (FCN) та рекурентних нейронних мереж (RNN), у тому числі блоки довготривалої короткочасної пам'яті (LSTM) для класифікації деяких основні фізичні дії (рухи руками) на основі датасету GAL (який було детально описано на попередньому етапі). Отримані результати дозволяють зробити висновок, що ці гібридні DNN можна використовувати для класифікації фізичних дії надійно за допомогою невеликих і простих комбінацій, що важливо для перенесення гібридних моделей DNN для периферійних обчислювальних пристроїв (Edge Computing devices) з обмеженими обчислювальними ресурсами.

Результати цієї частини роботи опубліковані під час виконання етапу у статтях та доповідях із відповідним згадуванням грантової підтримки НФДУ [78]: *"The work was partially supported by the National Research Foundation of Ukraine by the grant 2020.01/0490 "Artificial Intelligence Platform for Distant Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."*

2.7.1 Варіанти гібридизації

Залежність між різними архітекторами DNN функцій та їх результативності у проблемі класифікації було досліджено для наступних типів CNN, описаних у наших попередніх роботах: повністю зв'язна нейронна мережа (FCN), подібна до Lenet CNN, подібна до Alexnet CNN, подібна до VGG-13 CNN, спрощена "ванільна" оригінальна DNN. На додаток до них на основі фреймворка

Tensorflow-Keras досліджували такі RNN: спрощена “ванільна” мережа LSTM на основі класичної архітектури LSTM (рис. 2.61) та гібридна мережа CNN-LSTM із компонентами CNN та додаванням блоку LSTM (рис. 2.62).

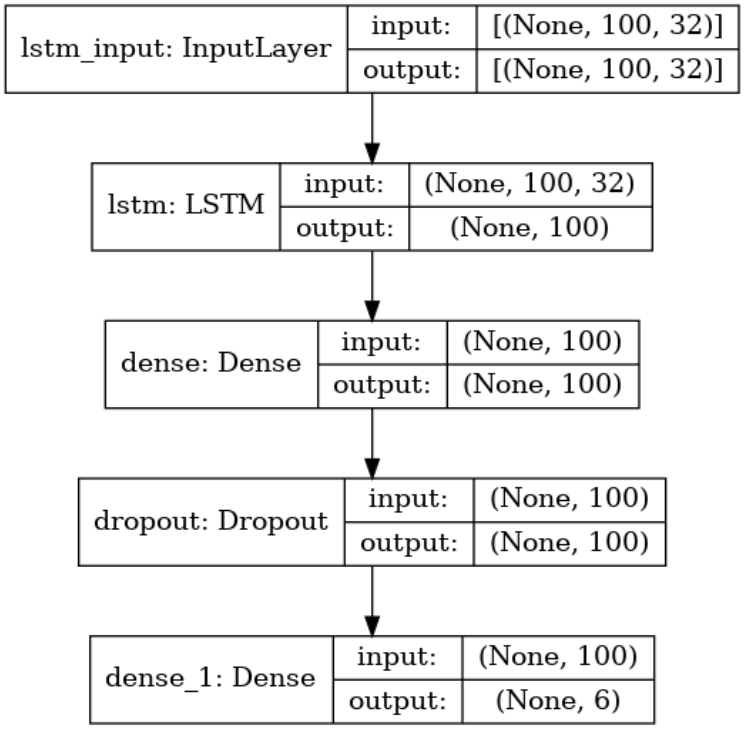


Рисунок 2.61 — Спрощена “ванільна” мережа LSTM на основі класичної архітектури LSTM та компонентів повністю зв’язаної нейронної мережі (FCN)

Як було показано в нашій роботі [78] (детальний опис дано у звіті попереднього етапу), основний стимулом до використання компонентів CNN було перевірити ефективність гібридизації операцій згортки із іншими компонентами, наприклад FCN. Робочі процеси з 32 каналів ЕЕГ-даних були об’єднані в цілком пов’язані щільні шари до класифікаційного шару. В даній частині було досліджено і перевірено інше уявлення про використання операцій згортки 1D на всіх 32 каналах ЕЕГ-даних для кожного часового кроку та 2D операцій згортки поперек всіх 32 каналів ЕЕГ-даних для вибірок з часової послідовності ЕЕГ для гібридних CNN-FCN та гібридних CNN-LSTM мереж.

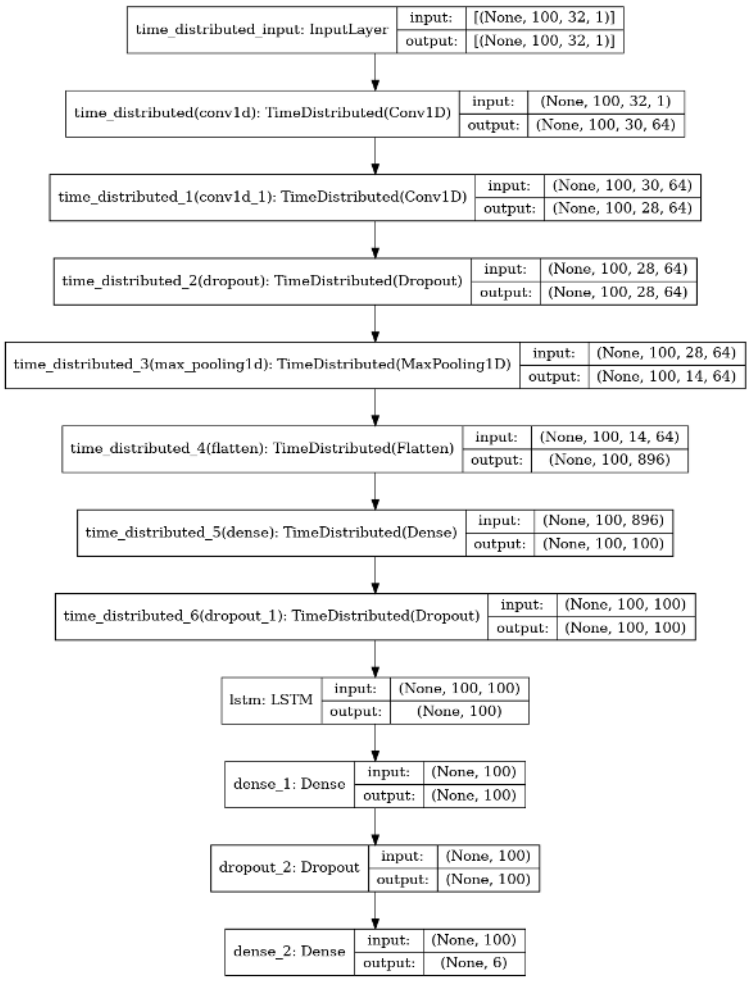


Рисунок 2.62 — Гібридна мережа CNN-LSTM на основі компонентів CNN та додаванням блоку LSTM класичної архітектури LSTM та компонентів повністю зв'язаної нейронної мережі (FCN)

2.7.2 Метрики

Для порівняння класифікаторів на основі гібридних конфігурацій використовували кілька стандартних метрик: точність, функцію втрат та їх статистичні параметри, як мінімальне значення (для втрати), максимальне значення (для точності); площа під кривою (area under curve -AUC) для робочих характеристик приймача (receiver operating characteristic– ROC), мікро-AUC та макро-AUC, а також їх статистичні параметри середнє та стандартне відхилення. AUC є порогово-інваріантною метрикою та вимірює якість прогнозів моделі незалежно від обраного порогу класифікації. В той час як точність залежить від

обраного порогу, AUC враховує усі можливі пороги. Ось чому іноді ця метрика може надати більш широкий погляд на ефективність класифікаторів.

Графіки залежності метрики AUC як функція від кількості зразків у вхідній послідовності даних (N) показані на рисунку 2.63.

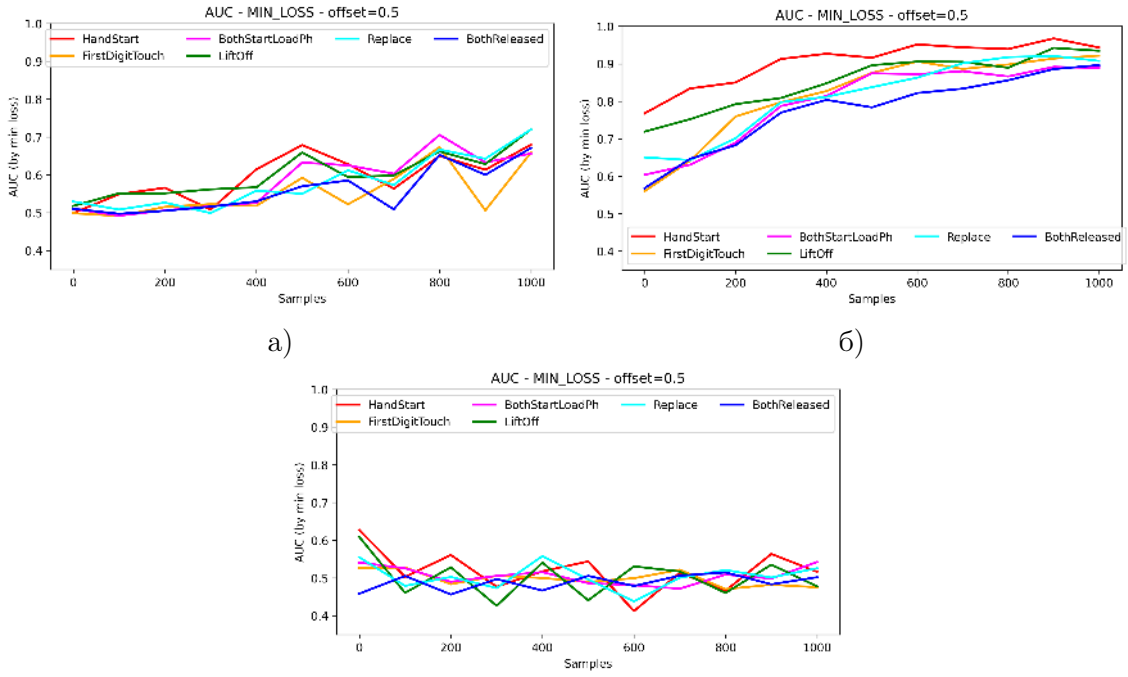


Рисунок 2.63 — Графіки залежності метрики AUC як функція від кількості зразків у вхідній послідовності даних (N) зі значенням зміщення: 0.5 для а) мережа FCN, б) гібридна мережа FCN-CNN, в) гібридна мережа CNN-LSTM

Отримані результати дозволяють зробити висновок, що гібридні DNN на основі компонентів FCN та CNN (рис. 2.63) можна використовувати для надійної класифікації фізичних дій (де AUC > 0.9 навіть після 1 навчальної епохи) навіть у невеликих та простих комбінаціях FCN/CNN. Це є дуже важливо для перенесення таких гібридних моделей DNN на периферійні обчислювальні пристрої (Edge Computing) з обмеженою кількістю обчислювальних ресурсів. Але внесок компонентів RNN (як і LSTM) не настільки ефективний у поодиноких і наївних комбінаціях з компонентами FCN/CNN. Ось чому додаткові дослідження щодо ефективності RNN для поточну проблему слід провести і вони будуть проведені в майбутньому. Ці висновки слід перевірити для більших наборів даних і з більшою увагою до внеску RNN-подібних компонентів. Ось чому

наступні кроки повинні бути присвячені забезпеченню більшого обсягу даних та підвищенню метрик продуктивності.

Загалом, такі розслідування можуть бути потенційно перспективними для прогнозування пов'язаних у часі подій у більш широкому сенсі, ніж у даному прикладі фізичних дії до їх фактичного початку, як в допоміжних пристроях для догляду за здоров'ям та людьми похилого віку (наприклад, екзоскелети) і можуть забезпечити негайний зворотний зв'язок як продовження цих дій, які були розпочаті лише мозковою діяльністю людей, але не можуть бути завершена через їх фізичні вади. В контексті легеневих захворювань гібридизація CNN та RNN-подібних компонентів дозволить в іншому ракурсі розглянути корелятивні залежності у сукупності окремих 2D знімків в рамках окремого КТ дослідження із 3D представленням даних. На даний момент цей напрямок не планувався в рамках даного проєкту, але відкриває нові можливості перспективних досліджень, які варто продовжити по завершенню даного проєкту.

2.8 Вибір/налаштування моделей глибинного навчання (DL) із застосуванням штучного розширення різноманітності даних

У даному розділі надано результати дослідження методу обробки медичних зображень, а саме:

1. Тренування стандартних і новітніх нейронних моделей поглибленого навчання із використанням методу крос-валідації (k-fold cross validation) і штучного збільшення кількості варіантів тренувальних зображень (data augmentation – DA).
2. Тестування отриманих тренованих моделей із залученням методу штучного збільшення кількості варіантів тестових зображень (post- training/testing time data augmentation – TTA).
3. Результати порівняльного аналізу отриманих даних на різних інфраструктурах, моделях і їх параметрах для дослідження впливу типу інфраструктури (TPU/GPU), типу і розміру моделей на час і точність прогнозування за стандартними метриками.

Результати цієї частини роботи опубліковані під час виконання етапу у статті та доповіді із відповідним згадуванням грантової підтримки НФДУ [79]: *"The work was partially supported by the National Research Foundation of Ukraine*

by the grant 2020.01/0490 "Artificial Intelligence Platform for Distant Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."

2.8.1 Тренування моделей та аналіз отриманих результатів

Детальний опис архітектури моделей було викладено у звіті про результати попереднього етапу проєкту. В даному підрозділі дається підсумковий аналіз отриманих і опублікованих результатів у розгорнутому форматі із підсумками відповідно до технічного завдання і поставлених метрик (якість – таблиця 2.3, та час тестування – таблиця 2.4) і статичні оцінки розкиду значень цих метрик.

Таблиця 2.3 — Величини показника якості класифікації та середнього квадратичного відхилення кожного типу архітектури моделі з урахуванням test-time augmentation

Модель	**TTA_AUC_mean**	**TTA_AUC_std**
EfficientNetB0	0.942	0.033
EfficientNetB6	0.931	0.027
DenseNet121	0.917	0.044
MobileNetV2	0.841	0.03
InceptionV3	0.926	0.04
ResNet101	0.928	0.054
ResNet101V2	0.917	0.03
InceptionResNetV2	0.944	0.043
VGG16	0.878	0.012
Xception	0.898	0.0352

Таблиця 2.4 — Показники часу тестування та середнього квадратичного відхилення для моделей, що були використані з урахуванням test-time augmentation

Модель	**TTA_time_mean**	**TTA_time_std**
EfficientNetB0	0.00102	0.000010
EfficientNetB6	0.00137	0.0000118
DenseNet121	0.00104	0.0000057
MobileNetV2	0.00102	0.000008
InceptionV3	0.00106	0.0000078
ResNet101	0.00105	0.0000081
ResNet101V2	0.00104	0.0000084
InceptionResNetV2	0.00140	0.000012
VGG16	0.00101	0.000011
Xception	0.00103	0.000006

На основі здобутих результатів було визначено, що:

Найменшими за розміром є мережі:

1. MobileNetV2
2. EfficientNetB0
3. DenseNet121

Найбільшу точність в межах похибки демонструють мережі:

1. EfficientNetB0
2. EfficientNetB6
3. InceptionResNetV

Найшвидшими без TTA є мережі:

1. MobileNetV2
2. EfficientNetB0
3. VGG16 та Xception мають майже ідентичні результати

Найшвидшими із TTA є мережі:

1. VGG16
2. MobileNetV2
3. EfficientNetB0

Отже, було сформульовано наступні рекомендації для практичного використання обстежених мереж та конфігурацій апаратного забезпечення.
Використання:

1. Для завдань із невисокими вимогами до часу прогнозу на стаціонарних пристроях із необмеженими ресурсами (хмарні ресурси на базі GPU-TPU вузлів) найбільш ефективно було б використовувати архітектури мережі EfficientNet, із найбільшою віддаючи точністю: перевагу моделі моделі EfficientNetB0.
2. Для завдань із високими вимогами до часу прогнозу на стаціонарних пристроях із обмеженими ресурсами (локальні ресурси на базі GPU карт) найбільш ефективно було б використовувати мережі із найбільшою точністю і найбільшою швидкістю: EfficientNetB0 єдина серед досліджених моделей, що здатна сумістити у собі всі три фактори.

На пристроях із обмеженими ресурсами (пристроях із портативними TPU-модулями, наприклад, Google Coral або Intel Movidius):

1. Для завдань із високими вимогами до часу прогнозу найбільш ефективно було б використовувати мережі найменшого розміру та найбільшою швидкістю: EfficientNetB0 та MobileNetV2, що мають найкращі результати у даних показниках.
2. Для завдань із невисокими вимогами до часу прогнозу найбільш ефективно було б використовувати мережі найменшого розміру та найбільшою точністю: EfficientNetB0 має найкращу комбінацію даних показників.

2.8.2 Аналіз результатів

Результати дослідження методу обробки медичних зображень людини на основі нейронної мережі для класифікації захворювань (на прикладі меланоми) на на реальних медичних зображеннях пацієнтів було підсумовано із визначенням основних параметрів їх роботи для прикладного застосування у сфері класифікації шкірних захворювань загалом і зокрема меланом.

Загалом для досягнення даної мети було:

1. Підготовлено датасет із медичними даними на основі відкритих даних 2018-2020 років із https://challenge2020.isic-archive.com/.

2. Сконфігуровано дослідницьку інфраструктуру на основі віддалених хмарних ресурсів і залученням графічних прискорювачів (graphic processing units– GPU) і прискорювачів тензорних обчислень (tensor processing units – TPU).
3. Виконано тренування MobileNetV2, на MobileNetV2, кількох стандартних InceptionResNetV2, (DenseNet121, ResNet101, ResNet101V2, VGG16, Xception) і новітніх (EfficientNetB0-B6) моделях поглибленого навчання із використанням методу крос-валідації (k-fold cross validation) і штучного збільшення кількості варіантів тренувальних зображень (data augmentation – DA).
4. Виконано тестування отриманих тренованих моделей із залученням методу штучного збільшення кількості варіантів тестових зображень (post-training/testing time data augmentation – TTA).
5. Виконано порівняльний аналіз отриманих даних на різних інфраструктурах, моделях і їх параметрах для дослідження впливу типу інфраструктури (TPU/GPU), типу і розміру моделей на час і точність прогнозування за стандартними метриками.
6. Сформульовано рекомендації щодо можливості практичного застосування отриманих результатів в якості методу обробки медичних зображень людини на основі нейронної мережі у вигляді сервісу із локальним або віддаленим доступом.

У кінцевому результаті було визначено, що модель EfficientNetB0 має найкращі перспективи, адже вона займає лідируючі позиції у більшості результатів вимірювань основних показників (розмір моделі, точність, час прогнозу) нейронних моделей і може стати основою для подальших впроваджень в даному контексті. Ці запропоновані вдосконалення вважаються дуже перспективними для потенційного використання досліджуваних DNN в додатках Edge Computing на обчислювальних пристроях з обмеженими ресурсами на периферійному рівні обслуговування пацієнтів.

2.9 Рішення на основі технології блокчейн для доказу автентичності медичного контенту глибинного навчання (deep learning content - DLC)

У цій частині роботи запропоновано рішення на основі технології блокчейн для доказу автентичності медичного контенту глибинного навчання (deep learning content – DLC), які можуть включати набори медичних даних, медичну метаінформацію (маркування професійних експертів), моделі, навчені на цих наборах медичних даних із пов'язаною медичною метаінформацією, медичні прогнози (результати застосування медичних моделей на нових медичних даних) тощо.

Результати цієї частини роботи опубліковані під час виконання етапу у статтях та доповідях із відповідним згадуванням грантової підтримки НФДУ [52], [80]:

"The work was partially supported by the National Research Foundation of Ukraine by the grant 2020.01/0490 "Artificial Intelligence Platform for Distant Computer-Aided Detection (CADe) and Computer-Aided Diagnosis (CADx) of Human Diseases"."

2.9.1 Структура системи

Розглянемо структуру пропонованої системи - вона буде базуватись на базових принципах блокчейну - прозорості та можливості відстеження, де захищена та довірена історія записів надається в децентралізованому форматі без посередників або довірених третіх сторін. В даному підрозділі ми опишемо деталі реалізації фреймворку, основою якого є блокчейн і який покликаний забезпечувати доказ справжності медичного контексту машинного навчання. Даний підхід також може застосовуватись і до інших контентів машинного навчання, що можуть містити кілька типів мультимедійної інформації крім медичних застосувань.

Пропонована система на основі блокчейну складається з таких основних основних компонентів:

1. Медичний датасет (D_i): містить важливу медичну інформацію таку як необроблені медичні дані(RD_i)(результати будь-яких медичних досліджень) і пов'язані з ними медичні метадані або медичні мітки (L_i) (рис. 2.64). Останні містять дані про саму RD_i: дані, про пристрій який надав RD_i , його настройки, час і дату проведення аналізів, разом з додатковою інформацією, яку вирішив додати автор датасету D_i. Кожен медичний датасет може бути асоційований з смарт-контрактом, що може бути створений як автором так і користувачем на основі того, хто займався розміткою даних, тобто $D_i = L(L_i, RD_i)$, де L це певна функція яка відображає процес помічання даних (рис. 2.65). Адреса в системі Ethereum та адреса смарт-контракту є невід'ємною частиною D_i.

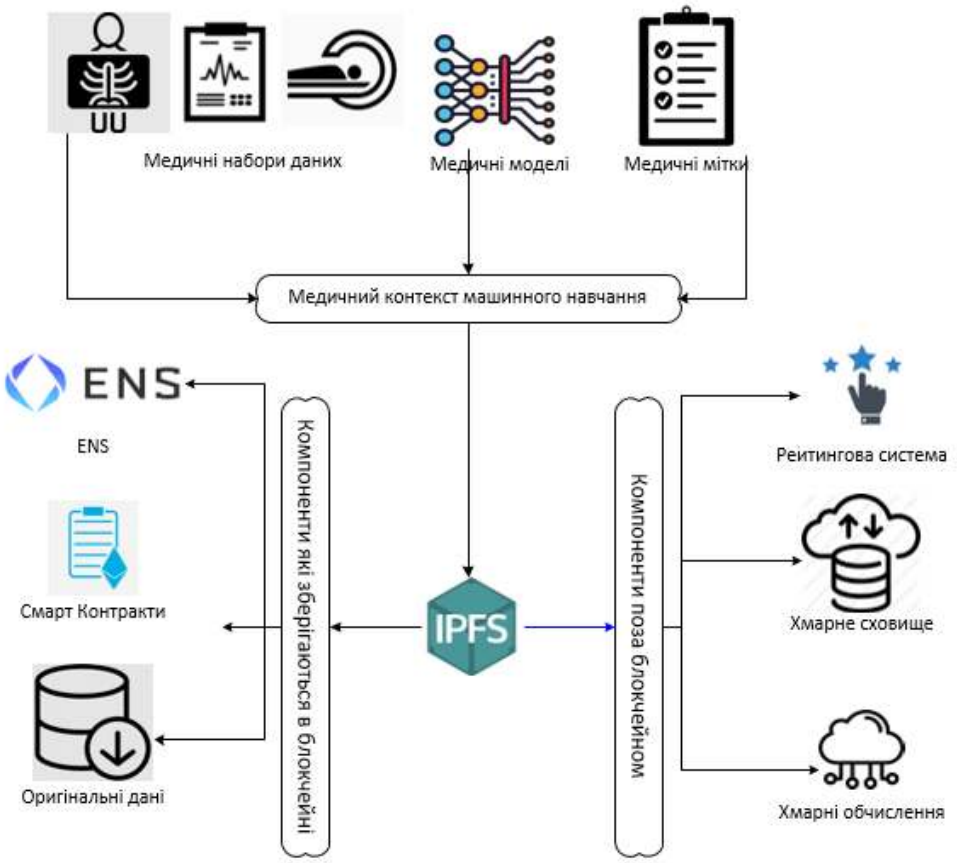


Рисунок 2.64 — Загальна структура пропонованої системи

2. Медична модель машинного навчання(M_i): Може містити як необроблену модель RM_i (тобто опис архітектури моделі машинного навча-

ння), пов'язані з моделлю метадані про датасет чи процес тренування(кількість ітерацій, частота навчання, тощо), і ваги моделі, яку натренували на конкретному датасеті D_i . Структура RM_i містить також інформацію про актуальну нейронну мережу – шари, активаційні функції, тощо. Модель M_i відображає тренований стан RM_i . Кожна модель M_i повинна бути асоційована з смарт-контрактом що може бути створений автором (в даному випадку, розробником моделі машинного навчання). Адреса автора та смарт-контракт моделі є невід'ємними частинами M_i.

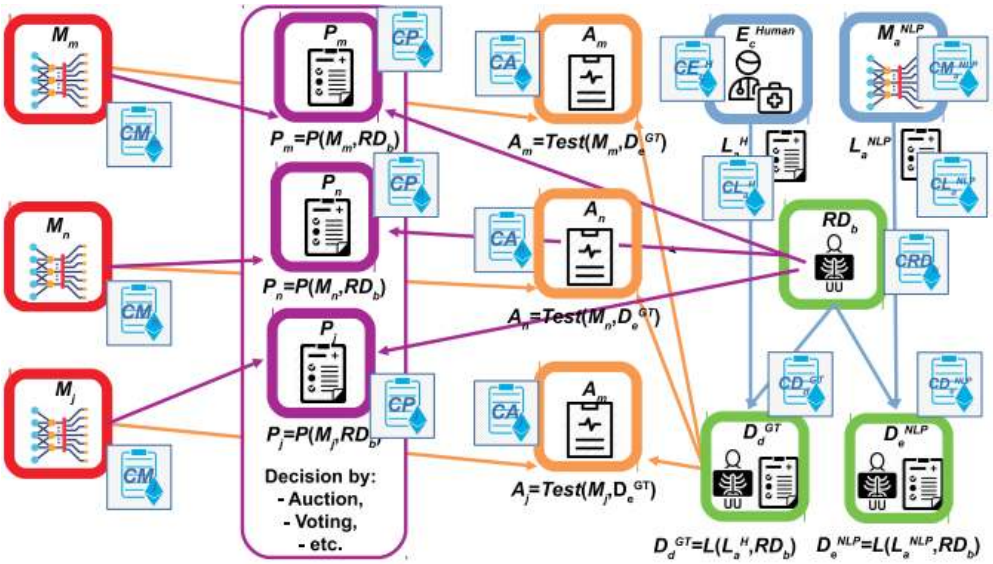


Рисунок 2.65 — Загальна структура пропонованої системи

3. Медичні передбачення(P_i): Може бути результатом виконання певної M_i застосованої до нових даних RD_i з передбаченими мітками(передбачення їх стану) P_i. Здається, що P_i призводить до створення "нових" даних D_i , оскільки P_i містить нові дані RD_i і новостворені мітки L і (мітки, які були згенеровані в результаті передбачення). Але, насправді P_i не може стати "новими" даними D_i , якщо тільки людина їх не перевірить L_i^H (рис. 2.65), або ці дані повністю заново розмітять кваліфіковані експерти. Кожна P і має бути асоційована з смарт-контрактом, що може бути створений автором моделі або користувачем моделі. Адреса автора в Ethereum так само як адреса смарт контракту є невід'ємними частинами P_i.

4. IPFS сховище: всі складові контенту машинного навчання зберігаються в децентралізованому, з адресацією по вмісту сховищі такому як IPFS, яке ми розглядали раніше. Воно забезпечить генерацію унікального хешу, який і буде адресою всього контенту або його частини.
5. Компоненти, які працюють в блокчейні: Після того, як ми згенерували IPFS хеш контенту машинного навчання, автором створюється смарт-контракт в Ethereum блокчейні. Контракт матиме атрибути і змінні, які зможуть відображати деталі контексту машинного навчання та інформацію про автора. Змінні також можна використовувати для зберігання статичної інформації, наприклад дані зв'язані з станом контракту або контекстом машинного навчання. На цьому етапі, будь-яке редагування контексту машинного навчання іншою особою окрім автора буде створювати свій окремий смарт-контракт з посиланням на оригінальний контекст машинного навчання. Звідси випливає, що всі редаговані контексти будь-якого оригінального контексту будуть "наслідника" оригінального контексту і будуть доступні до перегляду в списку в його смарт-контракті. Тому користувач, який хоче перевірити походження контексту машинного навчання до його оригіналу, з легкістю може зробити це використовуючи дані доступні в блокчейні – наприклад смарт-контракти контекстів з посиланням на оригінал а також списки контрактів, які використали поточний контекст за основу.
6. ENS: користувач також може скористатись Ethereum Name Service (ENS) — сервісом для встановлення відповідності адрес авторів чи користувачів в Ethereum до реальних даних (їх імена, організації, чи профілю) [78].
7. Компоненти, які працюють поза блокчейном: Користувач, який проводить відслідковування даних може також брати до уваги і компоненти, які працюють поза блокчейном, вони є невід'ємними частинами пропонованої системи, наприклад, хмарні обчислювальні ресурси, через великі розміри контенту машинного навчання та супутніх обчислень.

2.9.2 Варіанти еволюції контенту машинного навчання

На рисунку 2.66 зображено можливі варіанти еволюції нових версій моделі машинного навчання та набору даних.

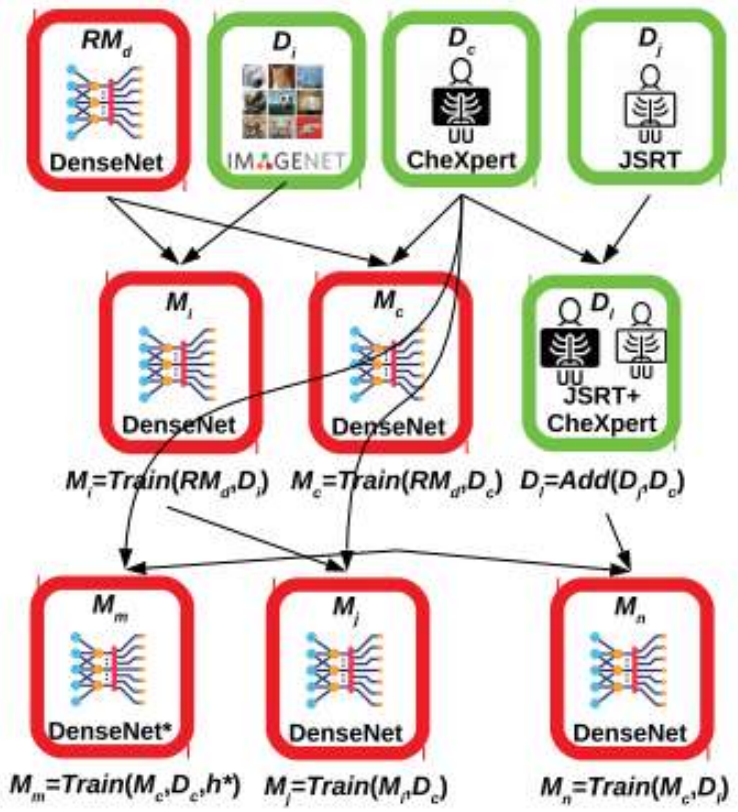


Рисунок 2.66 — Можливі варіанти еволюції нових версій набору даних та моделі

На даний час, без використання блокчейну, існують такі можливості типової еволюції набору даних та моделей:

1. Навчання необробленої моделі RM_d (наприклад, за допомогою алгоритмів сімейства DenseNet) на наборі медичних даних D_c (наприклад, CheXpert) може призвести до створення нової навченої моделі $M_c = \text{Train}(RM_d, D_c)$, і згодом додаткове тренування на тому ж самому наборі даних D_c , але з іншими параметрами (h^*), може призвести до створення нової моделі $M_m = \text{Train}(M_c, D_c, h^*)$, де Train – це функція, яка відображає процес повторного тренування.
2. Тренування необробленої моделі RM_d (наприклад, за допомогою алгоритмів сімейства DenseNet) на стандартному наборі даних D_i (на-

приклад Imagenet [80]) може призвести до нової тренованої моделі $M_i = \text{Train}(RM_d, D_i)$, потім, додаткове тренування на іншому наборі даних D_e (наприклад, CheXpert) може у результаті створити модель $M_i = \text{Train}(M_i, D_e)$.

3. Поєднання медичного набору даних D_e (наприклад CheXpert) з іншим набором медичних даних D_e (наприклад, JSRT [1]), створює новий комбінований набір даних $D_i = \text{Add}(D_j, D_e)$, де Add – це функція, яка показує процес поєднання (рис. 2.66), а потім додатково натренувати модель M_c на комбінованому наборі даних D_i що призведе до створення нової моделі $M_n = \text{Train}(M_c, D_i)$.

2.9.3 Відслідковування походження контенту машинного навчання

Зазвичай, розробники контенту машинного навчання, чітко розуміють еволюцію та походження фінальних версій наборів даних та моделей, які вони розробляють. Але для користувача, це не є настільки очевидним, особливо, беручи до уваги велику кількість нових контентів машинного навчання для рентгенівських знімків багатьох хвороб(наприклад раку, туберкульозу, та деяких інших аномалій легень), які публікуються науковими спільнотами(безліччю науковців та незалежних розробників). Це стало очевидним, під час поточної пандемії COVID-2019, коли з'являється багато нових наборів контенту машинного навчання, але після ретельної їх перевірки, багато з них виявились неякісними.

Основна мета пропонованого рішення – допомогти користувачам відслідковувати контент машинного навчання, який має багато версій до свого оригінального творця. Якщо походження контенту машинного навчання неможливо відслідкувати, тоді йому не можна довіряти, особливо в медичному контексті. Зазвичай користувачі мають доступ до певних контентів машинного навчання, наданих розробниками, з обмеженими метаданими про їх реальне походження. Але контент машинного навчання – це не тільки функція від попередніх станів, але також і їх процесів(наприклад, навчання з конкретними параметрами або поєднання наборів даних з певною попередньою обробкою), що веде до попередніх станів. Ось чому методи, які базуються на блокчейні(включаючи смарт-контракти, IPFS, ENS, та інші компоненти для встановлення автентично-

сті контексту машинного навчання) використовуються для надійного відстеження походження, особливо для недовірливої наукової спільноти.

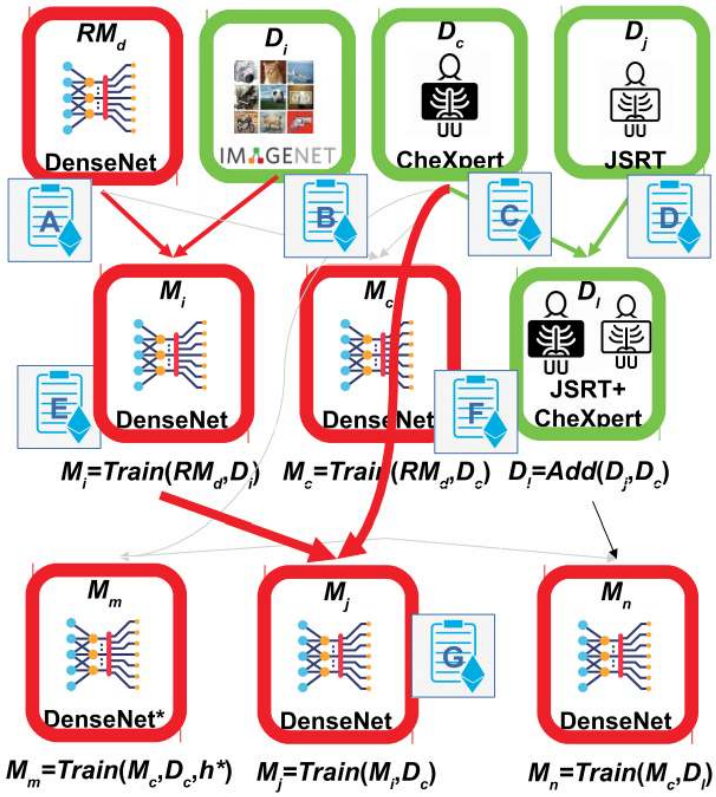


Рисунок 2.67 — Еволюція контенту машинного навчання моделі(червоні стрілки) та набору даних(зелені стрілки)

Рис. 2.67 показує, як користувач може відслідкувати еволюцію контенту машинного навчання внаслідок залучення смарт-контрактів. Наприклад, контракт A відповідає необробленій моделі RM_d тренованій на датасеті D_i (відповідає контракту B) з результуючою моделлю $M_i = \text{Train}(RM_d, D_i)$ (відповідає контракту E який є дочірнім контрактом його батьківських контрактів A та B). Подальший розвиток даного контенту машинного навчання включає перетренування моделі M_i (відповідає контракту C), що призводить до результуючої моделі $M_j = \text{Train}(M_i, D_c)$ (відповідає контракту G що є дочірнім контрактом E та C). Для контенту машинного навчання такого як набір даних, поєднання медичного набору даних D_c (відповідає контракту C) з іншим медичним набором даних D_j (відповідає контракту D) веде до створення набору даних $D_j = \text{Add}(D_j, D_c)$(відповідає контракту F який є дочірнім від C та D).

2.9.4 Контент машинного навчання з точки зору користувача

На рисунку 2.65 схематично показано вигляд на систему з точки зору користувача системи, де медичний контекст машинного навчання був отриманий від розробників шляхами, які відображені на рисунку 2.67 з залученням відповідних смарт-контрактів.

Варто зазначити, що еволюція необроблених медичних даних(наприклад, RD_b) до медичних наборів даних може проходити різними шляхами залежно від природи процесу анотації даних. У випадку анотації даних людьми L_a^H для необроблених медичних даних RD_b експертами E_e^{Human} можна отримати набір даних $D_e^{\text{GT}} = L(L_a^H, RD_b)$. Але в багатьох випадках мітки ставляться комп'ютерними системами L_a^{NLP} для необроблених медичних даних RD_b виконується за допомогою NLP моделей машинного навчання і отримуємо набір даних $D_e^{\text{NLP}} = L(L_a^{\text{NLP}}, RD_b)$. При відстеженні походження цих двох шляхів, обидва з них можуть бути відстежені до їх контрактів, наприклад, контракт CD_d^{GT} це дочірній контракт CRD_b та CL_a^H, де останній – дочірній контракт CE_a^H. Користувач може отримати контент машинного навчання (моделі M_m, M_j, M_n) з надійним походженням для побудови наступних прогнозів: $P_m = P(M_m, RD_b)$, $Pn = P(M_n, RD_b)$, $P_j = P(M_j, RD_b)$. На базі даних передбачень, користувач може отримати рішення, які будуть допомагати в визначенні точного діагнозу по рентгенівським знімкам пацієнтів. Також, метрики, які можуть збиратись під час отримання результатів від моделей, можуть бути використані для подальшого покращення моделей. Фронт-енд децентралізованого застосунку може бути розроблений для автоматизації процесу автентифікації для користувачів, або може бути інтегрований в інші веб-застосунки, або сервіси, для того, щоб визначати автентичність певних контентів машинного навчання. Як показано на рисунку 2.67, кожен контент машинного навчання зв'язаний з дочірніми, і кожен батьківський контент зв'язаний з дочірнім в ієрархічному порядку. Саме тому і можна відслідкувати смарт-контракт до його першої версії. Всі ці дані про походження, будуть публічно доступні в блокчейні Ethereum.

В даному підрозділі розглянуто технології які будуть потрібні для розробки системи, визначили їх переваги та недоліки, особливості роботи з ними, а також вибрали найбільш підходящі варіанти для створення системи походження медичного контенту машинного навчання. Спроектувано схему роботи систе-

ми, яка базується на використання технології блокчейну, визначення походження медичного контенту машинного навчання. Власник оригінального контенту може бути надійно визначений, завдяки використанню децентралізованого підходу. Пропонована система також буде використовувати децентралізоване сховище IPFS, систему визначення власника адрес ENS, а також хмарні системи які надають обчислювальні потужності.

Дане рішення пропонується під використання медичних даних, але є досить гнучким і з невеликими модифікаціями зможе бути використане для аналогічних цілей як в інших галузях, так і для інших типів даних. Описана система може допомогти в боротьбі з некоректними та ненадійними контентами машинного навчання які можуть бути розповсюджені зловмисниками. Користувач зможе відслідкувати контент до його автора, яким може бути довірена і відповідальна особа чи організація. Якщо ж користувач не може відслідкувати контент, то це означає, що його не можна використовувати.

Так само, таке рішення дозволить забезпечити систему для отримання дозволів на редагування контенту машинного навчання, як повністю, так і його частини, а також надати можливість автору оригінального контексту перевірити зміни, які вніс інший автор, перед публікацією.

Запропоноване рішення дозволяє виконати перевірку загальнодоступного, надійного та достовірного походження даних, з відстеженням та відстеженням історії опублікованого медичного журналу DLC. Наше рішення зосереджено на DLC медичного застосування, що може бути використано в різних медичних застосуваннях і буде адаптовано для федеративного навчання в рамках поточних викликів щодо глобального спільного використання DLC.

Оскільки запропоноване в цій роботі рішення є досить загальним, то воно може бути застосованим до будь-якої іншої форми DLC, наприклад наборів даних, метаінформації, моделей, навчених на цих наборам даних з відповідною метаінформацією, тощо.

2.10 Висновки до РОЗДІЛ 2

У даному розділі описані результати роботи методу обробки медичних зображень людини на основі нейронної мережі на основі результатів дослідження процесу класифікації об'єктів з медичних датасетів. Зроблено вибір/налаштува-

ння стандартних моделей глибинного навчання (DL) на конфігурації віртуального прототипу обладнання та програмного забезпечення на основі платформи медичного застосування Nvidia Clara та моделей для периферійних обчислень (включно із Google Coral, Inel Movidius, тощо). Досліджено, розроблено та протестовано: міні-модель DTL-19 з перенавчанням щільних шарів (WU1), міді-модель DTL-19 з перенавчанням деяких згорткових CNN-шарів + щільних шарів (WU1), максі-модель DTL-19 з перенавчанням усіх шарів (WU2) та інших моделей з елементами гібридизації компонентів нейронних мереж інших архітектур.

Для цих моделей виконано оцінку пропускної здатності моделей глибинного навчання на робочій станції WU1 (Titan 1080/HDD) та робочій станції WU2 (A100/SSD), які були встановлені в рамках попереднього етапу. В результаті тестування виявлено практичну можливість отримувати прогнози щодо наявності ячи відсутності ознак певних легеневих хвороб або аномалій на рівні 14-16 зображень в секунду на даному програмному оточенні та апаратному забезпеченні (GPU Titan 1080, HDD). На даному етапі роботи це відповідає поставленій фінальній меті проєкту обробляти від 10 до 100 зображень в секунду. Додатковий потенціал покращення було реалізовано за рахунок використання нового обладнання (GPU NVIDIA A100, SDD), де виявлено практичну можливість отримувати прогнози щодо наявності чи відсутності ознак певних легеневих хвороб або аномалій на рівні $\sim$ 36 зображень в секунду. На даному етапі роботи це значно (більше ніж в 2 рази) покращує результат на менш потужному оригінальному обладнанні і ще кращою мірою відповідає поставленій фінальній меті проєкту обробляти від 10 до 100 зображень в секунду. Варто зауважити, що ці результати також були отримані без застосування інших додаткових засобів та способів оптимізації обрахунку моделей, які планується застосувати на наступному етапі проєкту.

Створено інфраструктури (демо-версія) для тестування знімків від інших зовнішніх джерел. Також цей тестовий майданчик використовується для відпрацювання окремих компонентів платформи: вибір/налаштування моделей глибинного навчання (DL) для підвищення якості зображень в контексті методів Image Super Resolution, підвищення якості стискання та розархівування зображень, кращих методів сегментації зображень на основі моделей U-Net сімейства, застосування способів штучного розширення різноманітності даних із покраще-

нням метрик пропускної здатності, розробка рішення на основі технології блокчейн для доказу автентичності медичного контенту глибинного навчання.

У кінцевому підсумку було визначено найкращі перспективи використання певних DL моделей для певних практичних застосувань на основі результатів вимірювань основних показників (розмір моделі, точність, час прогнозу) цих DL моделей.

РОЗДІЛ 3. ПЛАТФОРМА ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ДИСТАНЦІЙНОГО АВТОМАТИЗОВАНОГО ВИЯВЛЕННЯ ТА ДІАГНОСТИКИ ЗАХВОРЮВАНЬ ЛЮДИНИ

У даному розділі наводяться результати виконання роботи відповідно до наступних завдань, які були заплановані для виконання під час проєкту:

1. Розробити науково обґрунтовані DL-моделі (на базі архітектур DenseNet, ResNet, InceptionResNet, Inception, Xception, EfficientNet, VGG та їхніх модифікованих версій та ансамблів), які були та адаптовані для використання в якості ШІ-асистентів лікаря на базі розробленої ШІ-платформи.
2. Протестувати обладнання та програмне забезпечення для компонентів ШІ-платформи та ШІ-платформи в цілому для застосування у практичних умовах. Підготувати конфігурацію придбаного обладнання (модулі для периферійних обчислень (включно із Google Coral, Intel Movidius, тощо) та розробленого програмного забезпечення для використання в польових умовах.
3. Спроектувати ШІ-платформу (інтегрована апаратна та програмна інфраструктура) на базі придбаного, налаштованого та протестованого апаратного забезпечення із розробленим програмним забезпеченням медичного призначення (включно із встановленими та адаптованими для використання науково обґрунтованими DL-моделями).
4. Розробити прототип ШІ-платформи, тобто пілотний варіант конфігурації ШІ-платформи (серверна сторона та веб-інтерфейс для медичного працівника).
5. Протестувати ШІ-платформу, модуль для периферійних обчислень і врахувати зворотній зв'язок від медичних працівників, зробити висновки щодо техніко-економічного обґрунтування подальшої імплементації та масштабування ШІ-платформи для використання в CADe та CADx в практичних умовах.
6. Підготовувати заключний звіт.

Для виконання цих завдань та інтенсифікації досліджень було сформовано кілька паралельних потоків для виконання окремих складових частин роботи, які не є залежними і не пов'язані причинно-наслідковими зв'язками, а саме

розробка прототипу ШІ-платформи, тестування окремих компонентів (ШІ-сервісів та комбінацій ШІ-сервіс + ШІ-модель) ШІ-платформи, розробка модулю для периферійних обчислень, ідентифікація аномальності (ознак хвороб), візуалізація теплових карт, оцінка пропускної здатності окремих компонентів (ШІ-сервісів та комбінацій ШІ-сервіс + ШІ-модель) ШІ-платформи на наявному апаратному забезпеченні.

Ці паралельні робочі потоки виконувалися на створеному на підготовчому етапі робочому вузлі 1 (work unit 1 — WU1), вузлі 2 (work unit 2 - WU2) на встановленій робочій станції, та модулі для периферійних обчислень, опис яких було дано у звіті (РОЗДІЛ 1).

3.1 Загальний опис платформи

Застосування сучасних "розумних" інструментів на основі методів ШІ може значно допомогти в процесі аналізу надзвичайно великого обсягу медичних даних, включаючи одновимірні (1D) акустичні, двовимірні (2D) рентгенографічні, тривимірні (3D) КТ/МРТ томографії, багатоканальні електроенцефалографічні (ЕЕГ), як додатковий інструмент для аналізу та виявлення ознак захворювань експертами, що наразі є тривалим і складним процесом.

Сучасна медична спільнота чудово обізнана із потенціалом методів штучного інтелекту (ШІ). Федеральне управління США з санітарного нагляду за якістю харчових продуктів та медикаментів (Food and Drug Administration - FDA) до цього моменту схвалила до практичного медичного використання > 40 алгоритмів та пристроїв ШІ. В майбутньому лікарям потрібно не тільки бути готовими до використання ШІ технологій, але вони також мусять бути готовими формувати майбутні шляхи розвитку цих технологій.

Основна мета запропонованої демо-версії ШІ-платформи полягає в демонстрації і тестуванні практичної спроможності використання ШІ-методів для дистанційного автоматизованого виявлення (Computer-Aided Detection – CADe) і дистанційної автоматизованої діагностики (Computer-Aided Diagnostics – CADx) хвороб людини.

Для розробки і функціонування ШІ-платформи на основі отриманих протягом проєкту наукових результатів було створено колекцію медичних даних (датасетів), бібліотеку моделей, описів конфігураційних параметрів та рекомен-

дації щодо їх застосування. Для імплементації усього створеного набору даних, моделей та параметрів у практичних умовах і перевірки спроможності їх використання в робочих умовах було створено платформу штучного інтелекту (ШІ-платформу), яка складається із програмної та апаратної частин.

До апаратної частини[1] відносяться:

1. Спеціалізовані прискорювачі на основі графічного процесора (GPU).
2. Сервери на базі яких спеціалізовані прискорювачі використовуються.
3. Спеціалізовані прискорювачі на основі тензорних процесорів (TPU) для периферійних обчислень (Edge Computing – EC).

До програмної частини ШІ-платформи відносяться програмне забезпечення:

1. Для тренування, валідації та тестування ШІ-моделей.
2. Для використання ШІ-моделей в якості допоміжних інструментів лікаря (ШІ-асистенти лікаря).
3. Для побудови веб-інфраструктури та дистанційного доступу до ШІ-асистентів лікаря у вигляді ШІ-веб-сервісів для медичного застосування.
4. Для використання лікарем ШІ-асистентів у польових умовах без серверних спеціалізованих прискорювачів та доступу до глобальної мережі у вигляді дешевих портативних пристроїв для периферійних обчислень (Edge Computing – EC).

Особливістю побудови програмної частини ШІ-платформи є необхідність використання у практичних умовах великої кількості ШІ-моделей з боку потенційно великої кількості користувачів (лікарів) на обмеженому колі спеціалізованих прискорювачів на основі графічного процесора (GPU). Критичною властивістю і вадою найбільш поширених варіантів сучасних GPU є монопольне виділення ресурсів GPU для одного користувача однієї ШІ-моделі із необхідністю проходження всього ланцюга робочих процесів із великою кількістю накладних витрат на підготовку оточення (від відкриття робочого сеансу, створення оточення, завантаження програмного забезпечення, виконання необхідних обчислень, звільнення ресурсів, закриття сеансу) навіть для обробки одного фрагменту медичних даних (наприклад одного рентгенівського знімка для одного із ШІ-сервісів). На даний момент нами досліджено і запропоновано більше 30 ШІ-

[1] Придбане обладнання для виконання даного проєкту.

моделей, які можуть бути використані в якості допоміжних інструментів лікаря (ШІ-асистенти лікаря) для 5 різних типів практичних застосувань (ШІ-послуг):

1. Класифікація аномалій в легенях людини по 14 типах легеневих захворювань.
2. Детекція аномалій в легенях (визначення розташування та класу аномалій).
3. Класифікація COVID-аномалій в легенях людини.
4. Класифікація аномалій шкіри (меланома).
5. Класифікація аномалій шкіри (хвороба Лайма).

В умовах колективного використання, велика кількість лікарів буде формувати велику кількість різних запитів для обробки медичних даних своїх пацієнтів різними типами ШІ-послуг та різними ШІ-моделями в основі цих ШІ-послуг. Усі найбільш поширені варіанти сучасних GPU дозволяють використовувати ШІ-послуги (тобто ШІ-моделі в їх основі) лише в послідовному режимі із великими накладними витратами на підготовку оточення із подовженням часу обробки, що унеможливлює їх масове використання у звичайних лікарнях.

Донедавна єдиним рішенням цієї проблеми могло би бути лише використання великої кількості обчислювальних ресурсів (збільшення кількості GPU) із значними інвестиціями у апаратне та програмне забезпечення для кожної окремої лікарні. Із розповсюдженням парадигми хмарних обчислень виникла можливість використовувати у вигляді веб-сервісів лише ті обчислювальні ресурси, які потрібні в даний момент без накладних витрат на простої апаратного забезпечення у період їх невикористання. Але навіть за таких умов для швидкої обробки різноманітних запитів до різноманітних ШІ-послуг виникає необхідність виділення великої кількості обчислювальних ресурсів, тобто збільшення кількості GPU хмарного розташування (віртуальних машин із GPU). Ідеальним варіантом (для максимально швидкої обробки запитів) було б використання по одній віртуальній машині із GPU на одну комбінацію ШІ-послуги+ШІ-модель. Але цей варіант не є практичним з огляду на його високу вартість при умові масового практичного використання в медичних закладах України.

У даному проєкті запропоновано, досліджено та перевірено практичну спроможність кількох варіантів вирішення цієї проблеми, тобто пошуку ефективної (в рамках компромісу між швидкістю та вартістю) обробки великої

кількості різних запитів з боку лікарів для обробки медичних даних своїх пацієнтів різними типами комбінацій ШІ-послуга+ШІ-модель в практичних умовах.

Для імплементації варіантів вирішення означеної вище проблеми було використано кілька новітніх технологій, а саме:

1. Технологію конфігурування та використання великої кількості екземплярів спеціалізованого прискорювача GPU-типу.
2. Методи побудови та керування чергами завдань.
3. Способи створення та управління контейнерами програмного забезпечення.

Для цього було створено демо-версію ШІ-платформи на основі наступних компонентів:

1. Сервер для збору та збереження медичних даних.
2. Сервер із спеціалізованими прискорювачами GPU-типу для розробки, тренування, валідації та тестування ШІ-моделей.
3. Сервер для обробки медичних даних із спеціалізованими прискорювачами GPU-типу.
4. Веб-сервер для дистанційного доступу до ШІ-асистентів лікаря у вигляді ШІ-веб-сервісів для медичного застосування.
5. Модуль периферійних обчислень для застосування ШІ-асистентів лікаря у вигляді ШІ-веб-сервісів для медичного застосування у польових умовах.

3.2 Інструкції для користування платформою

Розроблена ШІ-платформа дозволить кінцевим споживачам (працівникам медичних закладів) створювати, розгортати та керувати спеціалізованими ШІ-сервісами на основі ШІ-моделей у гібридних обчислювальних середовищах (на основі хмарних та периферійних обчислень) для доступу до медичних даних надвеликого обсягу, їх обробки, аналізу та звітності в робочому процесі виявлення відхилень та ознак захворювань.

З точки зору кінцевого користувача (медичних працівників) робочий потік операцій (workflow) для обробки великої кількості різних запитів з боку лікарів для обробки медичних даних своїх пацієнтів різними типами комбінацій ШІ-

послуга+ШІ-модель в практичних умовах включає наступну послідовність дій за певними правилами:

1. Відкрити головну сторінку ШІ-платформи (рис. 3.1) у вікні веб-браузера за наступним посиланням: https://med.comsys.kpi.ua
2. Ознайомитися з:
    (а) Загальним описом ШІ-платформи (рис. 3.2): https://med.comsys.kpi.ua/about
    (б) Правилами користування та отримання доступу до повного функціоналу ШІ-платформи (рис. 3.3): https://med.comsys.kpi.ua/rules

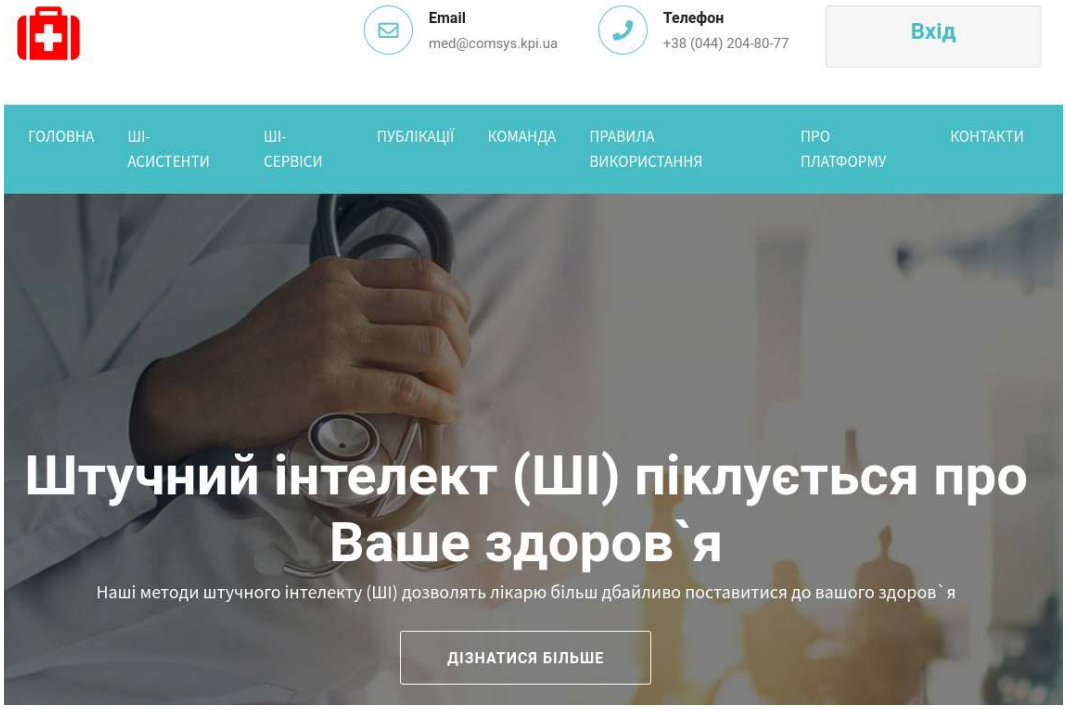


Рисунок 3.1 — Головна сторінка ШІ-платформи

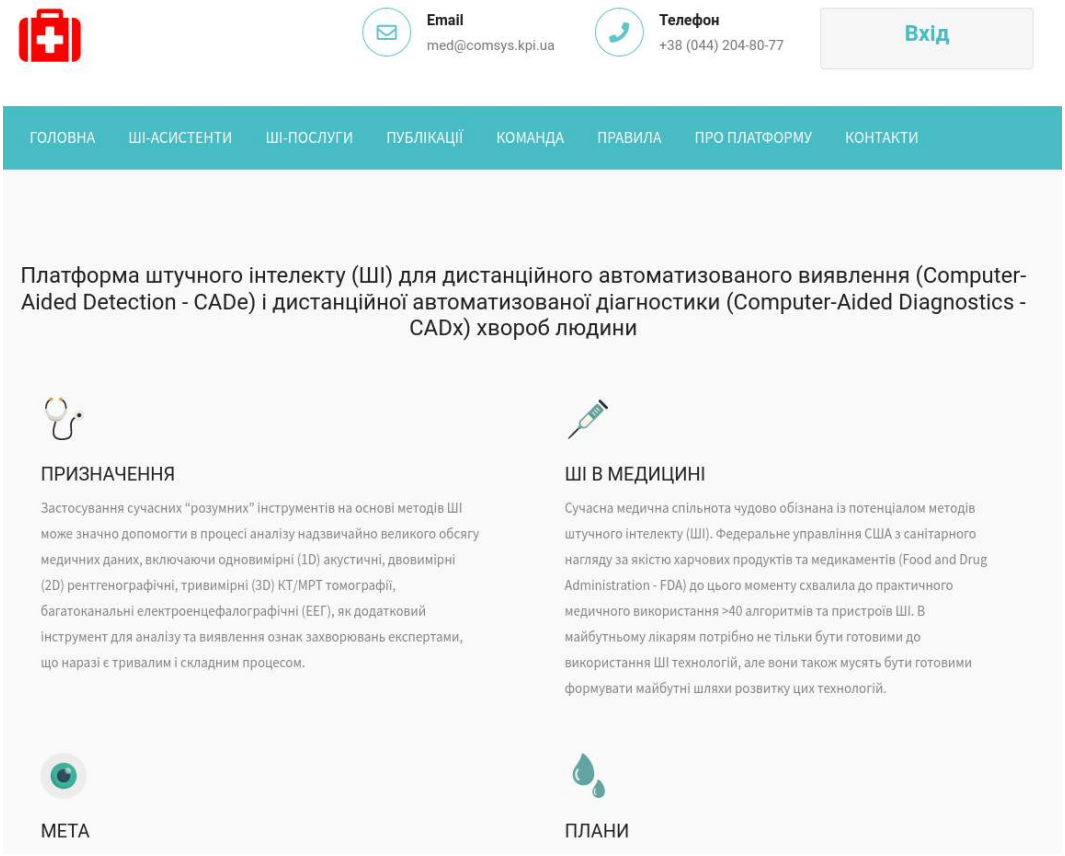


Рисунок 3.2 — Загальний опис ШІ-платформи

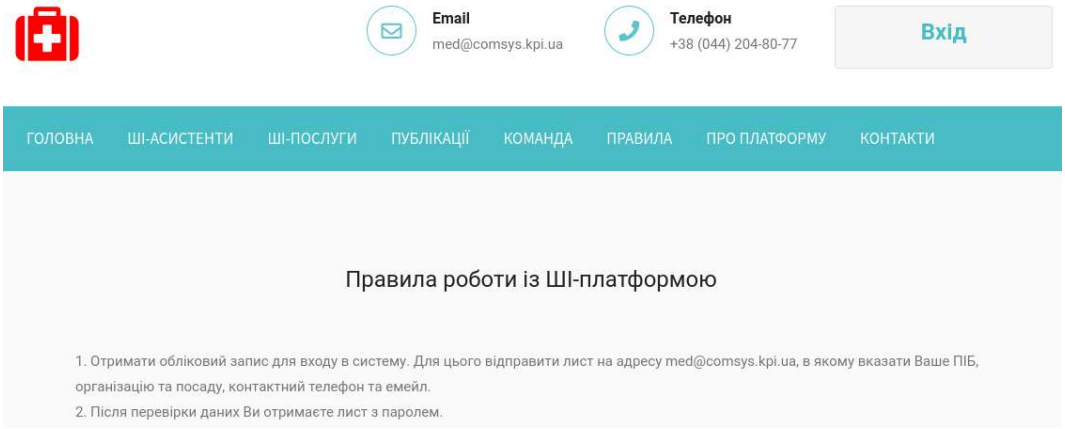


Рисунок 3.3 — Правила роботи із ШІ-платформою

(в) Описом ШІ-асистентів лікаря (рис. 3.4): https://med.comsys.kpi.ua/assistant

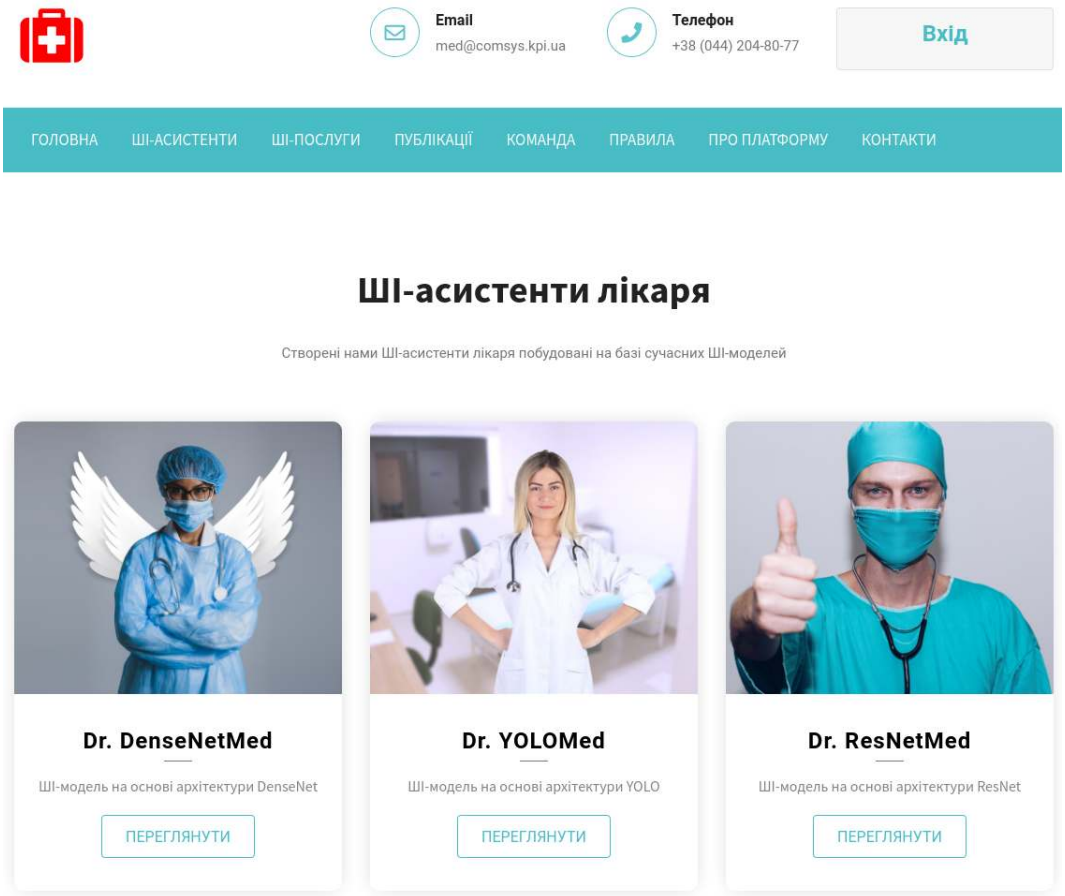


Рисунок 3.4 — Опис ШІ-асистентів лікаря

(г) Описом ШІ-послуг (рис. 3.5):
https://med.comsys.kpi.ua/service

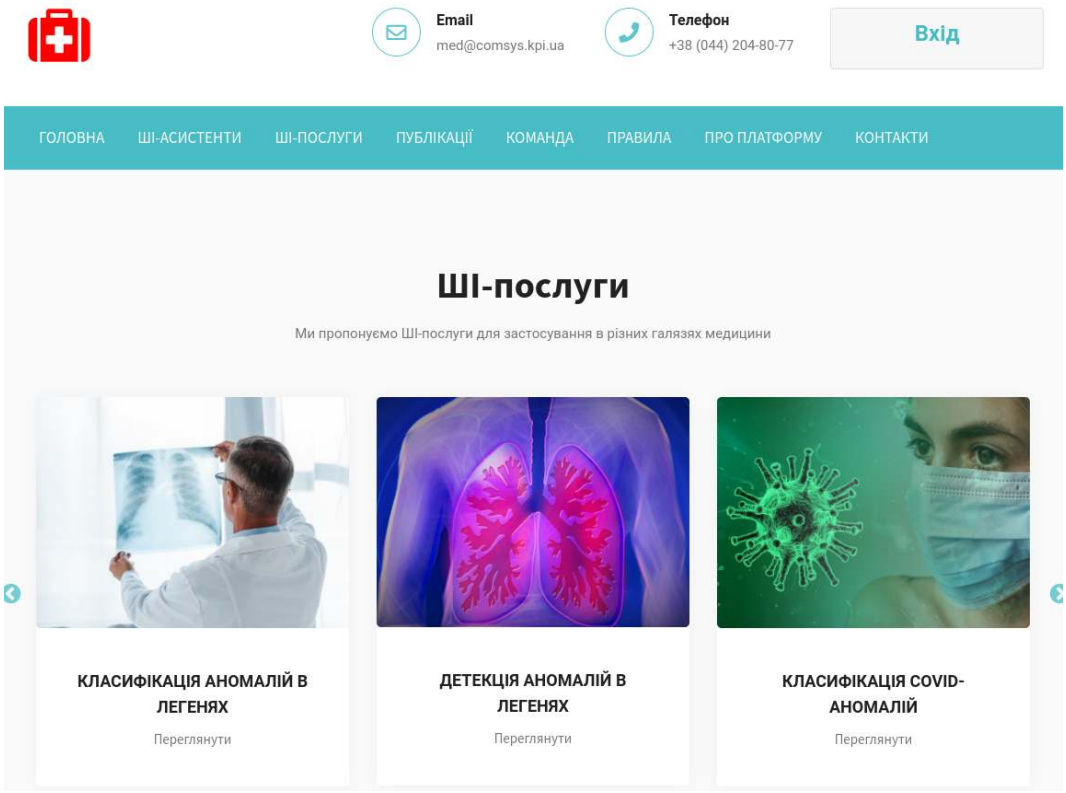


Рисунок 3.5 — Опис ШІ-послуг

(д) Переліком публікацій щодо наукових досліджень ШІ-моделей, які опубліковано у фахових наукових виданнях (рис. 3.6): https://med.comsys.kpi.ua/publication

(е) Колективом розробників ШІ-платформи (рис. 3.7): https://med.comsys.kpi.ua/team

3. Для зворотного зв'язку із колективом розробників ШІ-платформи заповнити форму (рис. 3.8): https://med.comsys.kpi.ua/contact
4. Для отримання доступу до повного функціоналу ШІ-платформи потрібно здійснити вхід в особистий кабінет (рис. 3.9). Тому, Вам потрібно отримати обліковий запис для входу в систему. Для цього відправте лист на наступну адресу: `med@comsys.kpi.ua`

   "*У листі вкажіть Ваше ПІБ, організацію та посаду, наміри щодо використання ШІ-платформи, контактний телефон та емейл. Після перевірки вказаної Вами інформації, на Вашу електронну адресу буде надіслано протягом 5-ти робочих днів лист з логіном та паролем для входу.*

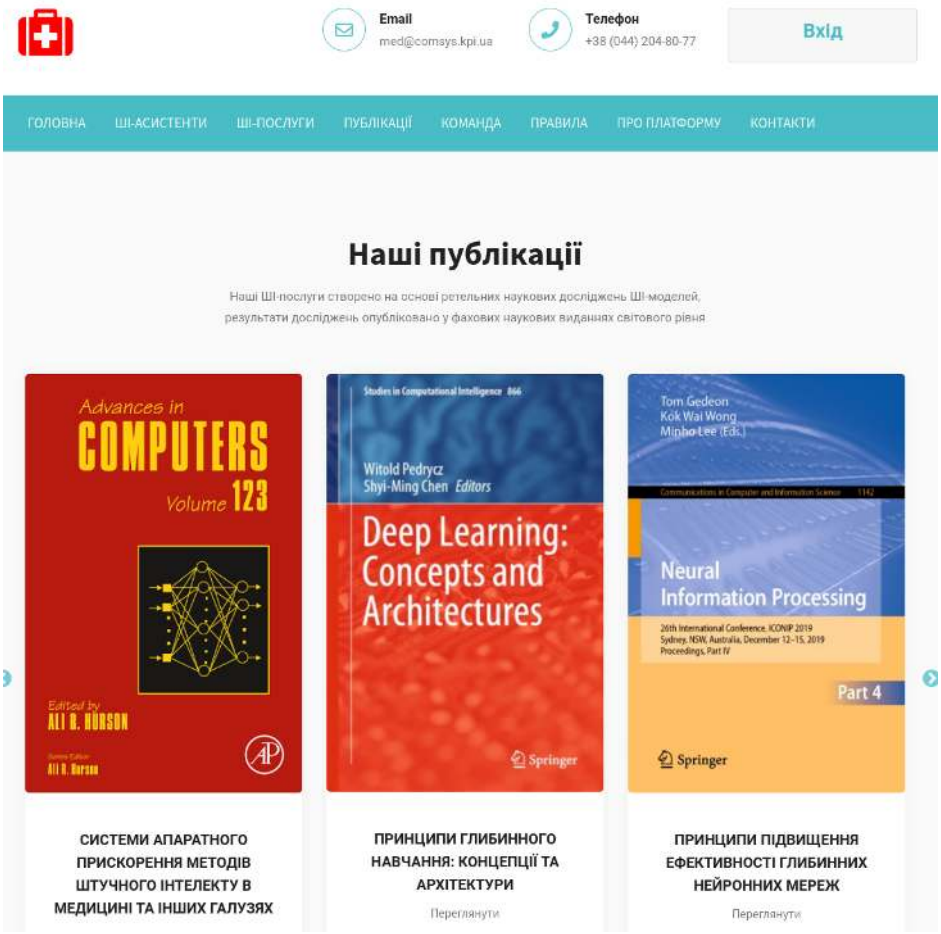


Рисунок 3.6 — Перелік наших публікацій

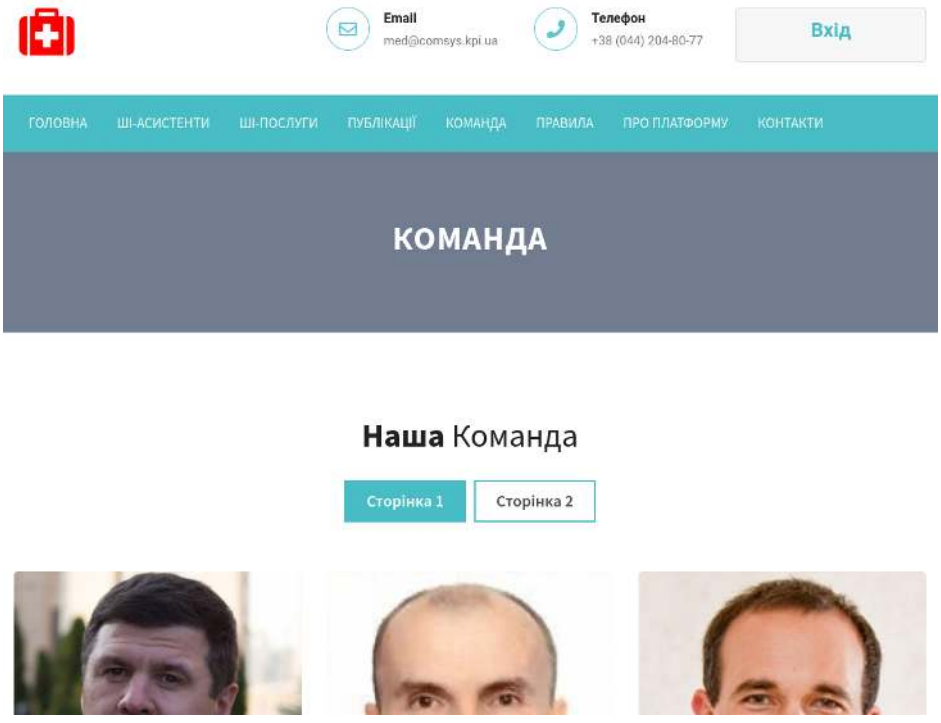


Рисунок 3.7 — Колектив розробників ШІ-платформи

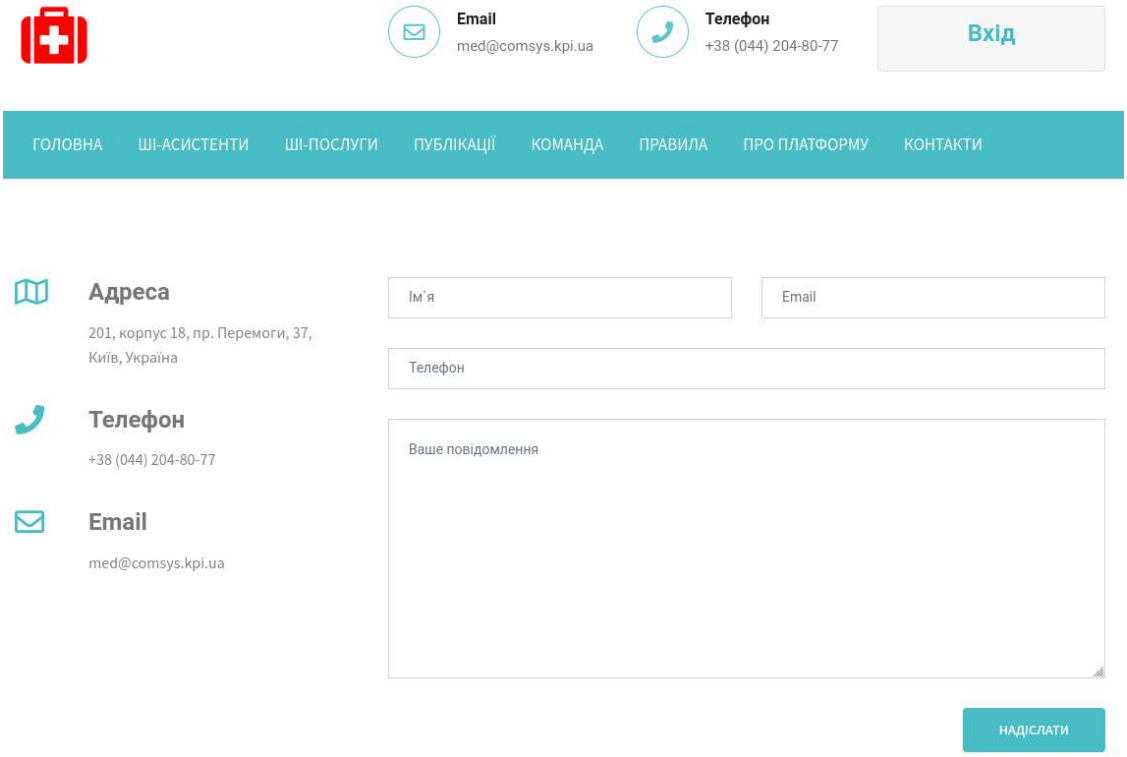


Рисунок 3.8 — Форма зворотного зв'язку із колективом розробників ШІ-платформи

5. Вхід у систему: натисніть кнопку **Вхід**, яка знаходиться у правому верхньому куті головної сторінки і введіть логін та пароль, який отримали на свою електронну адресу (рис. 3.9).

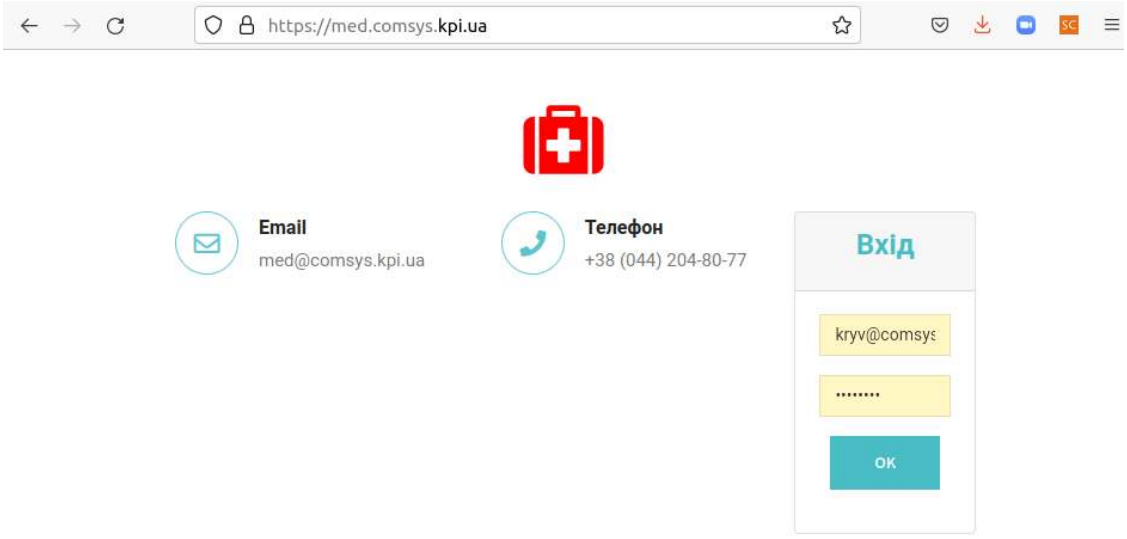


Рисунок 3.9 — Вхід у систему за логіном та паролем

6. Після входу в систему на екрані з'явиться інформація про минулі дослідження[2] та їх результати (рис. 3.10). Тепер можна обрати існуючі дослідження і знову їх переглянути або створити нове.

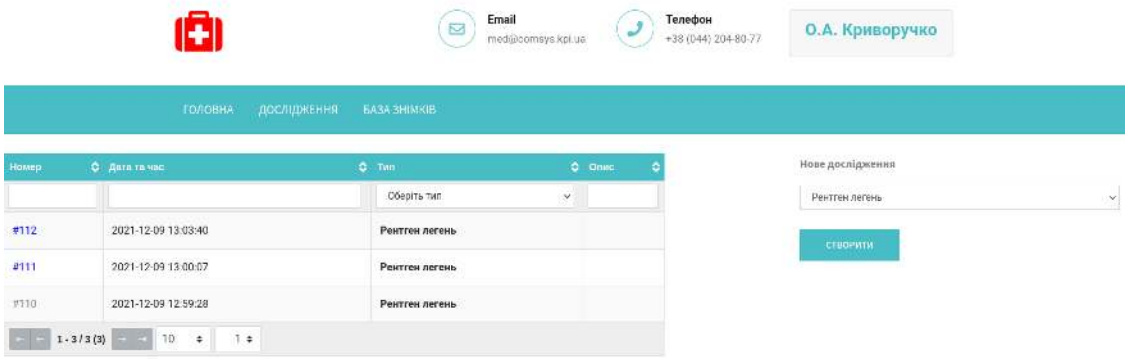


Рисунок 3.10 — Стан ШІ-платформи після входу в систему доктора Криворучка О.А. (демонстраційний акаунт)

(а) Для створення нового дослідження потрібно обрати пункт меню `Дослідження`, далі у випадному списку в правій частині сторінки обрати тип дослідження з списку `Нове дослідження` та натиснути кнопку `Створити`. Після створення дослідження, система перенаправить Вас на сторінку редагування створеного дослідження (рис. 3.11). Наразі доступні такі типи досліджень: рентген легень та діагностика хвороб шкіри.

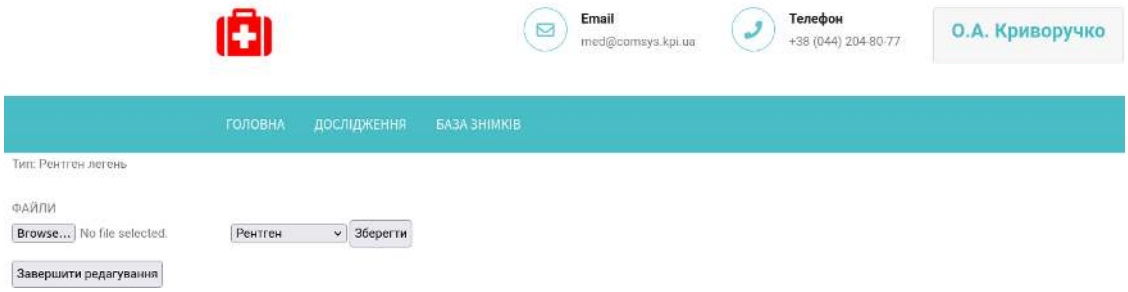


Рисунок 3.11 — Стан ШІ-платформи після створення дослідження типу `Рентген` (демонстраційний акаунт)

В залежності від типу запланованого дослідження потрібно додати відповідні файли. Для досліджень `Рентген` необхідно завантажити одну рентгенограму легень (файл у фор-

[2] Інформація про минулі дослідження не відображаться у таких випадках: при першому вході у систему та коли лікарем не було створено та не виконано жодного дослідження.

маті `.png` або `.jpg`). Додатково можна завантажити текстові файли з описом цього знімка. Для досліджень `Діагностика хвороб шкіри` необхідно завантажити фото шкіри (файл у форматі `.png` або `.jpg`). Додатково можна завантажити текстові файли з описом.

(б) Для того, щоб завантажити файл, потрібно натиснути на кнопку `Browse...` (назва може залежати від мови інтерфейсу браузера) та обрати потрібний файл (рис. 3.12).

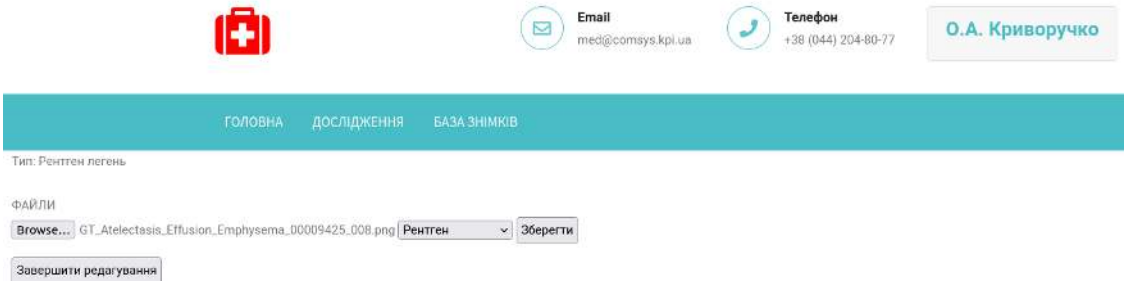


Рисунок 3.12 — Стан ШІ-платформи після вибору файлу

(в) Далі потрібно у випадному списку вказати тип файлу та натиснути кнопку `Зберегти`. Після додавання всіх файлів натиснути кнопку `Завершити редагування`.

(г) Для обрання існуючого дослідження, потрібно вибрати на панелі пункт `Дослідження` та натиснути на номер дослідження в таблиці (перший зліва стовпчик) (рис. 3.10). Заявки в таблиці можна впорядковувати за: номером, датою/часом створення, типом та описом.

Після вибору номера дослідження, яке Вас цікавить, (наприклад дослідження № 112 (рис. 3.10)), відобразиться перелік заявок в рамках цього дослідження (рис. 3.13), які було виконано для відповідного медичного знімка (тут `GT_Atelectasis_00009465_003.png`). Для перегляду результатів кожної із заявок потрібно обрати номер заявки.

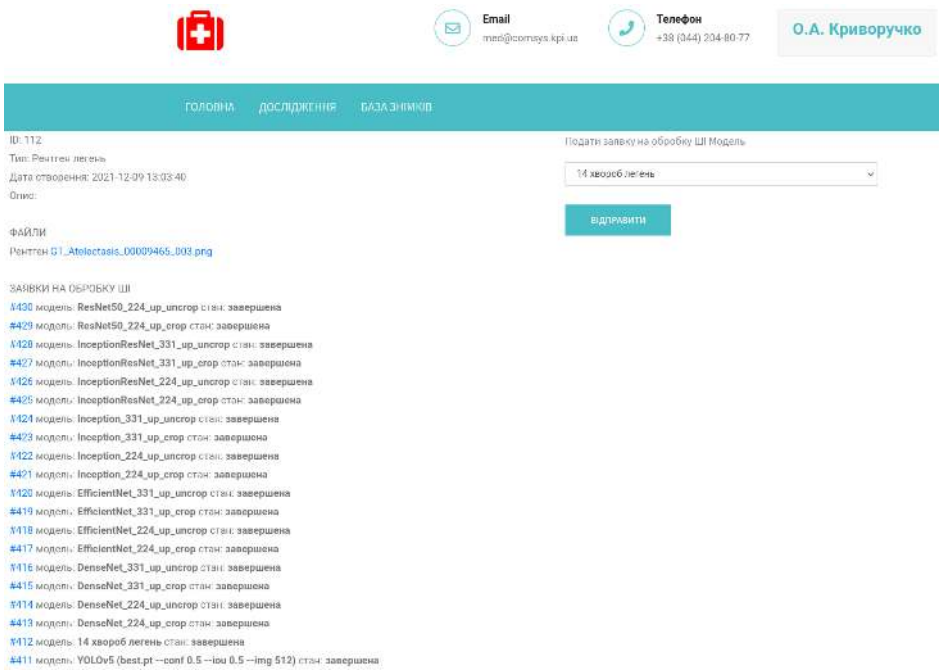


Рисунок 3.13 — Перелік виконаних заявок для дослідження з номером № 112

7. Після створення нової чи вибору існуючої заявки на дослідження, відобразиться сторінка з інформацією про цю заявку (наприклад, для заявки № 412 (рис. 3.14)), на якій вказано:
   (а) Номер дослідження (ID).
   (б) Статус дослідження (Стан).
   (в) ШІ-модель (Модель).
   (г) Дата створення.
   (д) Дата початку обробки.
   (е) Дата закінчення обробки.
   (ж) Результати обробки.
   (и) Перелік прикріплених файлів із результатами обробки (Результати обробки - Файли).
   (к) Перелік заявок на обробку ШІ-асистентами для цього дослідження (якщо такі є).

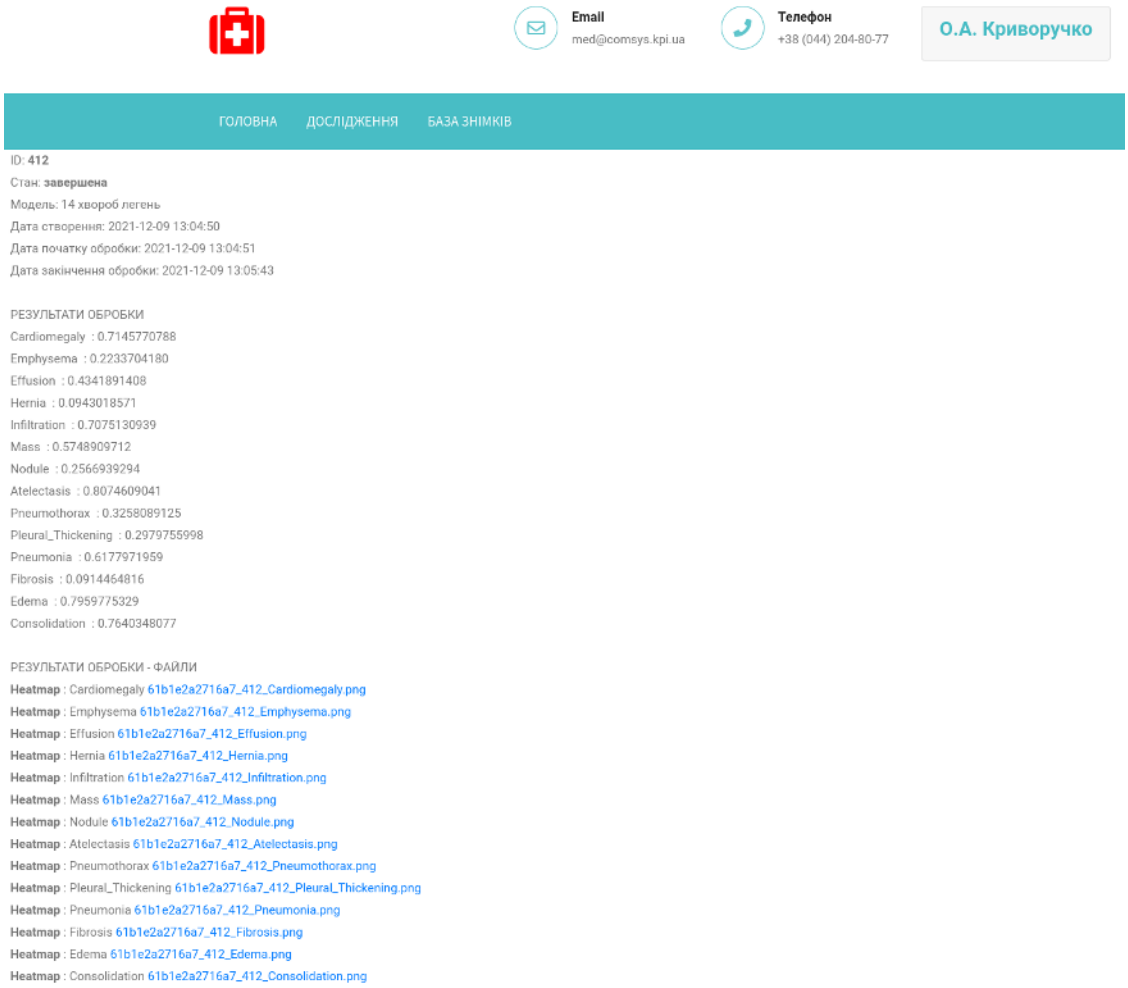


Рисунок 3.14 — Стан ШІ-платформи після вибору заявки за номером (№ 412)

8. Для подання нової заявки на обробку даних ШІ-асистентами потрібно у випадному списку `Подати заявку на обробку ШІ Модель` обрати відповідного ШІ-асистента та натиснути кнопку `Відправити` (рис. 3.13). Після чого створиться нова заявка, якій буде присвоєно номер[3] та буде поставлено в чергу на виконання. Стан заявки буде змінюватись у такому порядку: `нова` → `в обробці` → `завершена`.
9. Для перегляду результатів обробки[4], натисніть на номер заявки для якої завершено процес обробки зі списку заявок (стовпчик зліва, (рис. 3.13)). Після чого відкриється вікно з детальною інформацією про заявку (рис. 3.14).

   Формат виводу результатів обробки залежить від обраного ШІ-асистента. Результати обробки можуть бути представлені у вигляді:

[3] Створена заявка відобразиться у загальному списку заявок.

[4] Висновки прогнозу моделі на основі поданої їй на вхід інформації.

(а) `Ймовірності` виявлених аномалій ШІ-послугою для задачі класифікації **всього** знімку по ознаках 14 хвороб легенів (рис. 3.15).

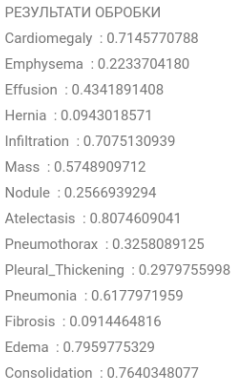


Рисунок 3.15 — Ймовірності аномалій для класифікації всього знімку по ознаках 14 хвороб легенів

(б) `Ймовірності` виявлених аномалій ШІ-послугою для задачі класифікації **всього** знімку по ознаках COVID-2019 (рис. 3.16).

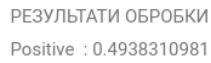


Рисунок 3.16 — Ймовірність аномалії для класифікації всього знімку по ознаках COVID-2019

(в) `Файл-зображення` з ймовірними зонами розташування виявлених ознак аномальності (heatmap) (рис. 3.17).

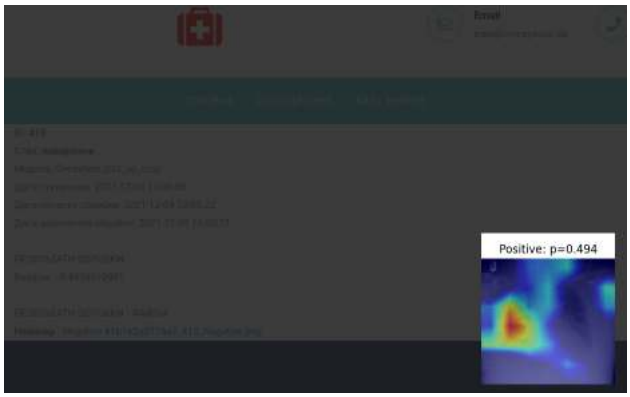


Рисунок 3.17 — Файл-зображення з ймовірними зонами розташування ознак аномальності (heatmap)

(г) `Ймовірності` відповідних аномалій в ШІ-послузі для ідентифікації зон можливих аномалій і класифікації аномалій в даних зонах (рис. 3.18) по ознаках 14 хвороб легенів.

Рисунок 3.18 — Ймовірності аномалій для класифікації всього знімку для ідентифікації зон можливих аномалій і класифікації аномалій в даних зонах по ознаках 14 хвороб легенів

(д) `Файл-зображення` з зонами (bounidng boxes) ймовірних аномалій, назвами аномалій та ймовірностями їх присутності (рис. 3.19).

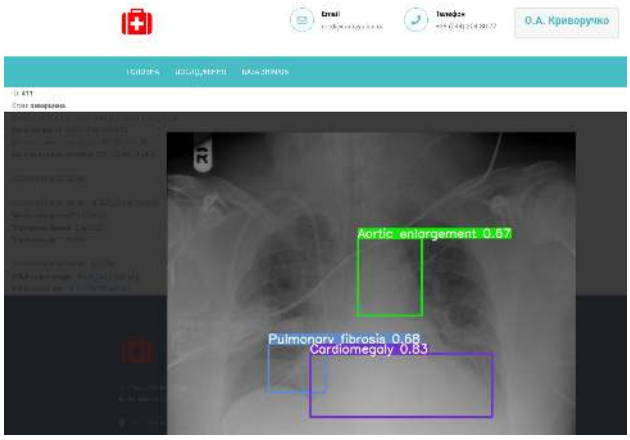


Рисунок 3.19 — Файл-зображення з зонами (bounidng boxes) ймовірних аномалій, назвами аномалій та ймовірностями їх присутності

(е) `Текстовий файл` з назвами виявлених ознак аномалій, ймовірністю та місцем їх розташування або комбінація вказаних варіантів. Для того, щоб переглянути вміст файлу з результатом, потрібно натиснути на його назву.

10. Вийти з системи після завершення роботи. Для цього потрібно натиснути на ПІБ в правому верхньому кутку сторінки та натиснути **Вихід**.

3.3 Опис функціональних можливостей платформи

3.3.1 ШІ-послуги

До переліку ШІ-послуг (рис. 3.5) входять:

1. Класифікація аномалій в легенях по 14 хворобах.
2. Детекція аномалій в легенях по 14 хворобах.
3. Класифікація аномалій в легенях по ознаках COVID-2019.
4. Класифікація аномалій шкіри (меланома).
5. Класифікація аномалій шкіри (хвороба Лайма).
6. класифікація аномалій в клітинах (гістологія).

Класифікація аномалій в легенях по 14 хворобах

Детальний опис процесу класифікації аномалій в легенях по 14 хворобах подано в наших попередніх публікаціях [9]—[12] із додатковими дослідженнями щодо підготовки даних та моделей для тренування, валідації та тестування, які описано в РОЗДІЛ **??** та РОЗДІЛ 2 на основі датасетів: ChestX-ray14 та локального датасету від субконтракторів медичного профілю, які описано в розділах РОЗДІЛ **??** та РОЗДІЛ **??**.

Класифікація аномалій в легенях по ознаках COVID-2019

Детальний опис процесу класифікації аномалій в легенях подано із додатковими дослідженнями щодо підготовки даних та моделей для тренування, валідації та тестування, які описано в підрозділах: 2.1 та 2.2 на основі датасетів із COVID-2019-аномаліями (ieee8023) та локальному датасеті від субконтракторів медичного профілю, які описано в підрозділах: **??**, **??** та **??**.

Класифікація аномалій шкіри (меланома)

Детальне дослідження щодо класифікації аномалій шкіри (меланома) описано в нашій публікації [79] із коротким викладом особливостей підготовки моделей для тренування, валідації та тестування, які описано в підрозділі 2.8 та особливостей підготовки даних на основі датасетів ISIC, які описано в підрозділі **??**.

В подальшій роботі на основі цієї ШІ-платформи передбачається створити загальний каркас для нанизування-додавання автоматизованих ШІ медичних сервісів з використанням наявних локальних та глобальних медичних даних (МРТ, КТ, ЕЕГ, ультразвуку, рентгенографії, оптичних методів, аналізу часових послідовностей показників датчиків) для полегшення процедури виявлення аномалій в органах та діагностики потенційних хвороб на ранніх стадіях їх виникнення. Фактично ШІ-платформа є прототипом інфраструктури для інтеграції сучасних та майбутніх медичних сервісів на основі ШІ-технологій для виявлення найбільш небезпечних захворювань в їх ранніх і часто безсимптомних стадіях (скринінгу), на яких лікування ефективніше, максимально поширити обсяг охоплення пацієнтів сучасними медичними технологіями, вивести скринінгові, профілактичні та діагностичні процедури на персоналізований рівень, та зменшити навантаження на медичний персонал (сімейних лікарів та медсестер).

В якості прикладів нанизування-додавання таких додаткових автоматизованих ШІ-послуг було розглянуто:

1. Класифікацію аномалій шкіри (хвороба Лайма).
2. Класифікацію аномалій в клітинах (гістологія).

Класифікація аномалій шкіри (хвороба Лайма)

Цей приклад ШІ-послуг не входив в технічне завдання проєкту і виник, як додатковий варіант застосування ШІ-платформи та розширення його функціоналу в майбутньому. На основі дослідження різноманітних ШІ-моделей (вивчених і тренованих в рамках даного проєкту архітектур типу Inception, ResNet, DenseNet, EfficientNet, MobileNet) виникла можливість їх застосування в режи-

мі transfer learning для інших типів захворювань типу хвороби Лайма. Наразі цей функціонал доступний на ШІ-платформи, але не описаний у звіті, оскільки не входив в технічне завдання проєкту і знаходиться в стадії тестування та покращення датасету. Він розглядається тут як приклад подальшої диверсифікації ШІ-методів та ШІ-моделей, які було досліджено в рамках даного проєкту.

Сегментація аномалій в клітинах (гістологія)

Цей приклад ШІ-послуг також не входив в технічне завдання проєкту і виник, як додатковий варіант застосування ШІ-платформи та розширення його функціоналу в майбутньому. На основі дослідження різноманітних ШІ-моделей (вивчених і тренованих в рамках даного проєкту архітектур типу U-Net) виникла можливість їх застосування в режимі transfer learning для сегментації об'єктів дослідження на клітинному рівні по даних мікроскопічних досліджень. Наразі цей функціонал доступний на ШІ-платформи, але не описаний у звіті, оскільки не входив в технічне завдання проєкту і знаходиться в стадії тестування та розширення датасету. Він теж розглядається тут як приклад подальшої диверсифікації ШІ-методів та ШІ-моделей, які було досліджено в рамках даного проєкту. Детальне дослідження щодо покращення методів сегментації об'єктів, зокрема медичного характеру, описано в наших публікаціях [76], [77] із коротким викладом особливостей підготовки моделей для тренування, валідації та тестування, які описано в підрозділі 2.6.

3.3.2 ШІ-асистенти лікаря (ШІ-моделі)

Для кожної із розглянутих вище ШІ-послуг було досліджено набори ШІ-моделей, які в рамках даної ШІ-платформи грають роль ШІ-асистентів лікаря.

DenseNet. Архітектура DenseNet (Densely Connected Convolutional Networks) [81] описана в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі саме з цією архітектурою давали одні з найкращих показників якості прогнозів для деяких прикладів медичного застосування [9]—[12] і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація по 14 типах хвороб в легенях.
2. Класифікація ознак COVID.
3. Класифікація ознак меланоми.

MobileNet. Архітектура MobileNet [82] також описана в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі саме з цією архітектурою давали одні з найбільш високих показників при застосуванні для датасетів невеликого розміру і необхідності застосування ШІ-моделей для пристроїв периферійних обчислень (Edge Computing) [83], [84], які описані в підрозділі **??**. Кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація ознак меланоми.
2. Класифікація ознак хвороби Лайма.
3. Класифікація ЕЕГ-сигналів діяльності мозку.

InceptionNet. Архітектура InceptionNet [85], [86] описана в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі саме з цією архітектурою давали одні з найбільш стабільно високих (навіть при при різних розмірах вхідних зображень) показників якості прогнозів для деяких прикладів медичного застосування (максимальні при класифікації ознак COVID) і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація ознак COVID.
2. Класифікація ознак меланоми.

ResNet. Архітектура ResNet [87] описана також в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі з цією архітектурою давали стабільно високі (але не завжди максимальні) показники якості прогнозів для деяких прикладів медичного застосування і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря та для пристроїв периферійних обчислень (Edge Computing) [83], [84], які описані в підрозділі **??**:

1. Класифікація по 14 типах хвороб в легенях (лише для апробації використання на пристроях периферійних обчислень – Edge Computing).
2. Класифікація ознак COVID.
3. Класифікація ознак меланоми.

InceptionResNet. Опис архітектури InceptionResNet [86] подано в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі з цією архітектурою давали високі значення (але з більшою чутливістю точності прогнозу щодо різних розмірів вхідних зображень) показників якості прогнозів для деяких прикладів медичного застосування і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація ознак COVID.
2. Класифікація ознак меланоми.

VGG. Архітектура VGG [88] описана в підрозділі **??**. Після досліджень в рамках проєкту треновані ШІ-моделі з цією архітектурою давали стабільно високі (але не завжди максимальні) показники якості прогнозів для деяких прикладів медичного застосування, також для спеціалізованих архітектур, які виявилися ефективними при роботі із датасетами малого розміру, і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація по 14 типах хвороб в легенях (лише для апробації використання на пристроях периферійних обчислень – Edge Computing).
2. Класифікація ознак COVID.
3. Класифікація ознак меланоми.

Xception. Архітектура Xception [89] описана в підрозділі **??**. Базові моделі глибинного навчання, підрозділі Xception. Після досліджень в рамках проєкту треновані ШІ-моделі з цією архітектурою давали високі значення (але з більшою чутливістю точності прогнозу щодо різних розмірів вхідних зображень) показників якості прогнозів для деяких прикладів медичного застосування і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація ознак COVID.
2. Класифікація ознак меланоми.

EfficientNet. Архітектура EfficientNet [90] описана в підрозділі **??**. Базові моделі глибинного навчання, підрозділі EfficientNet. Після досліджень в рамках проєкту треновані ШІ-моделі з цією архітектурою давали різноманітні показники якості прогнозів для деяких прикладів медичного застосування, які більшою мірою залежали від конкретної версії (від B0 до B6) із різними розмірами і

складністю. Це дозволяло спланувати їх для використання в рамках обраного компромісу між точністю прогнозу та вимогами до обчислювальних ресурсів спеціалізованих архітектур на рівні периферійних обчислень. Найменші версії цієї архітектури виявилися ефективними при роботі із датасетами малого розміру, і тому саме кілька варіантів цієї мережі було обрано для застосування на платформі в складі наступних ШІ-асистентів лікаря:

1. Класифікація ознак COVID.
2. Класифікація ознак меланоми.

На основі отриманих та опублікованих результатів ШІ-моделі на основі вказаних архітектур планується використовувати також для інших прикладів медичного застосування.

Варто відзначити, що точність прогнозу може варіюватися при застосуванні різних ШІ-моделей навіть при застосування до однакових вхідних медичних даних. Тому в якості отримання більш достовірних прогнозів в майбутньому для подальших досліджень варто використовувати гібридні конфігурації ШІ-моделей, які будуть відігравати роль "консиліуму" ШІ-моделей.

Точність прогнозу також може варіюватися в залежності від їх розмірів (наприклад, розмірів вхідного рентгенівського зображення легенів), способу та умов отримання вхідних медичних даних, виконаних етапів попередньої обробки. Тому для оцінки відхилення прогнозу в залежності від наявних факторів впливу, операції штучного розширення обсягу даних (які зазвичай використовуються на етапі тренування) варто включати на етапі прогнозу (test time augmentation), як це описано в підрозділі **??**.

3.4 Компоненти платформи

Для реалізації функціональних можливостей ШІ-платформи на основі комбінацій ШІ-послуга + ШІ-модель було створено демо-версію ШІ-платформи на основі наступних компонентів (рис. 3.20):

1. Сервер керування та представлення даних.
2. Сервери обробки даних.
3. Сервери тренування ШІ-асистентів (моделей).
4. Система збереження даних.

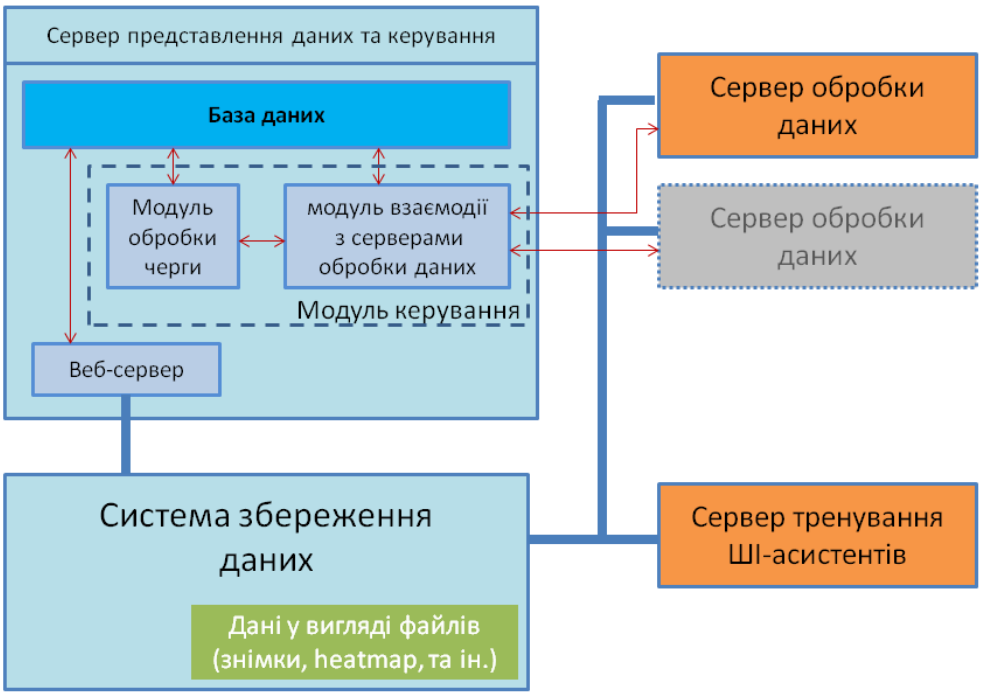


Рисунок 3.20 — Компоненти ШІ-платформи

3.4.1 Сервер керування та представлення даних

Сервер керування та представлення даних – це фізичний сервер або віртуальна машина, який відповідає за надходження даних в систему, представлення та зберігання результатів обробки, керування чергою обробки заявок в системі. На сервері керування та представлення даних розміщені такі модулі:

1. Веб-сервер, який обслуговує веб-сайт, через який користувачі можуть завантажити дані в систему, переглянути наявні дані, відправити заявку на обробку, та отримати результати. Інтерфейс користувача веб-сайту детально описаний у параграфі 3.2.
2. Модуль керування, який можна поділити на дві частини:
    (a) Модуль обробки черги, який формує та обробляє чергу заявок на обробку даних. Він визначає порядок обробки заявок, формує пакет з декількох заявок для пакетної обробки.

(б) Модуль взаємодії з серверами обробки, який реалізує інтерфейс (API) між програмним забезпеченням на сервері обробки даних, яке безпосередньо запускає заявки на виконання (локальний планувальник), та модулем керування на сервері представлення даних та керування (глобальний планувальник). Інтерфейс, який реалізовано, являє собою REST API, який, зокрема, дозволяє отримати номери завдань з черги, які має виконувати певний контейнер, повідомити про зміну стану виконання задач, відправити код результату та імена файлів з результатом. Самі файли при цьому передаються не на сервер керування, а напряму в систему збереження даних. Це дозволяє оптимізувати процес обміну даними та збільшити швидкість обробки заявок.

3. База даних в якій зберігаються дані щодо заявок поданих на виконання в системі та допоміжні (службові) дані.

3.4.2 Сервер обробки даних

Сервер обробки даних – це фізичний сервер з GPU або віртуальна машина, яка монопольно використовує GPU. На рисунку 3.21 показано порядок проходження заявок на обробку з черги від глобального планувальника до локального, та надходження до контейнерів для обробки.

Для повноцінної роботи та забезпечення всіх можливостей GPU має підтримувати технологію Multi-Instance GPU (MIG), деталі якої описані в наступному підрозділі 3.5. Використання GPU без підтримки MIG теж можливе, але у такому випадку один сервер зможе обробляти одночасно заявки тільки для одного типу ШІ-асистента з певними налаштуваннями. На даний час користувачу платформи надається можливість працювати з 6 ШІ-асистентами та декількома параметрами для кожного, всього приблизно 30 варіантів налаштувань ШІ-асистентів (тобто комбінацій ШІ-асистент+ШІ-модель), ця кількість буде збільшуватись в майбутньому. Локальний планувальник взаємодіє з глобальним планувальником (фактично – модулем обробки черги) за допомогою REST API (його реалізує модуль взаємодії з серверами обробки). Він повідомляє глобальному планувальнику про зміну стану заявок, а при завершенні обробки

заявки (або пакету заявок, в залежності від налаштованого способу обробки) та звільнені ресурсів, звертається до глобального планувальника за новими заявками на обробку.

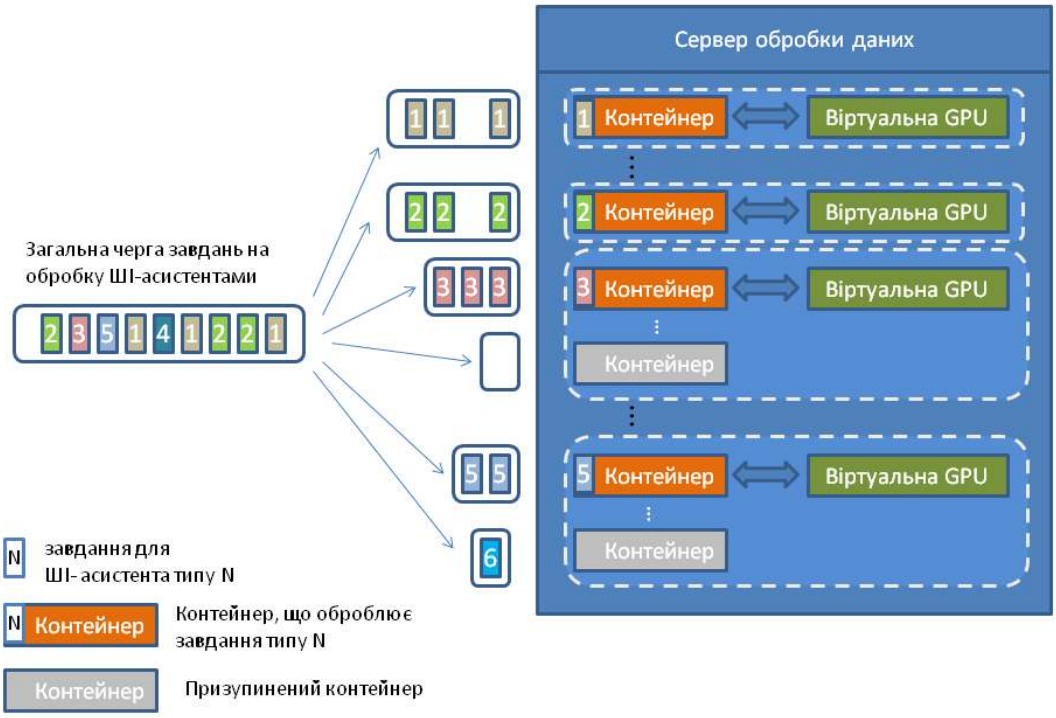


Рисунок 3.21 — Порядок проходження заявок на обробку з черги від глобального планувальника до локального, та надходження до контейнерів для обробки

В залежності від типу ШІ-асистента локальний планувальник передає заявки на обробку відповідному контейнеру (якщо він зупинений, то запускає його). У випадку великої кількості заявок для певного типу ШІ-асистента контейнер не зупиняється та постійно очікує надходження заявок. У випадку невеликої кількості заявок для певного типу ШІ-асистента локальний планувальник зупиняє контейнер після обробки пакету заявок, щоб звільнити ресурси для іншого контейнера (з іншим типом ШІ-асистента). Рішення про розподіл заявок по серверам обробки (якщо їх декілька) приймається на рівні глобального планувальника. Рішення про запуск відповідного контейнера приймається на рівні локального планувальника. Глобальний планувальник (модуль обробки черги) може опосередковано впливати на запуск відповідних контейнерів на

конкретному сервері обробки шляхом вибору певних заявок з глобальної черги. Наприклад, при отриманні запиту по API з сервера обробки він може передати йому тільки заявки для ШІ-асистентів певного типу, а інші залишити в глобальній черзі. Алгоритм вибору заявок з глобальної черги може враховувати такі параметри, як пріоритетність заявок, середній час проходження заявок у системі (взагалі та по конкретних типах), завантаженість серверів обробки, та багато інших. Оптимальний перелік та порогові значення параметрів можуть бути предметом подальших досліджень.

3.4.3 Сервер тренування ШІ-асистентів

Сервер тренування ШІ-асистентів — це окремий фізичний сервер з GPU (або віртуальна машина, яка монопольно використовує GPU), який призначений для покращення якості ШІ-асистентів (навчання моделей нейронних мереж).

Сервер не підключений до модуля обробки черги та не оброблює поточні заявки в системі. Він, в залежності від налаштувань, використовує дані з сформованого датасету, що розміщений в системі збереження даних, або накопичені дані з файлів, які були завантажені через веб-сервер.

3.4.4 Система збереження

Система збереження – це окремий фізичний сервер (або віртуальна машина з монопольним використанням дискового простору великого об'єму), на якому зберігаються файли, які оброблюються ШІ-асистентами (наприклад, рентген знімки легень, фотознімки шкіри), а також результати обробки (наприклад, зображення з позначеними місцями ймовірних аномалій та їх назвами).

3.5 Технологія конфігурування та використання великої кількості екземплярів спеціалізованого прискорювача GPU-типу

Для паралельного виконання кількох робочих навантажень на одному GPU було використано технологію запуску великої кількості екземплярів спеціалізованого прискорювача на базі графічного процесора NVIDIA A100. Нова технологія запуску великої кількості екземплярів спеціалізованого прискорювача (Multi-Instance GPU – MIG) дозволяє паралельно і повністю ізольовано запускати відразу кілька програм CUDA на спеціалізованому прискорювачі обчислень на базі графічного процесора (GPU). Обчислювальні блоки графічного процесора, а також його пам'ять можна розділити на кілька екземплярів MIG, де кожен з цих екземплярів може бути представлений як автономний пристрій GPU з точки зору операційної системи і може бути прив'язаний до будь-якої програми, контейнера або віртуальної машини, що працює на обчислювальному вузлі. На апаратному рівні екземпляр MIG має власні виділені ресурси (процесори, кеш, пам'ять), тому робоче навантаження, що виконується на одному екземплярі, не впливає на те, що виконується на інших.

Геометрія поділу ресурсів з MIG – це спосіб розподілу апаратних ресурсів серед екземплярів MIG, який безпосередньо впливає на їх продуктивність із зазначенням кількості екземплярів, які можуть бути створені із такими ресурсами. Наявний спеціалізований прискорювач NVIDIA A100, який ми використовували в рамках цього проєкту, має 8 обчислювальних блоків і 40 ГБ оперативної пам'яті. При використанні технології MIG, один з 8 екземплярів зарезервований для керування ресурсами і для поточних завдань доступні лише 7, як максимум.

У таблиці 3.1 подано короткий опис можливих варіантів геометрії MIG із описом властивостей відповідних екземплярів для NVIDIA A100-40GB.

Таблиця 3.1 — Порівняння основних параметрів у процесі навчання мереж

Назва екземпляру	Пам'ять екземпляру	Обчислювачі екземпляру	Максимальна кількість екземплярів
1g.5gb	5 ГБ	1	7
2g.10gb	10 ГБ	2	3
3g.20gb	20 ГБ	3	2
4g.20gb	20 ГБ	4	1
7g.40gb	40 ГБ	7	1

На додаток до однорідних екземплярів (таблиця 3.1) можна створювати деякі неоднорідні комбінації, наприклад, для A100-40GB можна використати таку неоднорідну (або «змішану») геометрію:

1. `1x 1g.5gb`
2. `1x 2g.10gb`
3. `1x 3g.20gb`

Зручність MIG полягає в тому, що адміністратор серверу має можливість змінювати геометрію MIG для оптимального використання ресурсів MIG-сумісних GPU-карт сервера і використовувати змінену геометрію MIG для нових віртуальних машин чи контейнерів.

В рамках проєкту було досліджено кілька варіантів геометрії MIG. Зроблено порівняльний аналіз продуктивності при використанні кількох різних способів конфігурування геометрії MIG під поточні вимоги запитів з боку лікарів для обробки медичних даних своїх пацієнтів різними типами комбінацій ШІ-послуга+ШІ-модель в практичних умовах. Після аналізу метрик продуктивності різних конфігурацій було зроблено короткий підсумок щодо використання конфігурації:

1. `1x 3g.20gb`
2. `1x 3g.20gb`

3.6 Оцінка способів оптимізації обробки медичних даних і доцільності масштабування обчислювальної інфраструктури

Загалом розробка методів оптимізації обробки медичних даних своїх пацієнтів різними типами комбінацій ШІ-послуга + ШІ-модель в практичних умовах є перспективним напрямком подальших досліджень в контексті масового використання ШІ-платформи в практичних умовах. Наприклад, перспективним є дослідження використання кількох різних стратегій конфігурування геометрії MIG, наприклад:

1. Фіксований розподіл ресурсів.
2. Статичний спосіб.
3. Динамічний спосіб:
    (а) Із жорсткими умовами зміни геометрії MIG.
    (б) Із гнучкими умовами зміни геометрії MIG.

(в) Із гнучкими умовами зміни геометрії MIG на основі методів машинного або глибинного навчання, що є надзвичайно перспективним для майбутніх досліджень (не входить в даний проєкт та звіт).

Варто зазначити, що ця проблема потребує перевірки великої кількості варіантів в залежності від конкретних практичних умов (кількості MIG-сумісних ресурсів, необхідної пропускної здатності, кількості ШІ-моделей, ШІ-асистентів, параметрів вхідних даних, тощо) та статистичної обробки великого обсягу даних.

В якості перших оціночних спроб було проведено перевірку практичної спроможності реалізації різних стратегій конфігурування геометрії MIG, які описані далі. На рисунках 3.22, 3.23 та 3.24 проілюстрована залежність довжини черги заявок на обробку певним ШІ-асистентам (варіантам нейронної мережі) від часу.

Фіксований розподіл ресурсів. Як видно з графіку (рис. 3.22), протягом короткого проміжку часу велика кількість заявок була згенерована та завантажена у чергу.

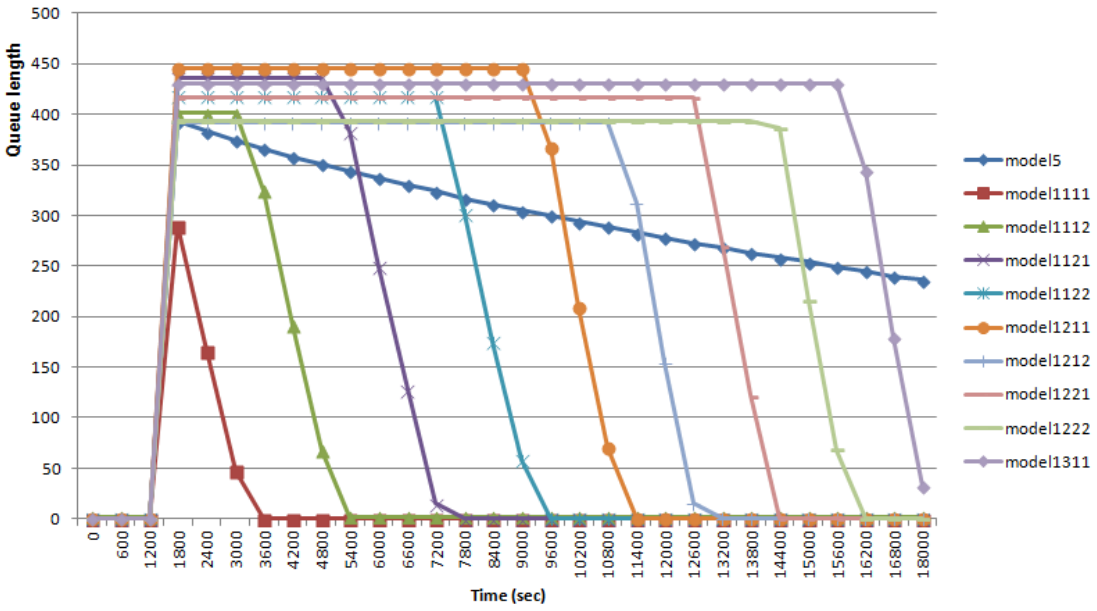


Рисунок 3.22 — Фіксований розподіл ресурсів

На рисунку 3.22 показано поведінку системи при отриманні одночасно великої кількості заявок різних типів. При цьому розподіл ресурсів відбувається так: глобальний планувальник передає локальному планувальнику всі завдання

для ШІ-асистента певного типу. Локальний планувальник при наявності вільного ресурсу запускає відповідний контейнер, який обробляє всі завдання в черзі. Після чого контейнер завершує виконання та звільняє ресурс. Ресурси не налаштовуються під розмір задачі. В цьому випадку деякі заявки користувачів можуть довго очікувати в черзі. Крім того, ресурси не завжди використовуються оптимально (може виділятися більше ресурсів, ніж потрібно). Цей спосіб планування є неприйнятним з точки зору користувачів.

Статичний спосіб розподілу ресурсів. На рисунку 3.23 показано поведінку системи при статичному налаштуванні системи на різні типи заявок.

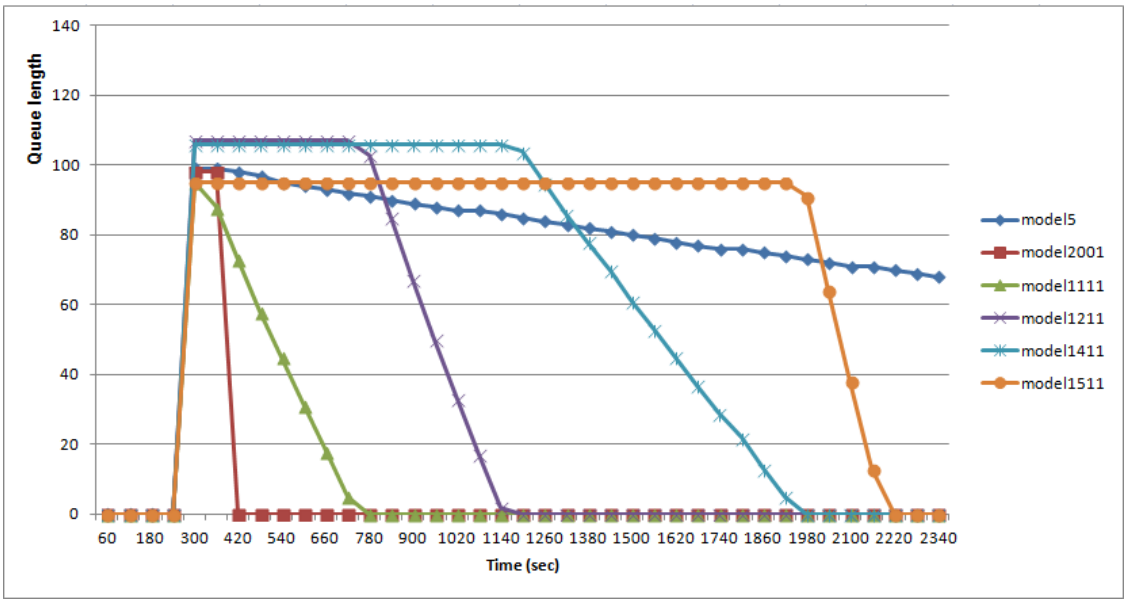


Рисунок 3.23 — Статичний розподіл ресурсів

GPU була розподілена на частини таким способом:

1. MIG `4g.20gb`
2. MIG `2g.10gb`
3. MIG `1g.5gb`

При цьому, відповідні віртуальні GPU використовують контейнери для заявок відповідного розміру.

Динамічний спосіб розподілу ресурсів. На рисунку 3.24 показано поведінку системи при можливості зміни конфігурації в залежності від кількості та типу заявок в черзі, а також призупиненні обробки локальної черги для тимчасового звільнення ресурсів для других заявок.

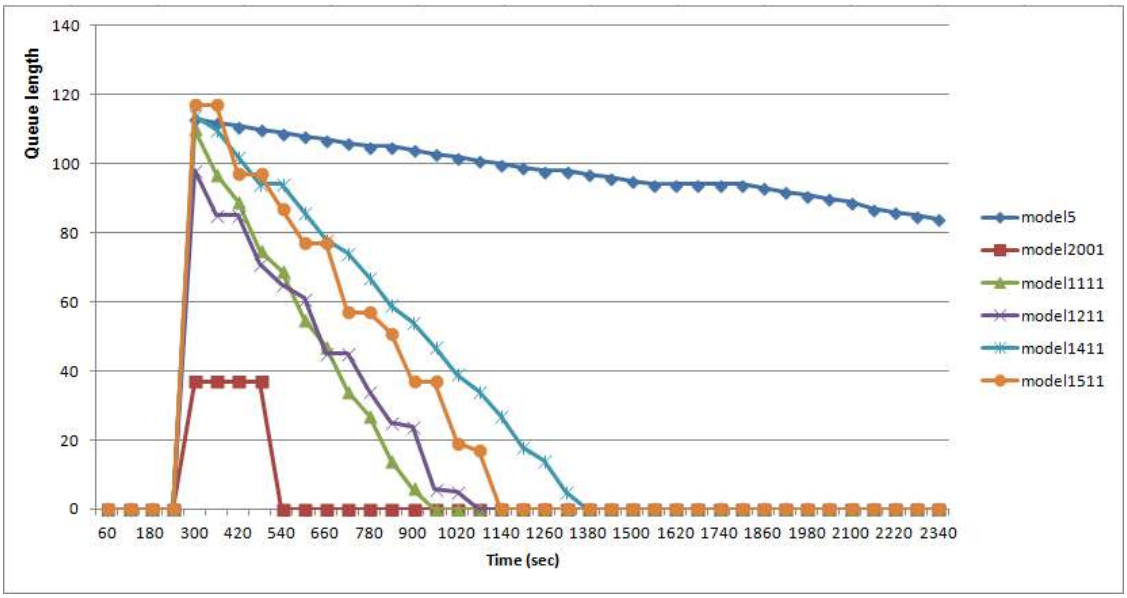


Рисунок 3.24 — Динамічний розподіл ресурсів

В цьому випадку завдання, що потребують багато ресурсів та занадто довго виконуються, можуть бути призупинені, налаштовані на більш повільні ресурси, для того пропустити вперед більш короткі заявки. Після цього всі ресурси (навіть ті,що раніше призначалися для коротких заявок), можуть бути налаштовані на виконання довгих заявок. Зміна конфігурації ресурсів можлива за допомогою технології MIG, яка описана раніше у підрозділі 3.5. В процесі, що показано на рисунку 3.24, відбулось переналаштування спочатку на розподіл MIG `1g.5gb`, `1g.5gb`, `2g.10gb`, `3g.20gb`, потім на `7g.40gb`.

З урахуванням оцінок, які було зроблено у підрозділі 2.3 та оцінки способів оптимізації обробки медичних даних в цьому розділі можна зробити висновок про доцільність використання запропонованої ШІ-платформи в більш глобальному масштабі для потреб мегаполісів, регіонів та країни в цілому. Загалом оцінки зроблені для конфігурації з одним спеціалізованим прискорювачем типу NVIDIA A100 на основі технології віртуалізації ресурсів MIG дозволяють сягнути запланованого рівня обробки до 103 (і навіть більше до 104) медичних знімків в секунду на ШІ-платформі при відповідному масштабуванню обчислювальної інфраструктури до 8 спеціалізованих прискорювачів типу NVIDIA A100 на основі технології віртуалізації ресурсів MIG.

Висновки до РОЗДІЛ 3

За результатами проведеного огляду сучасних методів прунінгу та квантизації для оптимізації продуктивності глибоких нейронних мереж було виявлено, що більшість досліджень зосереджені на використані одного з методів, або прунінгу, або квантизації, при роботі із нейронними мережами. При цьому результати поєднання обох методів для оптимізацію залишаються недостатньо дослідженими. Також, більшість сучасних досліджень обмежуються підвищенням продуктивності моделі глибокої нейронної мережі лише за допомогою зазначених вище методів, не оптимізуючи при цьому саму обчислювальну систему, до якої відноситься як апаратна так і програмна складова. Узагальнюючи отримані результати досліджень було запропоновано багаторівневу ієрархічну методологію оптимізації глибоких нейронних мереж для різних сфер застосування в контексті Edge Computing. На базі запропонованої методології було досліджено ефективність роботи прискорювача Coral Edge TPU USB Stick в медичному контексті для діагностики легеневих аномалій за даними рентген знімків. Важливим аспектом роботи розробленої портативної програми для діагностики легеневих аномалій є те, що дані рентген знімків нікуди не передаються і оброблюються локально на комп'ютері лікаря. Це дозволяє забезпечити захист персональних даних пацієнта. Також, використання даної портативної програми для діагностики дозволяє проводити аналіз рентген знімків в неідеальних польових умовах практичного використання, наприклад, за умов нестійкого Інтернет зв'язку або його повній відсутності, коли доступ до повнофункціональної платформи для діагностики захворювань не можливий.

У даному розділі описані результати роботи щодо тестування та аналізу результатів прогнозування (Prediction Tests and Analysis – PTA) для моделей DTL / ADL із використанням штучного збільшення різноманітності даних різного ступеню інтенсивності, дослідженння, розробки та тестування віртуальних версій компонентів ШІ-платформи для використання на придбаному обладнанні із спеціалізованими прискорювачами типу GPU для використання в локальному та хмарному середовищі, розробки та тестування платформи штучного інтелекту (ШІ-платформу), налаштовання пілотної конфігурації ШІ-платформи (з веб-демонстрацією) для широкого спектру ШІ-моделей.

В результаті виконання цих завдань було отримано наступні результати. Створено віртуальні версії компонентів ШІ-платформи для використання на придбаному обладнанні із спеціалізованими прискорювачами типу GPU для використання в локальному та хмарному середовищі (віртуальні машини та контейнери із дистанційним мережевим доступом). Створено програмне забезпечення (на основі хмарних серверів та клієнтів) для розробки та практичного використання DL-моделей, для зберігання та розширення наборів даних (JSRT, ChestXray, CheXpert, ISIC, ЕЕГ, та інші) та для взаємодії із користувачами (медичними працівниками) в практичних умовах. Спроєктовано ШІ-платформу (інтегрована апаратна та програмна інфраструктура) на базі придбаного, налаштованого та протестованого апаратного забезпечення із розробленим програмним забезпеченням медичного призначення (включно із встановленими та адаптованими для використання науково обґрунтованими DL-моделями). Створено прототип ШІ-платформи (https://med.comsys.kpi.ua), тобто пілотний варіант конфігурації ШІ-платформи (серверна сторона та веб-інтерфейс для медичного працівника). Зроблено висновки щодо техніко-економічного обґрунтування подальшої імплементації та масштабування ШІ-платформи для використання в CADe та CADx в практичних умовах.

ВИСНОВКИ

У результаті виконання завдань проєкту було отримано низку запланованих (відповідно до сформульованих та запланованих технічних завдань і мети проєкту) та нових результатів. Побудовано технічну та методичну базу для подальшого створення і використання нових методів ШІ в практичних цілях широкого медичного застосування, а саме методів машинного навчання (machine learning – ML) та методів поглибленого навчання (deep learning – DL) для автоматизованого аналізу медичних даних різного типу і розмірності, включаючи одновимірні акустичні, двовимірні рентгенографічні, тривимірні комп'ютерної томографії (КТ/МРТ), багатоканальної електроенцефалографії (ЕЕГ), тощо.

Налаштовано конфігурацію придбаного обладнання із спеціалізованими прискорювачами типу GPU, яку налаштовано для використання в локальному та хмарному середовищі для потреб розробленої ШІ-платформи.

Спроєктовані та імплементовані віртуальні версії компонентів ШІ-платформи для використання на придбаному обладнанні із спеціалізованими прискорювачами типу GPU для використання в локальному та хмарному середовищі (віртуальні машини та контейнери із дистанційним мережевим доступом).

Розроблено програмне забезпечення (на основі хмарних серверів та клієнтів) для розробки та практичного використання DL-моделей, для зберігання та розширення наборів даних (JSRT, ChestXray, CheXpert, ISIC, ЕЕГ, та інші) та для взаємодії із користувачами (медичними працівниками) в практичних умовах.

Встановлені та адаптовані для використання набори закордонних та національних медичних даних (JSRT, ChestXray, CheXpert, CXR COVID-19, ISIC, ЕЕГ, та інші). Розроблено та протестовано науково обґрунтовані DL-моделі (на базі архітектур DenseNet, ResNet, InceptionResNet, Inception, Xception, EfficientNet, VGG та їхніх модифікованих версій та ансамблів), які були та адаптовані для використання в якості ШІ-асистентів лікаря на базі розробленої ШІ-платформи.

Проведено тести обладнання та програмного забезпечення для компонентів ШІ-платформи та ШІ-платформи в цілому для застосування у практичних умовах. Налаштовано конфігурацію придбаного обладнання (модулі для пери-

ферійних обчислень (включно із Google Coral, Intel Movidius, тощо) та розробленого програмного забезпечення для використання в польових умовах.

Спроєктовано ШІ-платформу (інтегрована апаратна та програмна інфраструктура) на базі придбаного, налаштованого та протестованого апаратного забезпечення із розробленим програмним забезпеченням медичного призначення (включно із встановленими та адаптованими для використання науково обґрунтованими DL-моделями).

Розроблено та імплементовано прототип ШІ-платформи (https://med.comsys.kpi.ua), тобто пілотний варіант конфігурації ШІ-платформи (серверна сторона та веб-інтерфейс для медичного працівника).

На основі результати тестів ШІ-платформи, модулю для периферійних обчислень і врахування зворотного зв'язку від медичних працівників зроблено висновки щодо техніко-економічного обґрунтування подальшої імплементації та масштабування ШІ-платформи для використання в CADe та CADx в практичних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. Shiraishi, S. Katsuragawa, J. Ikezoe, T. Matsumoto, T. Kobayashi, K. Komatsu, M. Matsui, H. Fujita, Y. Kodera, and K. Doi, "Development of a digital image database for chest radiographs with and without a lung nodule: Receiver operating characteristic analysis of radiologists' detection of pulmonary nodules", *American Journal of Roentgenology*, vol. 174, no. 1, pp. 71–74, 2000.
2. S. G. Armato III, G. McLennan, L. Bidaut, M. F. McNitt-Gray, C. R. Meyer, A. P. Reeves, B. Zhao, D. R. Aberle, C. I. Henschke, E. A. Hoffman, *et al.*, "The lung image database consortium (lidc) and image database resource initiative (idri): A completed reference database of lung nodules on ct scans", *Medical physics*, vol. 38, no. 2, pp. 915–931, 2011.
3. S. Jaeger, S. Candemir, S. Antani, Y.-X. J. Wáng, P.-X. Lu, and G. Thoma, "Two public chest x-ray datasets for computer-aided screening of pulmonary diseases", *Quantitative imaging in medicine and surgery*, vol. 4, no. 6, p. 475, 2014.
4. S. Jaeger, A. Karargyris, S. Candemir, L. Folio, J. Siegelman, F. Callaghan, Z. Xue, K. Palaniappan, R. K. Singh, S. Antani, *et al.*, "Automatic tuberculosis screening using chest radiographs", *IEEE transactions on medical imaging*, vol. 33, no. 2, pp. 233–245, 2013.
5. X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. Summers, "Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases", in *IEEE CVPR*, vol. 7, 2017.
6. J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, B. Haghgoo, R. Ball, K. Shpanskaya, *et al.*, "Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 590–597.
7. J. Yanase and E. Triantaphyllou, "A systematic survey of computer-aided diagnosis in medicine: Past and present developments", *Expert Systems with Applications*, vol. 138, p. 112 821, 2019. DOI: 10.1016/j.eswa.2019.112821.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419305238.

8. *Latest global cancer data: Cancer burden rises to 19.3 million new cases and 10.0 million cancer deaths in 2020*, World Health Organization, 2020. [Online]. Available: https://www.iarc.who.int/wp-content/uploads/2020/12/pr292_E.pdf.

9. Y. Gordienko, P. Gang, J. Hui, W. Zeng, Y. Kochura, O. Alienin, O. Rokovyi, and S. Stirenko, "Deep learning with lung segmentation and bone shadow exclusion techniques for chest x-ray analysis of lung cancer", in *International Conference on Computer Science, Engineering and Education Applications*, Springer, 2018, pp. 638–647.

10. S. Stirenko, Y. Kochura, O. Alienin, O. Rokovyi, Y. Gordienko, P. Gang, and W. Zeng, "Chest x-ray analysis of tuberculosis by deep learning with segmentation and augmentation", in *2018 IEEE 38th International Conference on Electronics and Nanotechnology (ELNANO)*, IEEE, 2018, pp. 422–428.

11. P. Gang, W. Zhen, W. Zeng, Y. Gordienko, Y. Kochura, O. Alienin, O. Rokovyi, and S. Stirenko, "Dimensionality reduction in deep learning for chest x-ray analysis of lung cancer", in *2018 tenth international conference on advanced computational intelligence (ICACI)*, IEEE, 2018, pp. 878–883.

12. P. Gang, W. Zeng, Y. Gordienko, Y. Kochura, O. Alienin, O. Rokovyi, and S. Stirenko, "Effect of data augmentation and lung mask segmentation for automated chest radiograph interpretation of some lung diseases", in *International Conference on Neural Information Processing*, Springer, 2019, pp. 333–340.

13. C. J. Pannucci and E. G. Wilkins, "Identifying and avoiding bias in research", *Plastic and reconstructive surgery*, vol. 126, no. 2, p. 619, 2010.

14. A. Ng, *AI is the new electricity*. O'Reilly Media, 2018.

15. M. Yetisgen-Yildiz, M. L. Gunn, F. Xia, and T. H. Payne, "A text processing pipeline to extract recommendations from radiology reports", *Journal of biomedical informatics*, vol. 46, no. 2, pp. 354–362, 2013.

16. A. Ng, *AI Transformation Playbook*. Landing AI, 2020. [Online]. Available: https://landing.ai/wp-content/uploads/2020/05/LandingAI_Transformation_Playbook_11-19.pdf.

17. A. Ng, *Deep learning specialization*, Coursera, 2019. [Online]. Available: https://www.coursera.org.

18. Y. Bengio, "Deep learning of representations: Looking forward", in *International conference on statistical language and speech processing*, Springer, 2013, pp. 1–37.

19. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

20. J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural networks*, vol. 61, pp. 85–117, 2015.

21. R. Buyya and S. N. Srirama, *Fog and edge computing: principles and paradigms*. John Wiley & Sons, 2019.

22. Y. Wang, "Cloud-dew architecture", *International Journal of Cloud Computing*, vol. 4, no. 3, pp. 199–210, 2015.

23. K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing", *Open Journal of Cloud Computing (OJCC)*, vol. 2, no. 1, pp. 16–24, 2015.

24. Y. Gordienko, S. Stirenko, O. Alienin, K. Skala, Z. Sojat, A. Rojbi, J. L. Benito, E. A. González, U. Lushchyk, L. Sajn, *et al.*, "Augmented coaching ecosystem for non-obtrusive adaptive personalized elderly care on the basis of cloud-fog-dew computing paradigm", in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, 2017, pp. 359–364.

25. F. Al-Turjman and F. Al-Turjman, *Edge Computing*. Springer, 2019.

26. K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research", *IEEE access*, vol. 8, pp. 85 714–85 728, 2020.

27. G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future", *arXiv preprint arXiv:1602.04283*, 2016.

28. E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?", in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 5–14.

29. T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning", *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

30. F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip", *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

31. P. Ienne *et al.*, "Architectures for neuro-computers: Review and performance evaluation", *Computer Science Department Technical Report No. 93*, vol. 21, 1993.

32. *Programming tensor cores in cuda 9*, NVIDIA Corporation. [Online]. Available: https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/.

33. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit", in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

34. M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource characterisation of personal-scale sensing models on edge accelerators", in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, 2019, pp. 49–55.

35. *Ai iot repository*, NVIDIA Corporation. [Online]. Available: https://github.com/NVIDIA-AI-IOT.

36. *Open source projects for coral.ai*, Google. [Online]. Available: https://github.com/google-coral.

37. *Open visual inference and neural network optimization*, The OpenVINO™. [Online]. Available: https://github.com/intel/ros_openvino_toolkit.

38. Horned Sungem. [Online]. Available: https://github.com/HornedSungem.

39. *Develop and prototype on the latest arm technology*, 6Boards. [Online]. Available: https://github.com/96boards/documentation.

40. J. Chen and X. Ran, "Deep learning with edge computing: A review.", *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

41. B. Varghese, N. Wang, D. Bermbach, C.-H. Hong, E. de Lara, W. Shi, and C. Stewart, "A survey on edge performance benchmarking", *arXiv preprint arXiv:2004.11725*, 2020.

42. C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cerin, and J. Wan, "Energy aware edge computing: A survey", *Computer Communications*, vol. 151, pp. 556–580, 2020.

43. E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing", *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

44. A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey", *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

45. C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, "Fog computing for the internet of things: A survey", *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, pp. 1–41, 2019.

46. W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds", in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1270–1278.

47. M. Shulha, Y. Gordienko, and S. Stirenko, "Comparative analysis of the datasets with multimodal content", in *6th International Conference High Performance Computing*, Kyiv, Ukraine, Nov. 2020.

48. R. Statkevych, Y. Gordienko, and S. Stirenko, "Possible improvements of automotive pedestrian detection methods", in *The First International Conference on Artificial Intelligence and Logistics Engineering (ICAILE-2021)*, Kyiv, Ukraine, Jan. 2021.

49. A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare", *Nature medicine*, vol. 25, no. 1, pp. 24–29, 2019.

50. W. Cao, J. Su, Z. Peng, W. Xu, and Q. Liu, "Electronic medical record of university hospital based on deep learning", in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 569, 2019, p. 052 110.

51. H. Rathore, A. Mohamed, and M. Guizani, "Blockchain applications for healthcare", in *Energy Efficiency of Medical Devices and Healthcare Applications*, Elsevier, 2020, pp. 153–166.

52. V. Valko, Y. Gordienko, O. Alienin, O. Rokovyi, S. Stirenko, G. Fedak, and O. Lodygensky, "Blockchain and smart contracts for provenance of deep learning content in healthcare", in *6rd International Conference of High Performance Computing (HPC-UA)*, Kyiv, Ukraine, Nov. 2020.

53. M. Negassi, R. Suarez-Ibarrola, S. Hein, A. Miernik, and A. Reiterer, "Application of artificial neural networks for automated analysis of cystoscopic images: A review of the current status and future prospects", *World Journal of Urology*, pp. 1–10, 2020.

54. Q. Hu, L. F. d. F. Souza, G. B. Holanda, S. S. Alves, F. H. d. S. Silva, T. Han, and P. P. Reboucas Filho, "An effective approach for ct lung segmentation using mask region-based convolutional neural networks", *Artificial intelligence in medicine*, vol. 103, p. 101 792, 2020.

55. R. Sundhari, "Enhanced histogram equalization based nodule enhancement and neural network based detection for chest x-ray radiographs", *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 3, pp. 3831–3839, 2021.

56. E. Tasci, "Pre-processing effects of the tuberculosis chest x-ray images on pre-trained cnns: An investigation", in *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering*, Springer, 2019, pp. 589–596.

57. D. Verma, C. Bose, N. Tufchi, K. Pant, V. Tripathi, and A. Thapliyal, "An efficient framework for identification of tuberculosis and pneumonia in chest x-ray images using neural network", *Procedia Computer Science*, vol. 171, pp. 217–224, 2020.

58. P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning", *arXiv preprint arXiv:1711.05225*, 2017.

59. L. Wynants, B. Van Calster, G. S. Collins, R. D. Riley, G. Heinze, E. Schuit, M. M. Bonten, D. L. Dahly, J. A. Damen, T. P. Debray, *et al.*, "Prediction models for diagnosis and prognosis of covid-19: Systematic review and critical appraisal", *bmj*, vol. 369, 2020.

60. S. L. Halson, "Monitoring training load to understand fatigue in athletes", *Sports medicine*, vol. 44, no. 2, pp. 139–147, 2014.

61. N. Singh, K. J. Moneghetti, J. W. Christle, D. Hadley, V. Froelicher, and D. Plews, "Heart rate variability: An old metric with new meaning in the era of using mhealth technologies for health and exercise training guidance. part two: Prognosis and training", *Arrhythmia & electrophysiology review*, vol. 7, no. 4, p. 247, 2018.

62. S. Stirenko, P. Gang, W. Zeng, Y. Gordienko, O. Alienin, O. Rokovyi, N. Gordienko, I. Pavliuchenko, and A. Rojbi, "Parallel statistical and machine learning methods for estimation of physical load", in *International Conference on Algorithms and Architectures for Parallel Processing*, Springer, 2018, pp. 483–497.

63. Y. Gordienko, S. Stirenko, Y. Kochura, O. Alienin, M. Novotarskiy, and N. Gordienko, "Deep learning for fatigue estimation on the basis of multimodal human-machine interactions", *arXiv preprint arXiv:1801.06048*, 2017.

64. V. Vesterinen, K. Häkkinen, E. Hynynen, J. Mikkola, L. Hokka, and A. Nummela, "Heart rate variability in prediction of individual adaptation to endurance training in recreational endurance runners", *Scandinavian journal of medicine & science in sports*, vol. 23, no. 2, pp. 171–180, 2013.

65. P. Gang, J. Hui, S. Stirenko, Y. Gordienko, T. Shemsedinov, O. Alienin, Y. Kochura, N. Gordienko, A. Rojbi, J. L. Benito, *et al.*, "User-driven intelligent interface on the basis of multimodal augmented reality and brain-computer interaction for people with functional disabilities", in *Future of Information and Communication Conference*, Springer, 2018, pp. 612–631.

66. N. Gordienko, O. Lodygensky, G. Fedak, and Y. Gordienko, "Synergy of volunteer measurements and volunteer computing for effective data collecting, processing, simulating and analyzing on a worldwide scale", in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, 2015, pp. 193–198.

67. Y. Gordienko, Y. Kochura, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, and S. Stirenko, "Scaling analysis of specialized tensor processing architectures for deep learning models", in *Deep Learning: Concepts and Architectures*, Springer, 2020, pp. 65–99.

68. J. Russell, *Riseml benchmarks google tpuv2 against nvidia v100 gpu*, HPC wire. [Online]. Available: https://www.hpcwire.com/2018/04/30/riseml-benchmarks-google-tpuv2-against-nvidia-v100-gpu/.

69. Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, and S. Stirenko, "Batch size influence on performance of graphic and tensor processing units during training and inference phases", in *International Conference on Computer Science, Engineering and Education Applications*, Springer, 2019, pp. 658–668.

70. V. Rotemberg, N. Kurtansky, B. Betz-Stablein, L. Caffery, E. Chousakos, N. Codella, M. Combalia, S. Dusza, P. Guitera, D. Gutman, *et al.*, "A patient-centric dataset of images and metadata for identifying melanomas using clinical context", *Scientific data*, vol. 8, no. 1, pp. 1–8, 2021.

71. Y.-W. Chen and L. C. Jain, *Deep learning in healthcare*, 2020.

72. P. Gang, W. Zeng, Y. Gordienko, O. Rokovyi, O. Alienin, and S. Stirenko, "Prediction of physical load level by machine learning analysis of heart activity after exercises", in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2019, pp. 557–562.

73. *Covid-19 chest x-ray segmentations dataset, covid-19 detection with heat-map visualization*, https://github.com/ieee8023/covid-chestxray-dataset, https://www.kaggle.com/basu369victor/covid-19-detection-with-heat-map-visualization, 2021.

74. A. Prasolov, S. Stirenko, and Y. Gordienko, "Improvement of image super resolution by deep neural networks", in *IEEE EUROCON 2021-19th International Conference on Smart Technologies*, IEEE, 2021, pp. 140–145. DOI: 10.1109/EUROCON52738.2021.9535575.

75. D. Vasylenko, S. Stirenko, and Y. Gordienko, "Improvement of image compression performance by deep neural networks", in *IEEE EUROCON 2021-19th International Conference on Smart Technologies*, IEEE, 2021, pp. 135–139. DOI: 10.1109/EUROCON52738.2021.9535628.

76. R. Statkevych, S. Stirenko, and Y. Gordienko, "Human kidney tissue image segmentation by u-net models", in *IEEE EUROCON 2021-19th International Conference on Smart Technologies*, IEEE, 2021, pp. 129–134. DOI: 10.1109/EUROCON52738.2021.9535599.

77. R. Statkevych, Y. Gordienko, and S. Stirenko, "Improving pedestrian detection methods by architecture and hyperparameter modification of deep neural networks", in *The International Conference on Artificial Intelligence and Logistics Engineering*, Springer, 2021, pp. 44–53. DOI: 10.1007/978-3-030-80475-6_5.

78. Y. Gordienko, K. Kostiukevych, N. Gordienko, O. Rokovyi, O. Alienin, and S. Stirenko, "Deep learning for grasp-and-lift movement forecasting based on electroencephalography by brain-computer interface", in *The International Conference on Artificial Intelligence and Logistics Engineering*, Springer, 2021, pp. 3–12. DOI: 10.1007/978-3-030-80475-6_1.

79. V. Doms, Y. Gordienko, Y. Kochura, O. Rokovyi, O. Alienin, and S. Stirenko, "Deep learning for melanoma detection with testing time data augmentation", in *The International Conference on Artificial Intelligence and Logistics Engineering*, Springer, 2021, pp. 131–140. DOI: 10.1007/978-3-030-80475-6_13.

80. V. Valko, S. Stirenko, I. Babarykin, and Y. Gordienko, "Provenance detection system for deep learning content in healthcare", in *IEEE EUROCON 2021-19th International Conference on Smart Technologies*, IEEE, 2021, pp. 146–151. DOI: 10.1109/EUROCON52738.2021.9535621.

81. G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

82. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

83. Y. Gordienko, Y. Kochura, V. Taran, N. Gordienko, O. Rokovyi, O. Alienin, and S. Stirenko, "Last mile" optimization of edge computing ecosystem with deep learning models and specialized tensor processing architectures", in *Advances in Computers*, vol. 122, Elsevier, 2021, pp. 303–341. DOI: 10.1016/bs.adcom.2020.10.003.

84. V. Taran, Y. Gordienko, O. Alienin, O. Rokovyi, Y. Kochura та S. Stirenko, "Edge Intelligence for Medical Applications under Field Conditions", в *The Second International Conference on Artificial Intelligence and Logistics Engineering*, рукопис передано на рецензування, Springer, Kyiv, Ukraine, 2021.

85. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

86. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning", in *Thirty-first AAAI conference on artificial intelligence*, 2017.

87. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

88. K. Simonyan та A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

89. F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” в *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, с. 1251—1258.

90. M. Tan та Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” в *International Conference on Machine Learning*, PMLR, 2019, с. 6105—6114.

СПИСОК РИСУНКІВ

1.1 Схема взаємодії TPU з програмним середовищем 28
1.2 Приклади апаратних засобів, використаних у роботі: Horned Sungem а) та Google Coral б) 32

2.1 Імпорт бібліотек на основі оточення Tensorflow/Keras 37
2.2 Перевірка наявної інфраструктури 37
2.3 Перевірка контенту зібраних даних 38
2.4 Перевірка кількості навчальних знімків 38
2.5 Перевірка контенту зібраних даних 39
2.6 Підготовка даних до навчання 39
2.7 Підготовка даних до візуалізації результатів 40
2.8 Додавання відступу по краях зображення 41
2.9 Завантаження тренувальної підмножини датасету 41
2.10 Обираємо зображення для візуалізації 42
2.11 Приклади візуалізації до і після обробки 42
2.12 Адаптація швидкості навчання моделі 43
2.13 Ініціалізація моделі 43
2.14 Тренування мережі та вивід результатів після кожної епохи . . . 44
2.15 Підготовки файлу для збереження прогнозів 44
2.16 Тестування DL-моделі 45
2.17 Прогнози моделі на тестових даних 45
2.18 Представлення прогнозів моделі на тестових даних 46
2.19 Обираємо декілька тестових зображень для візуальної перевірки прогнозів моделі 46
2.20 Візуалізація результатів тестування моделі глибинного навчання (DL) 47
2.21 Веб-сторінка навчального майданчику на платформі Kaggle із датасетом, кодами та результатами їх виконання 48
2.22 Імпортування бібліотек на основі оточення Torch 49
2.23 Встановлення параметрів 50
2.24 Функція для виконання операцій командного рядка 50
2.25 Скачування оригінального датасету 50

2.26 Організація шляхів до даних локального сховища 51
2.27 Підвантаження набору даних 51
2.28 Зчитування та запис метаінформації до змінної `meta` 51
2.29 Проєкції у яких подані рентгенограми 51
2.30 Вивід метаінформації про наявні типи хвороб у наборі даних . . . 52
2.31 Вибір для подальшого аналізу рентгенограми у проєкції PA . . . 52
2.32 Аналіз кількості рентгенограм по кожному класу 53
2.33 Балансування рентгенограм по COVID-19 53
2.34 Набір даних після балансування 54
2.35 Отримані маркери унаслідок унітарного кодування 55
2.36 Оригінальні маркери 55
2.37 Унітарні та оригінальні маркери 56
2.38 Візуальне подання рентгенограм та класів, яким вони відповідають 58
2.39 Визначення DenseNet121 59
2.40 Фрагмент виводу результатів навчання моделі після першої (Epoch: 0) та останньої епохи навчання (Epoch: 39) 62
2.41 Візуалізація теплових мап для перших 25 (з 50) тестових зображень 68
2.42 Візуалізація теплових мап для наступних 25 (з 50) тестових зображень 69
2.43 Тестування з “холодного” старту оточення та апаратного забезпечення із виводом результатів на Titan1080 74
2.44 Тестування з “розігрітого” старту оточення та апаратного забезпечення із виводом результатів на Titan1080 75
2.45 Графік залежності часу тестування одного знімку для всієї послідовності знімків (“холодний” + “розігрітий” старт) на Titan1080 76
2.46 Графік залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту” на Titan1080 76
2.47 Результатами візуалізації теплових мап на Titan1080 77
2.48 Графік залежності часу тестування одного знімку для різних значень розміру міні-пакету на Titan1080 79

2.49 Конфігурація нового апаратного забезпечення (NVIDIA GPU A100) 80
2.50 Тестування з “холодного” старту оточення та апаратного забезпечення із виводом результатів на A100 81
2.51 Тестування з “розігрітого” старту оточення та апаратного забезпечення із виводом результатів на A100 81
2.52 Графік залежності часу тестування одного знімку для всієї послідовності знімків (“холодний” + “розігрітий” старт) на A100 . 82
2.53 Графік залежності часу тестування для послідовності знімків без першого знімку, тестування якого відбувалося з “холодного старту” на A100 83
2.54 Результатами візуалізації теплових мап на A100 84
2.55 Графік залежності часу тестування одного знімку для різних значень розміру міні-пакету на A100 86
2.56 Приклад виводу інформації на тестовому майданчику про ймовірність наявності аномалій, виявлених на рентгенограмах легень, та побудовою теплових мап цих аномалій 87
2.57 Модель методу компресії 93
2.58 Результати тренування 95
2.59 Архітектура моделі 4 stage mU-Net 97
2.60 Метрики для оцінки продуктивності моделей: 3-stage mU-Net, 4-stage mU-Net та U-Net 98
2.61 Спрощена “ванільна” мережа LSTM на основі класичної архітектури LSTM та компонентів повністю зв’язаної нейронної мережі (FCN) 101
2.62 Гібридна мережа CNN-LSTM на основі компонентів CNN та додаванням блоку LSTM класичної архітектури LSTM та компонентів повністю зв’язаної нейронної мережі (FCN) 102
2.63 Графіки залежності метрики AUC як функція від кількості зразків у вхідній послідовності даних (N) зі значенням зміщення: 0.5 для а) мережа FCN, б) гібридна мережа FCN-CNN, в) гібридна мережа CNN-LSTM 103
2.64 Загальна структура пропонованої системи 110
2.65 Загальна структура пропонованої системи 111

2.66 Можливі варіанти еволюції нових версій набору даних та моделі . 113
2.67 Еволюція контенту машинного навчання моделі(червоні стрілки) та набору даних(зелені стрілки) 115

3.1 Головна сторінка ШІ-платформи 125
3.2 Загальний опис ШІ-платформи 126
3.3 Правила роботи із ШІ-платформою 126
3.4 Опис ШІ-асистентів лікаря 127
3.5 Опис ШІ-послуг 128
3.6 Перелік наших публікацій 129
3.7 Колектив розробників ШІ-платформи 129
3.8 Форма зворотного зв'язку із колективом розробників ШІ-платформи 130
3.9 Вхід у систему за логіном та паролем 130
3.10 Стан ШІ-платформи після входу в систему доктора Криворучка О.А. (демонстраційний акаунт) 131
3.11 Стан ШІ-платформи після створення дослідження типу `Рентген` (демонстраційний акаунт) 131
3.12 Стан ШІ-платформи після вибору файлу 132
3.13 Перелік виконаних заявок для дослідження з номером № 112 . . . 133
3.14 Стан ШІ-платформи після вибору заявки за номером (№ 412) . . 134
3.15 Ймовірності аномалій для класифікації всього знімку по ознаках 14 хвороб легенів 135
3.16 Ймовірність аномалії для класифікації всього знімку по ознаках COVID-2019 135
3.17 Файл-зображення з ймовірними зонами розташування ознак аномальності (heatmap) 135
3.18 Ймовірності аномалій для класифікації всього знімку для ідентифікації зон можливих аномалій і класифікації аномалій в даних зонах по ознаках 14 хвороб легенів 136
3.19 Файл-зображення з зонами (bounidng boxes) ймовірних аномалій, назвами аномалій та ймовірностями їх присутності . . . 136
3.20 Компоненти ШІ-платформи 143

3.21 Порядок проходження заявок на обробку з черги від глобального планувальника до локального, та надходження до контейнерів для обробки 145
3.22 Фіксований розподіл ресурсів 149
3.23 Статичний розподіл ресурсів 150
3.24 Динамічний розподіл ресурсів 151

СПИСОК ТАБЛИЦЬ

1.1 Список основного апаратного забезпечення TPU 28
1.2 Параметри апаратних засобів, використаних у роботі (рис. 1.2) . . 32

2.1 Порівняння основних параметрів у процесі навчання мереж . . . 91
2.2 Метрики для найкращих епох навчання моделей 99
2.3 Величини показника якості класифікації та середнього квадратичного відхилення кожного типу архітектури моделі з урахуванням test-time augmentation 105
2.4 Показники часу тестування та середнього квадратичного відхилення для моделей, що були використані з урахуванням test-time augmentation 106

3.1 Порівняння основних параметрів у процесі навчання мереж . . . 147