МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
19 травня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія глибокого машинного навчання системи розпізнавання рентгенівських зображень»
здобувача групи ІН.м-11н. Видригана Владислава Олеговича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Владислав ВИДРИГАН
(підпис)

Керівник,
ст. викл., к.ф.-м.н. Оксана ШОВКОПЛЯС _____
(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»

здобувача групи ІН.м-11н Видригана Владислава Олеговича

1. Тема роботи: «Інформаційна технологія глибокого машинного навчання системи розпізнавання рентгенівських зображень»

затверджую наказом по СумДУ від «08» травня 2023 р. № _____

2. Термін здачі здобувачем кваліфікаційної роботи до 19 травня 2023 року

3. Вхідні дані до кваліфікаційної роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Літературний огляд 2)Огляд існуючих рішень 3) Постановка задачі 4) Вибір методів реалізації 5) Розробка моделі пригнічення кісткової тканини 6) Розробка моделі розпізнання рентгенівських зображень 7) Розроблення чат-бота Telegram 8) Висновок .*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Огляд літератури*		
2	*Постановка задачі та формування завдань дослідження*		
3	*Опис методів реалізації*		
4	*Розробка моделей та чат-бота Telegram*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 89 стор., 69 рис., 1 додаток, 23 джерело.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки розкриває проблему розпізнавання рентгенівських зображень шляхом розробки відповідної інформаційної технології.

Об'єкт дослідження – процес розпізнавання рентгенівських зображень.

Мета роботи – розроблення ефективної системи розпізнавання рентгенівських зображень з використанням методів глибокого машинного навчання для автоматичного аналізу та класифікації патологій.

Методи дослідження – алгоритми глибокого машинного навчання, додаткові застосовані техніки попередньої обробки даних та оцінки ефективності моделей.

Результати – проведений аналіз літературних джерел, методів виконання та інструментів, які дозволяють навчати моделі здатних розпізнавати патології на рентген з високою точністю та на основі цього створено чат-бот месенджеру Telegram для розповсюдження та перевірки цих моделей на більшому наборі даних.

АВТОКОДУВАЛЬНИК, ГЛИБИННЕ НАВЧАННЯ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, КЛАСИФІКАЦІЯ, НЕЙРОННА МЕРЕЖА, РЕНТГЕН, РОЗПІЗНАВАННЯ, PYTHON, TELEGRAM

ЗМІСТ

ВСТУП

Актуальність.

Традиційне тлумачення рентгенівських зображень залежить від знань та досвіду радіологів. Однак цей процес може зайняти багато часу та бути схильним до людських помилок. Поява глибокого навчання в охороні здоров’я може забезпечити багатообіцяюче рішення шляхом розробки систем, здатних автоматизувати аналіз та інтерпретацію рентгенівських зображень.

Об’єкт дослідження. Процес розпізнавання рентгенівських зображень.

Предмет дослідження. Методологія розпізнавання рентгенівських зображень.

Гіпотеза. Використовуючи технології глибокого машинного навчання, система може автоматично аналізувати рентгенівські зображення з високою точністю.

Наукова новизна. Описана у даній роботі модель відрізняється від існуючих інформаційних рішень і традиційних методів діагностики своєю здатністю автоматично обробляти великі обсяги вхідних даних. Порівняно з лікарем, який розглядає результати лише одного пацієнта, ця система може обробляти дані багато разів більше за той самий проміжок часу. В результаті цього вона забезпечує своєчасне виявлення хвороб, що зменшує потенційний ризик ускладнень та смертності.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (ІМА – 2023).

Структура. Дана робота складається зі вступу, аналізу предметної області, постановки задачі дослідження, вибір мови та інструментів для рішення поставленої проблеми, опису розробки програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Літературний огляд

Рентгенографія

Рентгенографія – метод перегляду внутрішньої структури тіла людини, який відіграє вирішальну роль у діагностиці та лікуванні широкого спектру захворювань. Однак, незважаючи на прогрес у науці, все ще існують проблеми та помилки під час аналізу. Помилки виникають через низку причин, пов’язаних із помилками людини, технічними факторами та системними збоями. Однією з загальноприйнятих систем класифікації для визначення цих проблем є класифікація Ренфрю [1].

Система класифікації Ренфрю визначає 12 різних типів помилок, які можуть виникнути в діагностичній радіології. До них належать неправильне міркування, відсутність знань, недостатнє тлумачення, погане спілкування, технічні помилки, відсутність перегляду попередніх результатів візуалізації, неповна клінічна інформація, відсутність аномалій за межами області інтересу, нездатність знайти наступні аномалії, ускладнення втручання і надмірна залежність від попередніх звітів.

Інша система класифікації, запропонована Brook et al [2]. висвітлює різні типи помилок, які можуть виникнути в діагностичній радіології, включаючи приховані помилки, активні збої або людські помилки, ускладнення процедур, зовнішні причини та причини, пов’язані з клієнтом. Ця система класифікації враховує не лише людські помилки, а й інші фактори, які можуть сприяти діагностичним помилкам.

Деякі поширені помилки, які трапляються в діагностичній радіології, включають неправильну інтерпретацію зображень, неправильний діагноз, неправильне положення пацієнта. Ці помилки можуть призвести до затримки або неправильного лікування, непотрібних тестів або процедур і шкоди пацієнту.

Дослідження, проведене Лондонським коледжем радіології [3], включало участь 138 лікарів, які аналізували рентгенівські знімки для діагностики

захворювань легень. Результати дослідження показали, що загальна точність діагностики становила 81%.

З урахуванням складності та вимогливості процесу діагностики, систем розпізнавання рентгенівських зображень може стати цінним інструментом для поліпшення точності та ефективності діагностичної радіології. Застосування цієї технології дозволяє автоматизувати процес аналізу та інтерпретації зображень, а також зменшити ймовірність помилок, пов'язаних з людським фактором.

Попереднє опрацювання рентгенологічних зображень

Виявлення аномалій може стати важким через те, що вони приховані наявністю кісткових структур, таких як ребра та ключиці, що призводить до неправильної інтерпретації. Створення моделі пригнічення кісткової тканини на основі глибокого навчання може допомогти зменшити помилки в радіологічній інтерпретації.

Рентгенограма грудної клітки є широко використовуваним діагностичним інструментом для оцінки стану легень і грудної клітки. Однак інтерпретація цих зображень може бути складною через наявність перекриваючих анатомічних структур, таких як ребра та ключиці, які можуть закривати підлеглі легеневі тканини. Це може ускладнити точне виявлення та класифікацію аномалій, включаючи вузлики, пухлини та інші патології.

Щоб вирішити цю проблему було досліджено методи обробки зображень, однією з яких є пригнічення кісткової тканини [4]. Пригнічення кісткової тканини має покращує видимість легеневих структур шляхом зменшення помітності кісток на рентгенівському зображенні, тим самим покращуючи точність виявлення та класифікації аномалій.

Техніка пригнічення кісткової тканини включає два основні етапи: видалення кісткової тканини та покращення зображення. На етапі вилучення кісткової тканини використовуються алгоритми для ідентифікації та сегментації кісткових структур на оригінальному рентгенівському зображенні. Цього можна

досягти за допомогою різних методів, таких як порогове значення, розширення регіону або підходи на основі машинного навчання.

Після видалення кісткових структур створюється зображення з пригніченням кісткової тканини шляхом пригнічення або послаблення кісткових областей із збереженням легеневих структур. Для цього можна використовувати різні алгоритми та фільтри, включаючи математичні моделі та методи ітераційної оптимізації.

Завдяки пригніченню кісткових структур отримане зображення забезпечує чіткіше уявлення про легеневі тканини, що лежать під ними, що дозволяє рентгенологам точніше виявляти та класифікувати аномалії. Покращена видимість вузлів, пухлин та інших легеневих патологій може допомогти в ранній діагностиці, плануванні лікування та моніторингу захворювань легень.

Кілька досліджень показали потенціал методів пригнічення кісткової тканини для покращення класифікації рентгенівських зображень грудної клітки. Зменшуючи перешкоди для кісток, ці методи можуть покращити видимість тонких аномалій, які інакше могли б залишитися непоміченими. Це може призвести до підвищення точності діагностики таких захворювань, як рак легень, пневмонія та інші легеневі захворювання[5].

Переваги впровадження чат-бота Telegram

У цифровому світі з’явилися нові технології, які змінюють способи взаємодії з інформацією та оточуючим світом. Останніми роками чат-боти набули значної популярності та виявилися кращим вибором порівняно з веб-сайтами чи додатками в різних аспектах.

Чат-боти відкрили безліч можливостей у різних галузях, таких як комунікація, підтримка клієнтів, торгівля, навчання та інше. Однією з популярних платформ для розробки та розповсюдження чат-ботів є Telegram.

Основна перевага Telegram полягає в його доступності на різних пристроях, включаючи комп’ютери, смартфони і планшети. Користувачі можуть отримати доступ до системи розпізнавання рентгенівських зображень з будь-якого

пристрою, включаючи веб-версію додатка. Це дозволяє їм отримувати та аналізувати рентгенівські зображення прямо через браузер на комп'ютері або планшеті, не обмежуючись використанням лише смартфона. Такий підхід забезпечує зручність і гнучкість в роботі з системою розпізнавання рентгенівських зображень і дозволяє користувачам мати постійний доступ до неї незалежно від пристрою [6].

Telegram відомий своєю високою увагою до безпеки та приватністю своїх користувачів. Цей месенджер забезпечує широкий спектр захисних заходів, що гарантують конфіденційність комунікації. Він використовує шифрування end-to-end, тому лише відправник і отримувач мають доступ до змісту повідомлень, навіть служба підтримки не може їх переглядати.

Порівняно зі створенням окремого веб-сайта або мобільного додатка, використання Telegram має свої переваги в швидкому запуску і зменшенні витрат на розробку. Завдяки API [7] можна легко створити бота, що дозволяє швидко розробити і впровадити певні системи.

Telegram має значні можливості для розширення. Можна легко впровадити додаткові модулі та функціонал, наприклад, використовувати нейронні мережі для покращення візуальної якості зображень або вдосконалювати алгоритми розпізнавання за допомогою нових навчальних даних Це дозволить покращити точність та потужність системи розпізнавання рентгенівських зображень.

1.2 Огляд наявних рішень

Класифікація зображень [8] є ключовим аспектом аналізу цифрових зображень і комп’ютерного зору. Він передбачає класифікацію пікселів у цифровому зображенні на заздалегідь визначені класи. Як правило, мультиспектральні дані використовуються для виконання цієї класифікації шляхом аналізу спектральних моделей, присутніх у даних. Мета полягає в тому, щоб призначити унікальний рівень сірого або колір для представлення особливостей на зображенні та відповідних їм об’єктів реального світу. Автоматизація процесу класифікації зображень за допомогою комп’ютерного

бачення є дуже цінною, особливо коли ви маєте справу з великою кількістю зображень. Ця автоматизація знаходить застосування в різних сферах, таких як автономне водіння, автоматизована організація зображень, стокові фотографії та відео-сайти, візуальний пошук, а також розпізнавання зображень і облич у соціальних мережах.

Контрольована класифікація передбачає вибір репрезентативних зразків пікселів на зображенні та використання їх як еталонних для класифікації всіх інших пікселів на зображенні. Користувач визначає сайти навчання та встановлює межі подібності на основі спектральних характеристик. Кількість класів для класифікації також визначається користувачем. Статистична характеристика виконується для кожного класу, а процес класифікації порівнює відбивну здатність пікселів, щоб визначити найбільш подібний клас.

Неконтрольована класифікація передбачає програмний аналіз зображення без надання користувачем зразків класів. Комп'ютер використовує алгоритми для групування пов'язаних пікселів у класи, при цьому користувач лише вказує алгоритм і бажану кількість вихідних класів. Однак користувач повинен мати знання про класифіковану область, щоб пов'язати групування пікселів із фактичними особливостями землі. Загальні використовувані алгоритми включають кластерний аналіз, виявлення аномалій, нейронні мережі та моделі латентних змінних.

K-найближчих сусідів (KNN) – це простий алгоритм, який використовується для завдань класифікації та регресії. Він працює за принципом пошуку k найближчих навчальних прикладів у просторі ознак. Це вважається непараметричним і ледачим навчанням, оскільки воно апроксимує функцію локально та відкладає обчислення до оцінки.

Щоб класифікувати точку даних за допомогою KNN, ми обчислюємо відстань між її вектором ознак і векторами ознак навчальних прикладів. Найпоширеніший клас серед k найближчих сусідів призначається як членство в

класі для невідомої точки даних. Загальноприйнятими показниками відстані є евклідова відстань і мангеттенська відстань.

Результатом KNN є призначений клас для об’єкта. Серед сусідів проводиться множинне голосування, при цьому об’єкт призначається до класу, який найчастіше зустрічається серед k його найближчих сусідів. Якщо k встановлено в 1, об’єкт просто призначається до класу його єдиного найближчого сусіда.

Наївні алгоритми Баєса (Naive Bayes) — це група алгоритмів класифікації, які спираються на теорему Баєса. Це не єдиний алгоритм, а сімейство алгоритмів, які мають спільний принцип: припущення незалежності кожної пари ознак, що класифікуються. Наївний Баєс – це проста техніка для створення класифікаторів, які призначають мітки класів примірникам на основі їхніх значень ознак. Ці класифікатори зазвичай використовуються для завдань двійкової та багатокласової класифікації, і вони особливо популярні для класифікації тексту та виявлення спаму. Однією з переваг Наївного Баєса є його швидкість і масштабованість, що робить його придатним для великих наборів даних. Однак він має обмеження в тому, що він розглядає всі функції як непов’язані та не може дізнатися про зв’язки між ними. Незважаючи на це, Наївний Байєс може визначити важливість індивідуальних особливостей.

Випадковий ліс (Random Forest) – це популярний алгоритм навчання під наглядом, який використовується як для завдань класифікації, так і для регресії. Він створює колекцію дерев рішень, утворюючи лісоподібну структуру. Кожне дерево у випадковому лісі самостійно робить прогнози, а остаточний прогноз визначається шляхом голосування. Цей ансамблевий підхід покращує надійність моделі та зменшує переобладнання завдяки поєднанню кількох дерев. Випадковий ліс використовує такі методи, як пакетування та випадковість функцій, щоб побудувати окремі дерева, які не корельовані одне з одним. Це спільне передбачення комітету дерев часто дає більш точні результати, ніж одне дерево рішень.

Методи опорних векторів (SVM) — це алгоритми машинного навчання, які можна використовувати для завдань класифікації та регресії. Вони відомі своєю ефективністю в обробці як безперервних, так і категоріальних змінних. SVM створюють гіперплощину в багатовимірному просторі для представлення різних класів у даних. Гіперплощина генерується ітераційно, щоб мінімізувати помилки та максимізувати поділ між класами. Кінцева мета — знайти гіперплощину з максимальним запасом між найближчими точками навчальних даних кожного класу. Продуктивність SVM залежить від вибору функції ядра з популярними варіантами, включаючи лінійне, гауссівське та поліноміальне ядра.

Штучні нейронні мережі (ШНМ) – це комп’ютерні алгоритми, які імітують поведінку біологічних нейронних мереж. Вони широко використовуються для різних завдань, таких як класифікація, комп’ютерне бачення та розпізнавання мови. ШНМ складаються з взаємопов’язаних елементів обробки, званих вузлами, які функціонують подібно до нейронів у мозку. Зв’язки між вузлами, відомі як ваги, можна регулювати, щоб дозволити мережі наближено виконувати потрібну функцію.

ШНМ зазвичай мають приховані шари, які діють як детектори ознак. Ці рівні поступово розпізнають складніші шаблони в даних у міру того, як інформація протікає через мережу. Наприклад, у задачі розпізнавання обличчя перший прихований шар може виявляти лінії, другий шар об’єднує ці лінії, щоб утворити ніс, третій шар поєднує ніс з оком і так далі, доки не буде реконструйовано все обличчя. Цей ієрархічний підхід дозволяє мережі ідентифікувати дуже складні об’єкти.

Згорткові нейронні мережі (CNN) – це тип штучних нейронних мереж, спеціально розроблених для розпізнавання візуальних шаблонів безпосередньо з піксельних зображень із мінімальною попередньою обробкою [9]. Вони широко використовуються в задачах комп’ютерного зору та досягли найсучасніших результатів. CNN складаються зі згорткових шарів і шарів об’єднання, які прості, але можуть бути організовані різними способами для різних проблем

комп’ютерного зору. Ключова перевага CNN полягає в тому, що вони автоматично навчаються виділяти ознаки із зображень, усуваючи потребу в ручному виділенні функцій. Основна концепція полягає в застосуванні операцій згортки між вхідним зображенням і фільтрами для створення незмінних функцій, які потім передаються на наступні шари. Цей процес повторюється з різними фільтрами для створення більш абстрактних функцій, доки не буде отримано остаточну інваріантну функцію/вихід, стійку до оклюзій.

1.3 Постановка завдання

Метою даного дипломного проекту є розробка інформаційної технології глибокого машинного навчання для системи розпізнавання рентгенівських зображень.

Робота буде складатися з:

- модернізації і тренування моделі пригнічення кісткової тканини для придушення кістковою тканиною на рентгенівських зображеннях;
- модернізації і тренування моделей розпізнавання рентгенівських зображень для класифікації зображень залежно від наявності патологій;
- розроблення чат-бота Telegram, який дозволить користувачам надсилати рентгенівські зображення та отримувати результати розпізнавання.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1) виконати аналіз проблемної області;
2) провести аналіз методів класифікації зображень;
3) обрати мову та бібліотеки реалізації проекту;
4) реалізувати інформаційну системи.

Предметом дослідження є методологія розпізнавання рентгенівських зображень.

Наукова новизна полягає в тому, що розроблене програмне рішення, яке описане у цій роботі, дозволить автоматично обробляти великі обсяги вхідних даних та з більшою точністю розпізнавати рентгенівські зображення.

Практичною значимістю розроблюваної системи є поліпшення точності та ефективності розпізнання рентгенівських зображень.

2 ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Вибір мови програмування

Для написання системи розпізнавання рентгенівських зображень, було обрано мова програмування Python.

Python відомий своєю простотою та зрозумілістю, що дозволяє розробникам писати чистий та лаконічний код. Це полегшує розуміння та підтримку коду, особливо при роботі з складними алгоритмами машинного навчання.

Ще одна причина обрати Python — це його обширна екосистема бібліотек та фреймворків. Вони надають потужні інструменти для маніпулювання даними, обробки зображень, візуалізації та машинного навчання, що є важливими компонентами системи розпізнавання рентгенівських зображень.

Універсальність Python є ще одним фактором, що вплинув на вибір. Він може бути використаний для широкого спектру застосувань, від веб-розробки до наукових обчислень. Його гнучкість дозволяє легко інтегрувати з іншими технологіями та платформами. Крім того, Python має велику та активну спільноту, що означає наявність багато ресурсів, підручників та підтримки при зустрічі труднощів під час розробки [10].

Крім того, Python здобув популярність у галузі машинного навчання та науки про дані завдяки потужним фреймворкам машинного навчання, таким як TensorFlow, PyTorch та scikit-learn. Ці фреймворки пропонують високорівневі абстракції, що полегшують реалізацію складних моделей та алгоритмів. Міцна підтримка чисельних обчислень та здатність Python ефективно обробляти великі набори даних також сприяють його відповідності для цього проекту.

2.2 Вибір бібліотек

В проекті буде використано кілька бібліотек Python.

TensorFlow є одним з найпопулярніших фреймворків для глибокого навчання. У проекті використовувався TensorFlow разом з його високорівневим

інтерфейсом Keras для побудови та тренування моделей глибокого навчання для розпізнавання рентгенівських зображень [11].

Scikit-learn є потужною бібліотекою для машинного навчання та аналізу даних [12]. Вона містить реалізації різних алгоритмів класифікації, регресії, кластеризації та багато іншого. Scikit-learn використовуватиметься для попередньої обробки даних, оцінки моделей та виконання інших завдань машинного навчання, які не вимагали глибокого навчання.

OpenCV є бібліотекою комп'ютерного зору, яка надає широкі можливості для обробки зображень та комп'ютерного зору [13]. В проекті OpenCV використовуватиметься для операцій зображення, таких як завантаження, обрізка та підготовки зображень перед подачею їх на вхід до моделей глибокого навчання.

NumPy є основною бібліотекою для наукових обчислень на Python. Вона надає потужні структури даних, такі як масиви та матриці, а також функції для їх маніпуляції [14]. NumPy буде використаний для обробки та маніпуляції числових даних, які використовувалися у моделях глибокого навчання.

Matplotlib є бібліотекою для візуалізації даних у Python. Вона дозволяє побудовувати різноманітні типи графіків, діаграм та інших візуалізаційних елементів [15]. У проекті Matplotlib використовуватиметься для відображення зображень, графіків та результатів аналізу даних.

Seaborn — це ще одна бібліотека для візуалізації даних, яка побудована на основі Matplotlib. Вона надає високорівневі функції для створення привабливих та інформативних статистичних графіків [16]. Seaborn використовуватиметься для візуалізації статистичних даних та відображення результатів аналізу.

pyTelegramBotAPI — це бібліотека Python, призначена для спрощення розробки ботів Telegram за допомогою API Telegram Bot [7]. Він забезпечує простий у використанні інтерфейс для надсилання та отримання повідомлень, керування взаємодією користувачів і доступу до різноманітних методів Telegram Bot API.

2.3 Додаткові інструменти

Jupyter Notebook надає можливість виконувати код по частинах, що дозволить мені візуалізувати проміжні результати та проводити експерименти [17]. Це особливо корисно при розробці та налагодженні моделей машинного навчання, де можна швидко перевірити результати та внести зміни в код. Також є можливість вставляти текстові блоки, рисунки та графіки поруч з кодом. А зберігає код у вигляді одного файлу, що сприяє документації проекту. Усі ці фактори роблять Jupyter Notebook зручним та потужним інструментом для розробки та документування проекту машинного навчання.

Середовищем для написання чат бота було обрано продукт компанії JetBrains PyCharm Community Edition 2021. PyCharm — це інтегроване середовище розробки (IDE) для Python, яке надає широкий спектр інструментів для розробки програмного забезпечення [18]. Має потужний редактор коду з підсвіткою синтаксису, автодоповненням, перевіркою помилок та іншими функціями, що полегшують процес програмування.

SQLite — це вбудована реляційна база даних, яка зберігає всю інформацію в одному файлі. Одна з головних переваг SQLite полягає в тому, що для його використання не потрібно окремого сервера баз даних. SQLite може бути використана без обмежень та безкоштовно з будь якою метою. Це робить його ідеальним вибором для локального зберігання та обробки даних в малих або середніх проектах.

Було вирішено використати DB Browser for SQLite для створення бази даних, таблиць та подальшої візуалізацій процесів. Даний продукт сумісних з SQLite [19].

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Модель пригнічення кісткової тканини

3.1.1 Набір даних

Дослідники з Будапештського університету технологій та економіки розробили власні алгоритми тіней ребр, які були використані для приглушення зображень кісток на 247 рентгенограмах грудної клітки. Отримані зображення м'яких тканин без кісток були розміщені вільно доступними [20]. Для створення розширеного набору зображень, було використано афінні перетворення, такі як обертання (від -10 до 10 градусів), горизонтальне та вертикальне зміщення (від -5 до 5 пікселів), горизонтальне відображення, масштабування, медіана, максимум, мінімум та нев'язке розмиття. Загальна кількість отриманих пар зображень склала 4080 зображень.

3.1.2 Початкові заходи

Перш за все потрібно імпортувати потрібні бібліотеки.

Наступним кроком є завантаження набору даних. Ці дії зображенні на рисунку 3.1. Для цього була написана функція getData, яка обходила директорії, зберігаючи зображення зі змінних image_path та target_path. Функція читала ці зображення та змінювала їх розмір до 256 на 256 пікселів. Після цього дані зображень було нормалізовано, поділивши значення пікселів на 255. Цей процес дав вхідні дані для мережі, де значення пікселів тепер знаходяться в діапазоні від 0 до 1.

```
def getData():
    im_with_array = []
    im_without_array = []
    for i in image_list:
        im_with = cv2.imread(os.path.join(image_path,i),cv2.IMREAD_GRAYSCALE)
        im_with = cv2.resize(im_with, (256, 256))[:, :]
        im_with = im_with/255

        im_without = cv2.imread(os.path.join(target_path, i),cv2.IMREAD_GRAYSCALE)
        im_without = cv2.resize(im_without, (256, 256))[:, :]
        im_without = im_without/255


        im_with_array.append(im_with)
        im_without_array.append(im_without)

    return im_with_array, im_without_array

image_array, target_array = getData()
```

Рисунок 3.1 – Розділення даних

3.1.3 Опис моделі

Алгоритм глибокого навчання, який був обраний для створення моделі, придатної для придушення кісткової тканини на рентгенівських знімках — це Автокодувальник.

Автокодувальник – це нейромережева модель, яка використовується для навчання без нагляду з метою здатності зменшувати розмірність та відтворювати вхідні дані. Вона складається з двох основних компонентів: кодера і декодера.

Кодер у складі Автокодувальника використовуватиме згорткові шари Conv2D з ядрами розміром 5x5 та функцією активації ReLU. Він скрадатиметься з трьох шарів з 16, 32 і 64 фільтрами відповідно. Ці шари дозволять виявити локальні особливості та шаблони у вхідних даних, зменшуючи їх розмірність і стискаючи інформацію до низькорозмірного коду. Використання згорткових шарів з різною кількістю фільтрів дозволяє моделі виявляти складніші та абстрактніші особливості зі збільшенням глибини мережі.

Декодер використовуватиме згорткові шари Conv2D та шари UpSampling2D для відновлення вихідних даних з низькорозмірного коду. Він скрадатиметься з шарів з 32 і 16 фільтрами та використовуватиме функцію активації ReLU. Згорткові шари декодера допомагають розгорнути низькорозмірний код та відновити структуру та деталі вихідних даних. Шари UpSampling2D з ядром 2x2 використовуватимуться для збільшення розміру подання шляхом повторення

значень, що допомагає відтворити пропущені деталі та відновити просторову структуру вихідних даних. Використання функції активації sigmoid у останньому шарі декодера дозволить отримати відновлені дані у діапазоні від 0 до 1.

```
def BSS():

  input = Input((256,256,1))

  #encoder
  conv1 = Conv2D(16, (5, 5), activation='relu', padding='same')(input)
  conv2 = Conv2D(32, (5, 5), activation='relu', padding='same', strides=2)(conv1)
  conv3 = Conv2D(64, (5, 5), activation='relu', padding='same', strides=2)(conv2)
  #decoder
  conv4 = Conv2D(32, (5, 5), activation='relu', padding='same')(conv3)
  up1 = UpSampling2D((2,2))(conv4)
  conv5 = Conv2D(16, (5, 5), activation='relu', padding='same')(up1)
  up2 = UpSampling2D((2,2))(conv5)
  decoded = Conv2D(1, (5, 5), activation='sigmoid', padding='same')(up2)

  return  Model(input, decoded)
```

Рисунок 3.2 – Код розробленої моделі за алгоритмом Автокодувальника

Опис кожному шару, що зображеного на рисунку 3.2:

- Вхідний шар (input): Приймає зображення розміром 256x256 пікселів із одним каналом.
- (conv1): Використовується згортковий шар з 16 фільтрами розміром 5x5, активаційною функцією ReLU і параметром "padding='same'", що зберігає розмір вхідного зображення.
- (conv2): Використовується згортковий шар з 32 фільтрами розміром 5x5, активаційною функцією ReLU, параметром "padding='same'" і параметром "strides=2", що зменшує розмір зображення удвічі по обох вимірах.
- (conv3): Використовується згортковий шар з 64 фільтрами розміром 5x5, активаційною функцією ReLU, параметром "padding='same'" і параметром "strides=2", що зменшує розмір зображення удвічі по обох вимірах.
- (conv4): Використовується згортковий шар з 32 фільтрами розміром 5x5, активаційною функцією ReLU і параметром "padding='same'", що зберігає розмір вхідного зображення.
- (up1): Використовується шар звуження, що збільшує розмір зображення удвічі по обох вимірах.

• (conv5): Використовується згортковий шар з 16 фільтрами розміром 5x5, активаційною функцією ReLU і параметром "padding='same'", що зберігає розмір вхідного зображення.

• (up2): Використовується шар звуження, що збільшує розмір зображення удвічі по обох вимірах.

• (decoded): Використовується згортковий шар з одним фільтром розміром 5x5, активаційною функцією сигмоїд і параметром "padding='same'", що зберігає розмір вхідного зображення. Цей шар відповідає за відновлення зображення.

3.1.4 Компіляція

Побудована модель і кількість параметрів на кожному шарі відображено на рисунку 3.3. Таким чином побудована модель містить всього 128 961 параметрів.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 1)]     0

 conv2d (Conv2D)             (None, 256, 256, 16)      416

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      12832

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        51264

 conv2d_3 (Conv2D)           (None, 64, 64, 32)        51232

 up_sampling2d (UpSampling2D  (None, 128, 128, 32)     0
 )

 conv2d_4 (Conv2D)           (None, 128, 128, 16)      12816

 up_sampling2d_1 (UpSampling  (None, 256, 256, 16)     0
 2D)

 conv2d_5 (Conv2D)           (None, 256, 256, 1)       401

=================================================================
Total params: 128,961
Trainable params: 128,961
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.3 – Параметри моделі

Також були прописані функції зворотного виклику. Приклад виклику зображений на рисунку 3.4.

ReduceLROnPlateau, EarlyStopping і ModelCheckpoint — це три поширені функції зворотного виклику, що використовуються в фреймворках машинного навчання. Вони використовуються для поліпшення процесу навчання та покращення продуктивності моделей глибокого навчання.

Функція ModelCheckpoint використовується для збереження ваг моделі під час навчання. Вона дозволяє зберігати модель на різних етапах навчання, наприклад, після кожної епохи або при поліпшенні продуктивності моделі на перевірочному наборі даних.

Параметри:

• weight_path: це шлях, де будуть збережені ваги моделі.

• monitor='val_loss': метрика, яку слід відстежувати для збереження найкращої моделі.

• verbose=1: визначає режим виводу. Буде відображатися повідомлення про збереження моделі.

• save_weights_only=True: вказує, чи потрібно зберігати лише ваги моделі.

• save_best_only=True: будуть збережені лише ваги моделі, коли контрольована метрика поліпшується.

• mode='min': вказує, що потрібно мінімізувати контрольовану метрику.

Функція EarlyStopping використовується для припинення процесу навчання, якщо продуктивність моделі на перевірочному наборі не покращується протягом певної кількості епох. Вона допомагає запобігти перенавчанню моделі і заощаджує обчислювальні ресурси.

Параметри:

• monitor='val_loss': метрика, яку слід відстежувати для припинення навчання.

• patience=10: кількість епох без покращення, після яких навчання буде припинено.

• verbose=1: визначає режим виводу. Якщо встановлено 1, буде відображатися повідомлення, коли відбувається припинення навчання.

• mode='min': вказує, чи потрібно мінімізувати ('min') контрольовану метрику.

Функція ReduceLROnPlateau використовується для динамічного зменшення швидкості навчання, коли продуктивність моделі на валідаційному наборі даних стагнує. Вона допомагає налаштувати модель шляхом поступового зниження швидкості навчання для пошуку кращої оптимальності. Ось розшифровка параметрів, які використовуються в вашому прикладі:

• monitor='val_loss': метрика, яку слід відстежувати для зниження швидкості навчання.

• factor=0.5: коефіцієнт, на який буде зменшена швидкість навчання. Він множиться на поточну швидкість навчання.

• patience=10: кількість епох без покращення, після яких швидкість навчання буде знижена.

• verbose=1: визначає режим виводу. Буде відображатися повідомлення, коли швидкість навчання знижується.

• mode='min': Вказує, що потрібно мінімізувати контрольовану метрику.

• min_lr=0.00001: нижня межа для швидкості навчання. Вона гарантує, що швидкість навчання не опуститься нижче цього значення.

```
weight_path="bss.best.hdf5"

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss',
                             verbose=1, save_weights_only=True,
                             save_best_only=True, mode='min')
earlyStopping = EarlyStopping(monitor='val_loss',
                              patience=10, verbose=1, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10,
                              verbose=1, mode='min', min_lr=0.00001)
callbacks_list = [checkpoint, earlyStopping, reduce_lr]
```

Рисунок 3.4 – Функції зворотного виклику

3.1.5 Навчання моделі

На рисунку 3.5 зображений код навчання моделі з використання параметру validation_data для оцінки продуктивності моделі на незалежному наборі даних, який не використовується для навчання. Під час кожної епохи навчання модель оцінюється на перевірочних даних, і значення метрик, розраховуються для

перевірочного набору. Це дозволяє контролювати процес навчання та виявляти перенавчання або недостатню навчання.

Використання перевірочних даних допомагає приймати рішення щодо функцій зворотного виклику. Це допомагає забезпечити кращу загальну продуктивність моделі на незалежних даних та покращити її здатність до узагальнення.

```
h = model.fit(X_train, Y_train,
              validation_data=(X_val, Y_val),
              epochs=200,
              callbacks= callbacks_list)
```

Рисунок 3.5 – Код навчання моделі

Змінна callbacks_list включає функції зворотного виклику, такі як ModelCheckpoint, EarlyStopping і ReduceLROnPlateau, які були розглянуті раніше.

З рисунку 3.6 видно, що під процесу навчання було зроблено 160 кроків і кожен крок займав близько 850 секунд (приблизно 14 хвилин). Кожен крок це одна ітерація навчання, де модель проходить через одну партію даних. Загальна тривалість навчання складала близько 37 годин. Протягом цього часу модель пристосовувалася до навчального набору даних, зменшуючи втрати та покращуючи свої прогностичні здібності. Найкраще значення було на кроці 150. Модель на даному кроці збережена.

Кінцевий результат — навчена модель, історія навчання h, яка містить метрики та втрати на тренувальному та перевірочному наборах даних, а також набір параметрів, які набула модель під час тривалого процесу навчання.

```
Epoch 1/200
102/102 [==============================] - ETA: 0s - loss: 0.0203 - mae: 0.0238 - ssim_multi_loss: 0.0196 - pnsr: 29.0284 - ssi
m: 0.9363 - ssim_multi: 0.9804
Epoch 1: val_loss improved from inf to 0.01595, saving model to bss.best.hdf5
102/102 [==============================] - 993s 10s/step - loss: 0.0203 - mae: 0.0238 - ssim_multi_loss: 0.0196 - pnsr: 29.0284
- ssim: 0.9363 - ssim_multi: 0.9804 - val_loss: 0.0159 - val_mae: 0.0193 - val_ssim_multi_loss: 0.0153 - val_pnsr: 30.0523 - va
l_ssim: 0.9483 - val_ssim_multi: 0.9847 - lr: 0.0010
Epoch 2/200
102/102 [==============================] - ETA: 0s - loss: 0.0155 - mae: 0.0188 - ssim_multi_loss: 0.0148 - pnsr: 30.1570 - ssi
m: 0.9481 - ssim_multi: 0.9852
Epoch 2: val_loss improved from 0.01595 to 0.01526, saving model to bss.best.hdf5
102/102 [==============================] - 974s 10s/step - loss: 0.0155 - mae: 0.0188 - ssim_multi_loss: 0.0148 - pnsr: 30.1570
- ssim: 0.9481 - ssim_multi: 0.9852 - val_loss: 0.0153 - val_mae: 0.0188 - val_ssim_multi_loss: 0.0146 - val_pnsr: 30.2551 - va
l_ssim: 0.9481 - val_ssim_multi: 0.9854 - lr: 0.0010
Epoch 3/200
102/102 [==============================] - ETA: 0s - loss: 0.0150 - mae: 0.0185 - ssim_multi_loss: 0.0144 - pnsr: 30.2660 - ssi
m: 0.9490 - ssim_multi: 0.9856
Epoch 3: val_loss improved from 0.01526 to 0.01495, saving model to bss.best.hdf5
102/102 [==============================] - 975s 10s/step - loss: 0.0150 - mae: 0.0185 - ssim_multi_loss: 0.0144 - pnsr: 30.2660
- ssim: 0.9490 - ssim_multi: 0.9856 - val_loss: 0.0150 - val_mae: 0.0180 - val_ssim_multi_loss: 0.0144 - val_pnsr: 30.2690 - va
l_ssim: 0.9492 - val_ssim_multi: 0.9856 - lr: 0.0010
Epoch 4/200
102/102 [==============================] - ETA: 0s - loss: 0.0147 - mae: 0.0178 - ssim_multi_loss: 0.0141 - pnsr: 30.4526 - ssi
m: 0.9499 - ssim_multi: 0.9859
Epoch 4: val_loss improved from 0.01495 to 0.01494, saving model to bss.best.hdf5

Epoch 156/200
102/102 [==============================] - ETA: 0s - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4141 - ssi
m: 0.9579 - ssim_multi: 0.9892
Epoch 156: val_loss did not improve from 0.01197
102/102 [==============================] - 844s 8s/step - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4141
- ssim: 0.9579 - ssim_multi: 0.9892 - val_loss: 0.0120 - val_mae: 0.0164 - val_ssim_multi_loss: 0.0111 - val_pnsr: 31.2400 - va
l_ssim: 0.9576 - val_ssim_multi: 0.9889 - lr: 1.0000e-05
Epoch 157/200
102/102 [==============================] - ETA: 0s - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4036 - ssi
m: 0.9578 - ssim_multi: 0.9892
Epoch 157: val_loss did not improve from 0.01197
102/102 [==============================] - 845s 8s/step - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4036
- ssim: 0.9578 - ssim_multi: 0.9892 - val_loss: 0.0120 - val_mae: 0.0164 - val_ssim_multi_loss: 0.0111 - val_pnsr: 31.2581 - va
l_ssim: 0.9576 - val_ssim_multi: 0.9889 - lr: 1.0000e-05
Epoch 158/200
102/102 [==============================] - ETA: 0s - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4389 - ssi
m: 0.9579 - ssim_multi: 0.9892
Epoch 158: val_loss did not improve from 0.01197
102/102 [==============================] - 844s 8s/step - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4389
- ssim: 0.9579 - ssim_multi: 0.9892 - val_loss: 0.0120 - val_mae: 0.0164 - val_ssim_multi_loss: 0.0111 - val_pnsr: 31.2415 - va
l_ssim: 0.9576 - val_ssim_multi: 0.9889 - lr: 1.0000e-05
Epoch 159/200
102/102 [==============================] - ETA: 0s - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4183 - ssi
m: 0.9579 - ssim_multi: 0.9892
Epoch 159: val_loss did not improve from 0.01197
102/102 [==============================] - 844s 8s/step - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4183
- ssim: 0.9579 - ssim_multi: 0.9892 - val_loss: 0.0120 - val_mae: 0.0164 - val_ssim_multi_loss: 0.0111 - val_pnsr: 31.2293 - va
l_ssim: 0.9576 - val_ssim_multi: 0.9889 - lr: 1.0000e-05
Epoch 160/200
102/102 [==============================] - ETA: 0s - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4420 - ssi
m: 0.9579 - ssim_multi: 0.9892
Epoch 160: val_loss did not improve from 0.01197
102/102 [==============================] - 844s 8s/step - loss: 0.0116 - mae: 0.0162 - ssim_multi_loss: 0.0108 - pnsr: 31.4420
- ssim: 0.9579 - ssim_multi: 0.9892 - val_loss: 0.0120 - val_mae: 0.0164 - val_ssim_multi_loss: 0.0111 - val_pnsr: 31.2399 - va
l_ssim: 0.9576 - val_ssim_multi: 0.9889 - lr: 1.0000e-05
Epoch 160: early stopping
```

Рисунок 3.6 – Процес навчання моделі перших та останніх 4 кроків

Було побудовано графіки зміни Втрат(loss), середьої абсолютна похибка (MAE), структурної схожості (SSIM) та пікового відношення сигналу до шуму (PSNR).

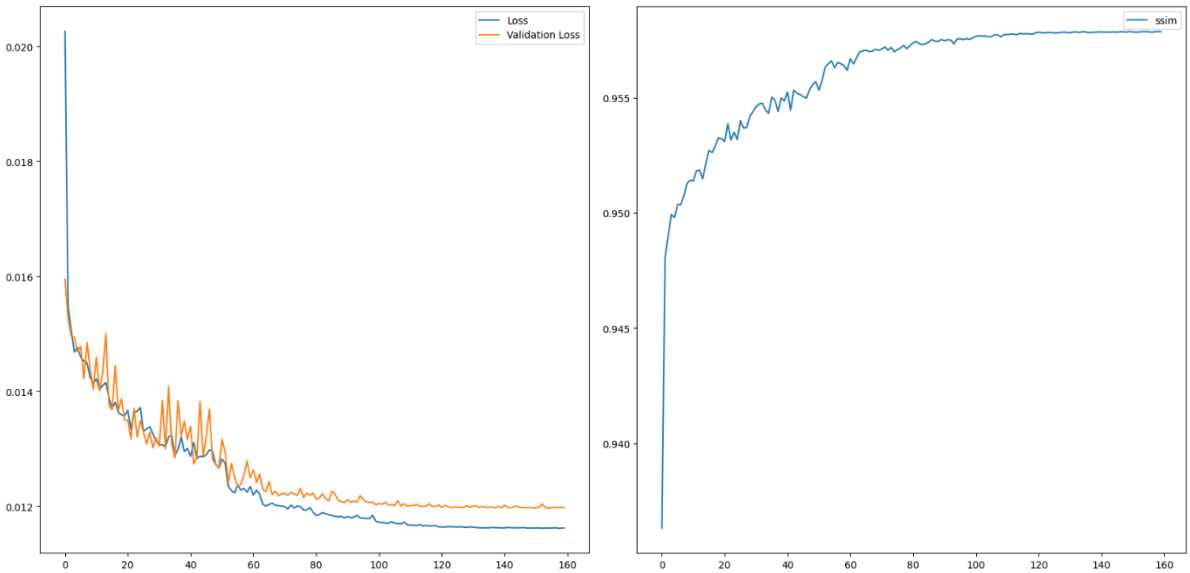


Рисунок 3.7 – Графіки Втрат (а) та SSIM (б)

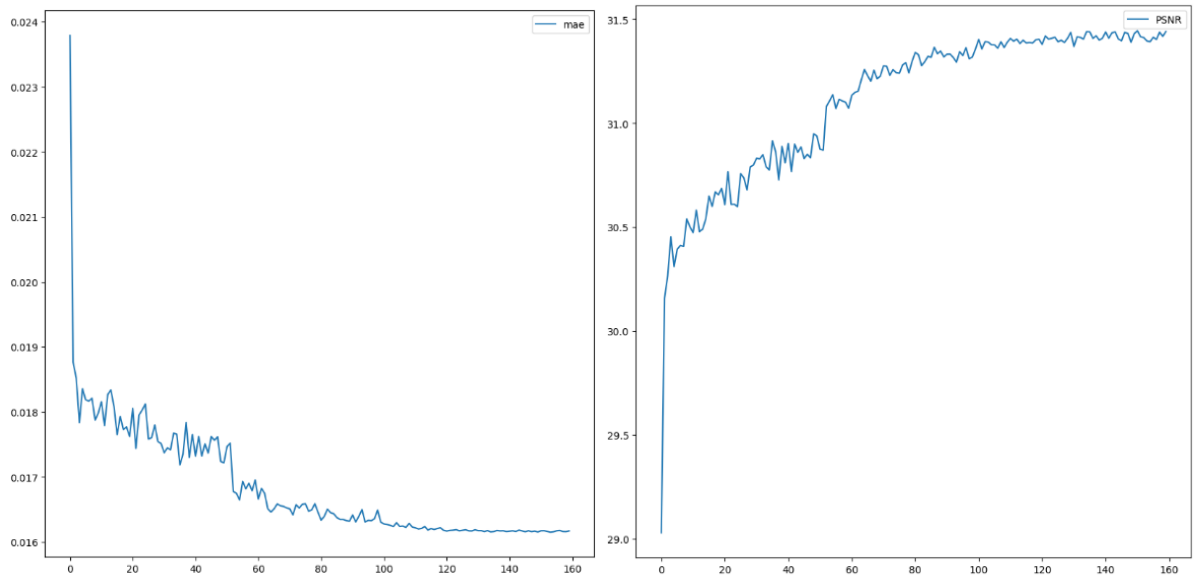


Рисунок 3.8 – Графіки MAE (а) та PSNR (б)

Рисунок 3.7 (а) відображає значення функцій втрат на тренувальному та перевірочному наборах даних з кожною ітерацією навчання. На 150 кроці Значення втрат на дорівнює 0.0116, перевірочних втрат(val_loss) дорівнює 0.0120

Рисунок 3.7 (б) відображає значення структурного схожості зображень з часом. Значення SSIM на 150 кроці дорівнює 0.9579

Рисунок 3.8 (а) відображає середню абсолютну похибку з часом. Чим менше значення MAE, тим краще прогнозування моделі. Значення MAE на 150 кроці дорівнює 0.0162

Рисунок 3.8 (б) відображає значення пікового відношення сигналу до шуму (PNSR) з часом. Значення PSNR на 150 кроці дорівнює 31.431.

З графіків видно, що навчання покращувалося на кожному кроці, Втрати та абсолютна похибка зменшувались, а структурна схожості зображень та відношення сигналу до шуму збільшувалась. На 100 кроці почалась зменшуватись величина змін, але завдяки функції ReduceLROnPlateau перенавчання не відбулося. Додаткові 50 кроків завершили навчання моделі дали змогу ще трішки навчити модель. Майбутнє навчання моделі може перенавчити модель.

3.1.6 Тестування

Після навчанні моделі було проведено її тестування на тестовому наборі, який не був задіяний у навчанні моделі.

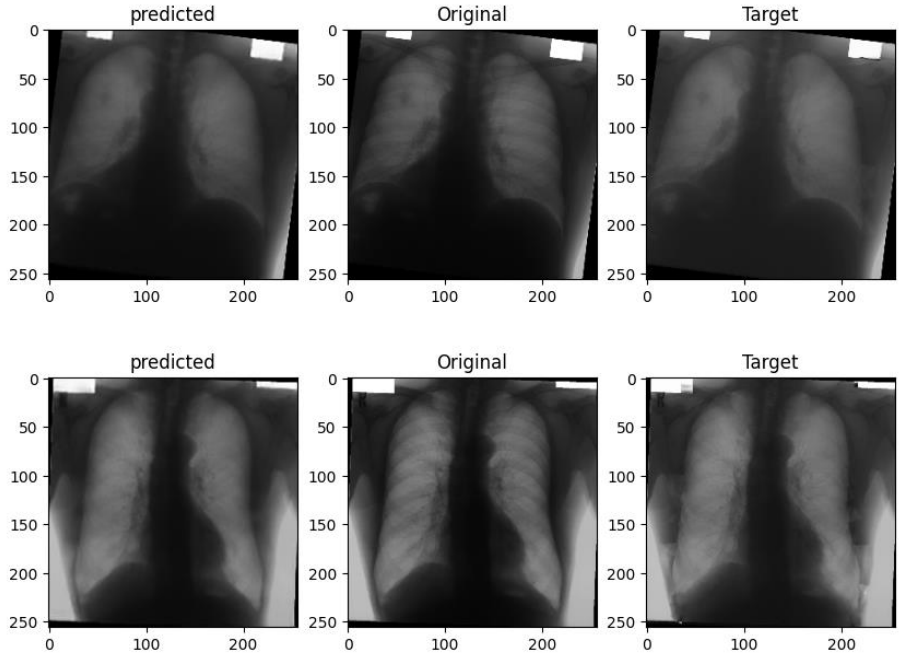


Рисунок 3.9 – Приклад 2 зображень спрогнозованих моделлю

З рисунку 3.9 видно, що модель вправилась з поставленим завданням.

3.2 Моделі розпізнання рентгенівських зображень

3.2.1 Набір даних

Набір даних Chest X-Ray Images складається Гуанчжоуським жіночим і дитячим медичним центром (Гуанчжоу, Китай) як частина звичайного клінічного догляду за педіатричними пацієнтами. Остання версія цього набору даних складається з 5856 рентгенівських зображень. Він був упорядковано в 3 дек (train, test, val) і містить піддеки для кожної категорії зображення (Pneumonia/Normal).

Дека train містить 3110 рентгенівських знімків з патологіями легень та 1082 без виявлених патологій. Дека test містить 773 зображень з патологіями легень та 267 без виявлених патологій. Дека val містить 390 зображень з патологіями легень та 234 без виявлених патологій.

Рентгенівські зображення грудної клітини були відібрані з педіатричних пацієнтів віком від одного до п’яти років із жіночого та дитячого медичного центру Гуанчжоу, Гуанчжоу. Усі рентгенівські дослідження органів грудної клітки проводилися як частина звичайної клінічної допомоги пацієнтам.

Для аналізу рентгенівських зображень грудної клітки всі рентгенівські зображення спочатку перевіряли для контролю якості шляхом видалення всіх зображень з низької якості або нечитабельних. Потім набір даних був оцінений двома лікарями-експертами після яких перевірив третій експерт, щоб врахувати будь-які помилки [21].

3.2.2 Початкові заходи

Наступним кроком після завантаження потрібних бібліотек потрібно завантажити набір даних.

З рисунку 3.10 видно, що функцію getData було модифіковано для отримання на вхід директорії зі зображеннями.

```
labels = ['PNEUMONIA', 'NORMAL']
def getData(data_dir):
    im_array = []
    target_array = []
    for label in labels:
        path = os.path.join(data_dir, label)
        target_numer = labels.index(label)
        for img in os.listdir(path):
            image_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
            image_array = cv2.resize(image_array, (256, 256))
            image_array = image_array / 255

            im_array.append(image_array)
            target_array.append(target_numer)
    return im_array, target_array


image_array_train, target_array_train = getData('chest_xray/train')
image_array_test, target_array_test = getData('chest_xray/test')
image_array_val, target_array_val = getData('chest_xray/val')
```

Рисунок 3.10 – Завантаження набору даних

Щоб підвищити точність начальної моделі, прийнято рішення ці набори даних (навчальний, тестовий та перевірочний) знов перетасувати.

На рисунку 3.11 зображений процес об’єднання наборів.

```
X = []
y = []

for feature, label in zip(image_array_train,target_array_train):
    X.append(feature)
    y.append(label)

for feature, label in zip(image_array_test,target_array_test):
    X.append(feature)
    y.append(label)

for feature, label in zip(image_array_val,target_array_val):
    X.append(feature)
    y.append(label)
```

Рисунок 3.11 – Об’єднання в один набір

А на рисунку 3.12 процес розподілу за допомогою train_test_split (рис 3.12).

```
X_train, X_val, Y_train, Y_val = train_test_split(X, y,
                                                  test_size=0.2,
                                                  random_state=42)
X_val, X_test, Y_val, Y_test = train_test_split(X_val, Y_val,
                                                  test_size=0.2,
                                                  random_state=42)
print(X_train.shape, X_val.shape,X_test.shape,Y_train.shape, Y_val.shape,Y_test.shape)
```

```
(4684, 256, 256, 1) (937, 256, 256, 1) (235, 256, 256, 1) (4684,) (937,) (235,)
```

Рисунок 3.12 – Розділення даних

Рисунок 3.13 відображає схематичне розділення, яке здійснювалося у наступних пропорціях: 80 на 16 та 4, що призвело до отримання 4684 зображень у навчальному наборі, 937 зображень у тестовому наборі та 235 зображення у перевірочному наборі. Ці набори мають відношення з патологіями до без 3 до 1. Схематичне зображення на (рис 3.13).

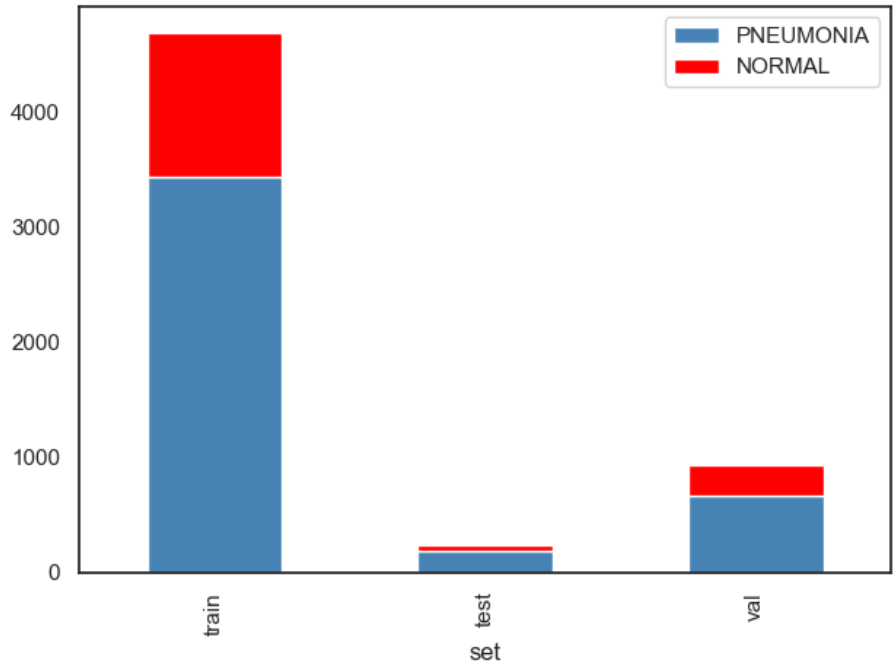


Рисунок 3.13 – Статистика розділення даних

Щоб порівняти доцільність, а також ефективність попередньої моделі, пригнічення кісткової тканини, було вирішено навчити 2 моделі класифікації рентгенівських зображень. Одну з використанням моделі пригнічення кісткової тканини на вхідних даних, та без.

На рисунку 3.14 зображено різницю коду. В одній з двох моделей перед розділенням функцією train_test_split набір було відправлено моделі з попереднього розділу.

```
X = bss.predict(X)
183/183 [==============================] - 2426s 13s/step
```

Рисунок 3.14 – Пригнічення кісткової тканини в одній з моделей

3.2.3 Аугментація Даних

Одним із ключових факторів, який впливає на ефективність моделі, є якість та розмір тренувального набору даних. Часто може статися, що набір даних обмежений або не репрезентативний, що призводить до перенавчання або поганої узагальнюючої здатності моделі.

Один з способів подолати ці обмеження полягає в використанні методів аугментації даних. Аугментація даних — це процес штучного збільшення тренувального набору даних шляхом здійснення різних операцій перетворення, таких як повороти, зміщення, збільшення/зменшення масштабу та перевертання [22]. Це дозволяє отримати нові зображення, що є варіаціями вхідних зображень, зберігаючи при цьому семантичну інформацію.

Використання методів аугментації даних має декілька переваг. Воно може допомогти покращити робастість моделі, зробити її менш чутливою до шуму та змін у вхідних даних. Крім того, аугментований тренувальний набір даних може допомогти запобігти перенавчанню і поліпшити узагальнюючу здатність моделі до нових прикладів.

ImageDataGenerator — це клас з бібліотеки Keras, який дозволяє генерувати зображення в режимі реального часу з підготованими даними під час тренування моделі нейронної мережі.

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=90,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        vertical_flip=True)

datagen.fit(X_train)
```

Рисунок 3.15 – Аугментація даних

На рисунку 3.15 зображений код виклику екземпляр ImageDataGenerator з параметрами:

• featurewise_center=False: Не віднімати середнє значення пікселів зображення по всій вибірці.

• samplewise_center=False: Не віднімати середнє значення пікселів кожного зображення.

• featurewise_std_normalization=False: Не нормалізувати пікселі зображення, ділячи на стандартне відхилення по всій вибірці.

• samplewise_std_normalization=False: Не нормалізувати пікселі кожного зображення, ділячи на його стандартне відхилення.

• zca_whitening=False: Не застосовувати алгоритм ZCA знечорнення.

• rotation_range=90: Випадкові повороти зображень на кут в межах від -90 до 90 градусів.

• zoom_range=0.1: Випадкове збільшення або зменшення масштабу зображень на 10%.

• width_shift_range=0.1: Випадкове горизонтальне зсування зображень на 10% ширини.

• height_shift_range=0.1: Випадкове вертикальне зсування зображень на 10% висоти.

• horizontal_flip=True: Випадкове горизонтальне перевернення зображень.

• vertical_flip=True: Випадкове вертикальне перевернення зображень.

3.2.4 Опис моделей

Для написання моделі розпізнання рентгенівських зображень була використана архітектура CNN (згорткова нейронна мережа), оскільки це ефективно працює з великими об'ємами даних, які мають просторову структуру, таку як зображення. Вона добре показує себе у виявленні різних ознак та шаблонів у зображеннях, що допомагає здійснювати класифікацію, сегментацію або розпізнавання об'єктів [23].

Спочатку я ініціалізував модель і додав перший згортковий шар. Додаю шар активації ReLU. Потім додаю шар згорткового пулінгу. Після цього додаю шар нормалізації пакетів.

Продовжую додавати шари згорткового шару, активації, пулінгу та нормалізації. Потім додаю шар "flatten", який перетворює тривимірні карти ознак у одновимірні вектори ознак.

Далі додаю шари регуляризації, зокрема "dropout", який допомагає уникнути перенавчання шляхом випадкового відключення певних нейронів. Після цього додаю пов'язаний (fully connected) шар.

На останок, додаю останній пов'язаний шар з одним нейроном і активацією sigmoid, оскільки в даному випадку підходить бінарна класифікація.

```
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=(256,256,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Flatten())

model.add(Dropout(0.2))
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

Рисунок 3.16 – Модель за алгоритмом CNN

Опис дій які виконуються на кожному шарі на рисунку 3.16:

• Згортковий шар (Conv2D): Використовую фільтри розміром 3x3 з 256 каналами для виявлення різних ознак на зображенні. Завдяки параметру padding='same', зберігається розмір вхідних зображень, а отримані ознаки будуть мати такий же розмір.

• Шар активації ReLU: ReLU перетворює від'ємні значення в нулі, а позитивні значення залишає без змін.

• Шар згорткового пулінгу(MaxPooling2D): Використовую пулінг з розміром 2x2 для зменшення розмірності отриманих ознак і зменшення кількості параметрів моделі.

• Шар нормалізації пакетів(BatchNormalization): Нормалізує активації згорткового пулінгу в межах кожного пакету (batch) шляхом центрування та масштабування.

• Повторюємо додавання шарів згорткового шару, активації, пулінгу та нормалізації для створення додаткових рівнів ознак у моделі. Це допомагає моделі вивчати більш складні шаблони на зображеннях.

• ("flatten"): Перетворює тривимірні карти ознак в одновимірний вектор ознак, що дозволяє передавати ці ознаки до пов'язаного шару.

• ("dropout"): Випадково вимикає певну кількість нейронів з метою регуляризації і уникнення перенавчання. Використовується значення 0.2, що означає, що друга частина нейронів буде випадково вимкнена на кожній епосі навчання.

• (fully connected): Має 64 нейрони і активацію ReLU. Цей шар допомагає збільшити складність моделі та розпізнавати більш високорівневі ознаки у зображеннях.

• Знову застосовуємо шар "dropout" з імовірністю 0.2.

• fully connected): Має один нейрон з активацією sigmoid, що відповідає за бінарну класифікацію. Значення, ближче до 0, вказує на один клас, а ближче до 1 — на інший клас.

3.2.5 Компіляція

Використовується функція втрат "binary_crossentropy", яка підходить для бінарної класифікації. Оптимізатор Adam з навчальною швидкістю 0.0001 використовується для оптимізації моделі.

Побудована модель і кількість параметрів на кожному шарі відображено на рисунку 3.17. Таким чином побудована модель містить всього 1 208 913 параметрів з яких 448 не будуть навчатися під час тренування моделі.

Ймовірно ці 448 ненавчальних параметрів походять від шару "batch_normalization". Цей шар має додаткові параметри для нормалізації

активацій в межах кожного пакету, і ці параметри можуть бути попередньо встановленими значеннями, які не змінюються під час навчання моделі.

Зазвичай ненавчальні параметри використовуються для забезпечення деякої константної функціональності моделі або для ініціалізації параметрів, які пізніше будуть навчатися. Це допомагає забезпечити стабільність та швидкість навчання моделі.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 256)     2560

 activation (Activation)     (None, 256, 256, 256)     0

 max_pooling2d (MaxPooling2D  (None, 128, 128, 256)    0
 )

 batch_normalization (BatchN  (None, 128, 128, 256)    512
 ormalization)

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      147520

 activation_1 (Activation)   (None, 128, 128, 64)      0

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 batch_normalization_1 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 64, 64, 16)        9232

 activation_2 (Activation)   (None, 64, 64, 16)        0

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 batch_normalization_2 (Batc  (None, 32, 32, 16)       128
 hNormalization)

 flatten (Flatten)           (None, 16384)             0

 dropout (Dropout)           (None, 16384)             0

 dense (Dense)               (None, 64)                1048640

 activation_3 (Activation)   (None, 64)                0

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

 activation_4 (Activation)   (None, 1)                 0

=================================================================
Total params: 1,208,913
Trainable params: 1,208,465
Non-trainable params: 448
_________________________________________________________________
```

Рисунок 3.17 – Параметри моделей

Метрики для оцінки моделі є точність (accuracy).

Accuracy (точність) є однією з найбільш простих та загальноприйнятих метрик оцінки ефективності моделей машинного навчання. Вона вимірює, наскільки точно модель передбачає правильні відповіді.

Для обчислення accuracy, спочатку порівнюються передбачені значення моделі зі знаними правильними відповідями. Кожне передбачене значення порівнюється з відповідним правильним значенням. Якщо передбачення збігається з правильним значенням, воно вважається "правильним", в іншому випадку — "неправильним". Потім обчислюється відношення кількості правильних передбачень до загальної кількості передбачень.

Також були прописані функції зворотного виклику, з відмінністю з передньою моделлю у параметрі patience=3 (кількість епох без покращення, після яких навчання буде припинено у випадку функції EarlyStopping або знижена у випадку функції ReduceLROnPlateau)

3.2.6 Навчання моделей

Дані моделі навчалась з використання параметру validation_data для оцінки продуктивності моделі на незалежному наборі даних, який не використовується для навчання та методу datagen.flow генерує потік (послідовність) пакетів даних для тренування моделі, які будуть використовуватись під час процесу навчання. Під час кожної епохи навчання модель оцінюється на перевірочних даних, і значення метрик точності (accuracy).

На рисунку 3.18 зображений код компіляції моделі.

```
h = model.fit(datagen.flow(X_train, y_train),
              validation_data=(X_val, y_val),
              epochs=20,
              callbacks= callbacks_list)
```

Рисунок 3.18 –Компіляція моделі

Змінна callbacks_list включає функції зворотного виклику, такі як ModelCheckpoint, EarlyStopping і ReduceLROnPlateau, які були розглянуті раніше.

3.2.7 Результати моделі без використання пригнічення кісткової тканини

На рисунку 3.19 зображений процес навчання моделі.

Під час процесу навчання було зроблено 13 кроків і кожен крок займав близько 980 секунд (приблизно 17 хвилин). Загальна тривалість навчання складала близько 4 годин. Найкраще значення було на кроці 9. Модель на даному кроці збережена.

```
Epoch 1/20
118/118 [==============================] - ETA: 0s - loss: 0.6308 - accuracy: 0.7078
Epoch 1: val_loss improved from inf to 0.56677, saving model to cnn.best.hdf5
118/118 [==============================] - 982s 8s/step - loss: 0.6308 - accuracy: 0.7078 - val_loss: 0.5668 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 2/20
118/118 [==============================] - ETA: 0s - loss: 0.5048 - accuracy: 0.7561
Epoch 2: val_loss did not improve from 0.56677
118/118 [==============================] - 979s 8s/step - loss: 0.5048 - accuracy: 0.7561 - val_loss: 0.6798 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 3/20
118/118 [==============================] - ETA: 0s - loss: 0.4414 - accuracy: 0.7852
Epoch 3: val_loss did not improve from 0.56677
118/118 [==============================] - 973s 8s/step - loss: 0.4414 - accuracy: 0.7852 - val_loss: 0.7788 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 4/20
118/118 [==============================] - ETA: 0s - loss: 0.3872 - accuracy: 0.8228
Epoch 4: val_loss improved from 0.56677 to 0.53526, saving model to cnn.best.hdf5
118/118 [==============================] - 975s 8s/step - loss: 0.3872 - accuracy: 0.8228 - val_loss: 0.5353 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 5/20
118/118 [==============================] - ETA: 0s - loss: 0.3518 - accuracy: 0.8457
Epoch 5: val_loss improved from 0.53526 to 0.42049, saving model to cnn.best.hdf5
118/118 [==============================] - 987s 8s/step - loss: 0.3518 - accuracy: 0.8457 - val_loss: 0.4205 - val_accuracy: 0.
7673 - lr: 1.0000e-04

Epoch 6: val_loss improved from 0.42049 to 0.26340, saving model to cnn.best.hdf5
118/118 [==============================] - 979s 8s/step - loss: 0.3228 - accuracy: 0.8505 - val_loss: 0.2634 - val_accuracy: 0.
8869 - lr: 1.0000e-04
Epoch 7/20
118/118 [==============================] - ETA: 0s - loss: 0.3109 - accuracy: 0.8700
Epoch 7: val_loss improved from 0.26340 to 0.24508, saving model to cnn.best.hdf5
118/118 [==============================] - 972s 8s/step - loss: 0.3109 - accuracy: 0.8700 - val_loss: 0.2451 - val_accuracy: 0.
8943 - lr: 1.0000e-04
Epoch 8/20
118/118 [==============================] - ETA: 0s - loss: 0.3079 - accuracy: 0.8631
Epoch 8: val_loss improved from 0.24508 to 0.22681, saving model to cnn.best.hdf5
118/118 [==============================] - 971s 8s/step - loss: 0.3079 - accuracy: 0.8631 - val_loss: 0.2268 - val_accuracy: 0.
9168 - lr: 1.0000e-04
Epoch 9/20
118/118 [==============================] - ETA: 0s - loss: 0.2909 - accuracy: 0.8780
Epoch 9: val_loss improved from 0.22681 to 0.22665, saving model to cnn.best.hdf5
118/118 [==============================] - 999s 8s/step - loss: 0.2909 - accuracy: 0.8780 - val_loss: 0.2267 - val_accuracy: 0.
9178 - lr: 1.0000e-04
Epoch 10/20
118/118 [==============================] - ETA: 0s - loss: 0.2616 - accuracy: 0.8919
Epoch 10: val_loss improved from 0.22665 to 0.19804, saving model to cnn.best.hdf5
118/118 [==============================] - 973s 8s/step - loss: 0.2616 - accuracy: 0.8919 - val_loss: 0.1980 - val_accuracy: 0.
9296 - lr: 1.0000e-04
Epoch 11/20
118/118 [==============================] - ETA: 0s - loss: 0.2660 - accuracy: 0.8922
Epoch 11: val_loss did not improve from 0.19804
118/118 [==============================] - 1007s 9s/step - loss: 0.2660 - accuracy: 0.8922 - val_loss: 0.2748 - val_accuracy:
0.8773 - lr: 1.0000e-04
Epoch 12/20
118/118 [==============================] - ETA: 0s - loss: 0.2661 - accuracy: 0.8948
Epoch 12: val_loss did not improve from 0.19804
118/118 [==============================] - 992s 8s/step - loss: 0.2661 - accuracy: 0.8948 - val_loss: 0.8456 - val_accuracy: 0.
6233 - lr: 1.0000e-04
Epoch 13/20
118/118 [==============================] - ETA: 0s - loss: 0.2611 - accuracy: 0.9007
Epoch 13: val_loss did not improve from 0.19804

Epoch 13: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
118/118 [==============================] - 983s 8s/step - loss: 0.2611 - accuracy: 0.9007 - val_loss: 0.2383 - val_accuracy: 0.
8997 - lr: 1.0000e-04
Epoch 13: early stopping
```

Рисунок 3.19– Процес навчання моделі

Було побудовано графіки зміни Втрат(loss) та точності(accurancy)

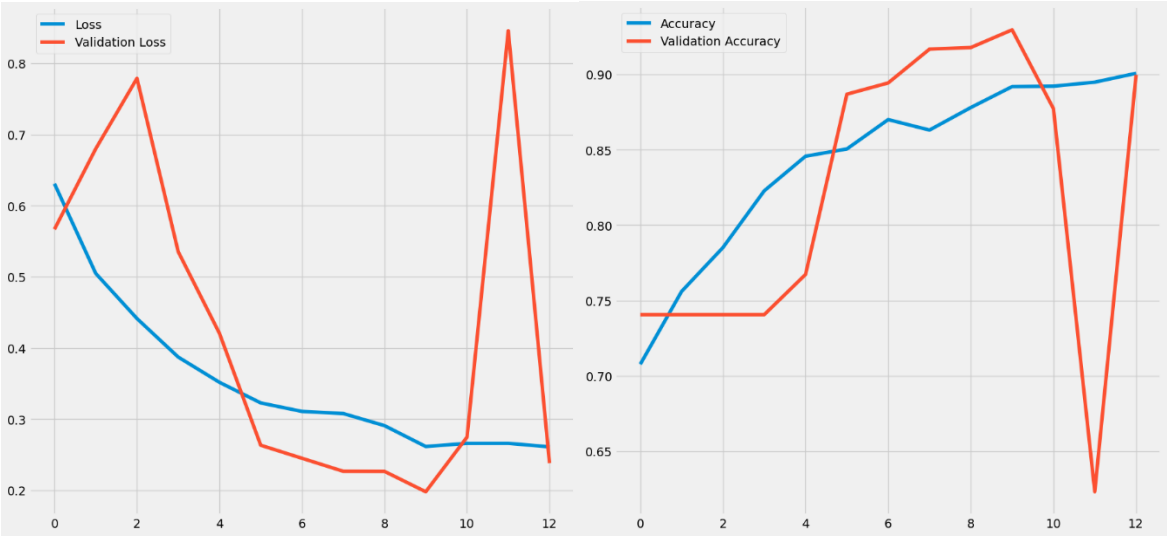


Рисунок 3.20 – Графіки Втрат (а) та Точності (б)

На рисунку 3.20 видно, що з кроком втрати зменшуються, а точність збільшується. З 5 кроку втрати та точність перевірочного набору менші(більші) ніж на навчального набору. Це значить, що дана модель добре підходить.

На рисунку 3.21 зображені метрики втрат та точності на тестовому наборі становлять і становлять 0.2466 та 0.8908 відповідно.

```
model.evaluate(X_test, Y_test)
37/37 [==============================] - 79s 2s/step - loss: 0.2466 - accuracy: 0.8908
[0.24657677114009857, 0.8907849788665771]
```

Рисунок 3.21 – Оцінка моделі

Також було оцінено інші характеристики навчання мережі такі як точність (precision), відгук (recall) та f-міру (f1-score). Це зображено на рисунку 3.22. Та на їх основі побудовані графіки (рис. 3.23- 3.25)

Загальні метрики для моделі на тренувальному наборі даних:

```
Accuracy on testing set: 0.9138225255972696
Precision on testing set: 0.9331306990881459
Recall on testing set: 0.7953367875647669
F1-score on testing set: 0.8587412587412587
```

Рисунок 3.22 – Основні метрики

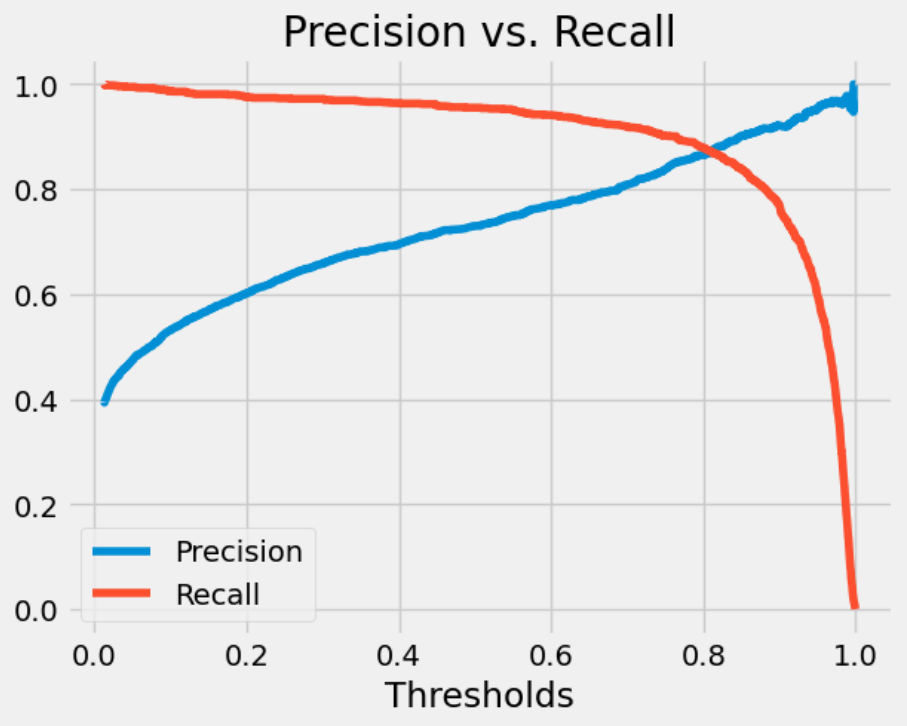


Рисунок 3.23 – Графік Precision-Recall

На рисунку 3.23 зображений Графік Precision-Recall (точність-повнота), який є інструментом для оцінки якості моделі класифікації. Він дозволяє виміряти ефективність моделі для різних значень порога прийняття рішення.

На графіку Precision-Recall по осі X відображається повнота (Recall), яка визначається як співвідношення правильно виявлених екземплярів позитивного класу до загальної кількості екземплярів позитивного класу. По осі Y відображається точність (Precision), яка визначається як співвідношення правильно виявлених екземплярів позитивного класу до загальної кількості екземплярів, визначених моделлю як позитивний клас.

Графік Precision-Recall підсумовує компроміс між справжнім позитивним коефіцієнтом і позитивним прогнозним значенням для прогнозної моделі з використанням різних порогів ймовірності. Зазвичай, чим більше площа під кривою Precision-Recall, тим краща модель класифікації.

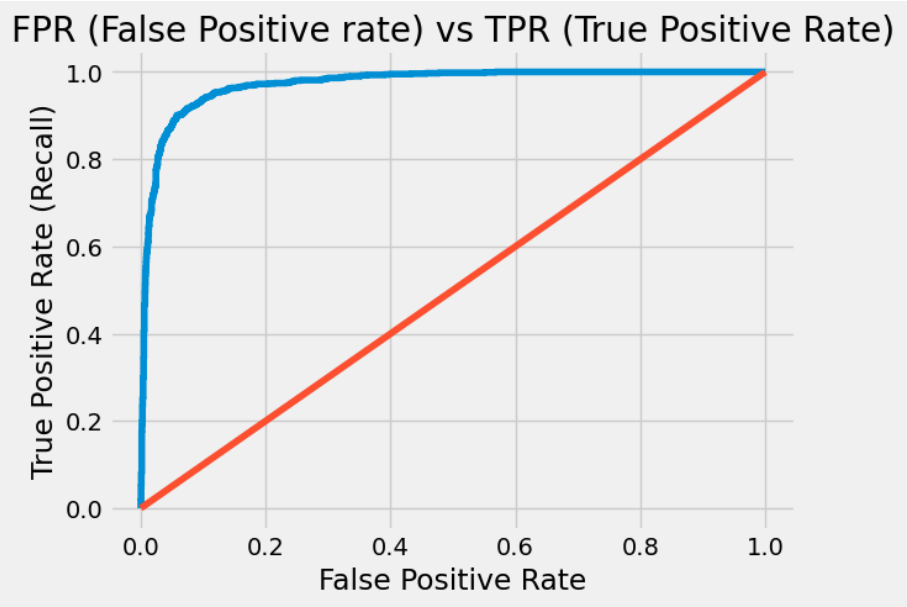


Рисунок 3.24 – Графік ROC

На рисунку 3.24 зображений Графік ROC, який також використовується для оцінки якості моделі класифікації. Він дозволяє виміряти здатність моделі розрізнити між позитивними і негативними класами.

На графіку ROC по осі X відображається специфічність (Specificity), яка визначається як співвідношення правильно виявлених екземплярів негативного класу до загальної кількості екземплярів негативного класу. По осі Y відображається чутливість (Sensitivity), яка визначається як співвідношення правильно виявлених екземплярів позитивного класу до загальної кількості екземплярів позитивного класу.

Графік ROC дозволяє аналізувати чутливість та специфічність моделі класифікації. Ідеальна модель класифікації буде мати ROC-криву, яка проходить через верхній лівий кут графіка (чутливість = 1, специфічність = 1). Зазвичай, чим більше площа під кривою ROC, тим краща модель класифікації.

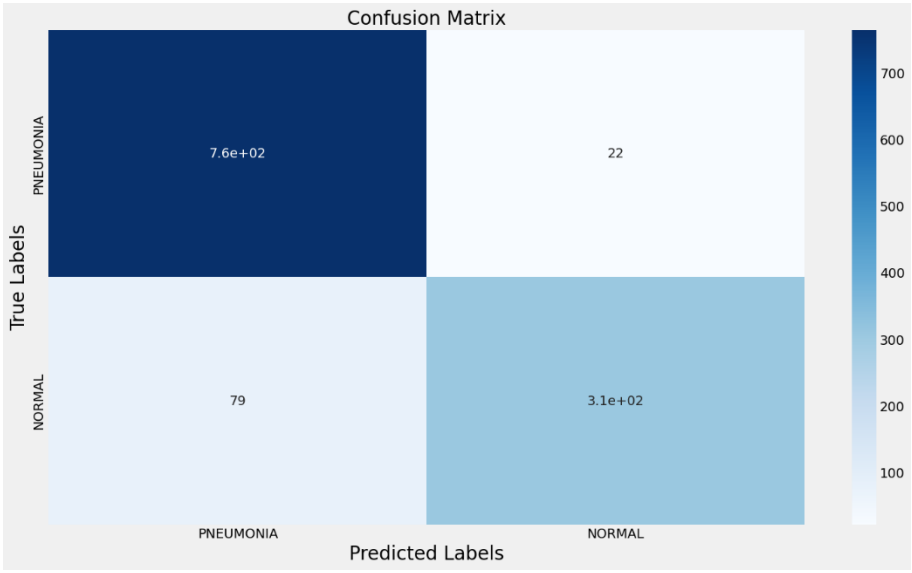


Рисунок 3.25 – матриця Плутанини

На рисунку 3.25 зображена матриця Плутанини, яка використовується для візуалізації продуктивності моделі класифікації на основі фактичних і передбачених значень класу. Вона представляє собою таблицю, де рядки відповідають фактичним класам, а стовпці — передбаченим класам.

Матриця Плутанини містить чотири основні значення:

True Positive (TP): Кількість екземплярів позитивного класу, які правильно виявлені моделлю.

True Negative (TN): Кількість екземплярів негативного класу, які правильно визначені моделлю.

False Positive (FP): Кількість екземплярів негативного класу, які помилково визначені моделлю як позитивний клас (тип I помилка).

False Negative (FN): Кількість екземплярів позитивного класу, які помилково визначені моделлю як негативний клас (тип II помилка).

Матриця Плутанини дозволяє оцінити точність, чутливість, специфічність, а також виявити типи помилок, які вона робить.

Ці три графіки та матриця є потужними інструментами для аналізу і визначення продуктивності моделей класифікації у машинному навчанні.

З матриці видно, що дана модель хибно ідентифікувала 22+79 зображень.

На рисунку 3.26 зображені результати класифікації з тестового набору.

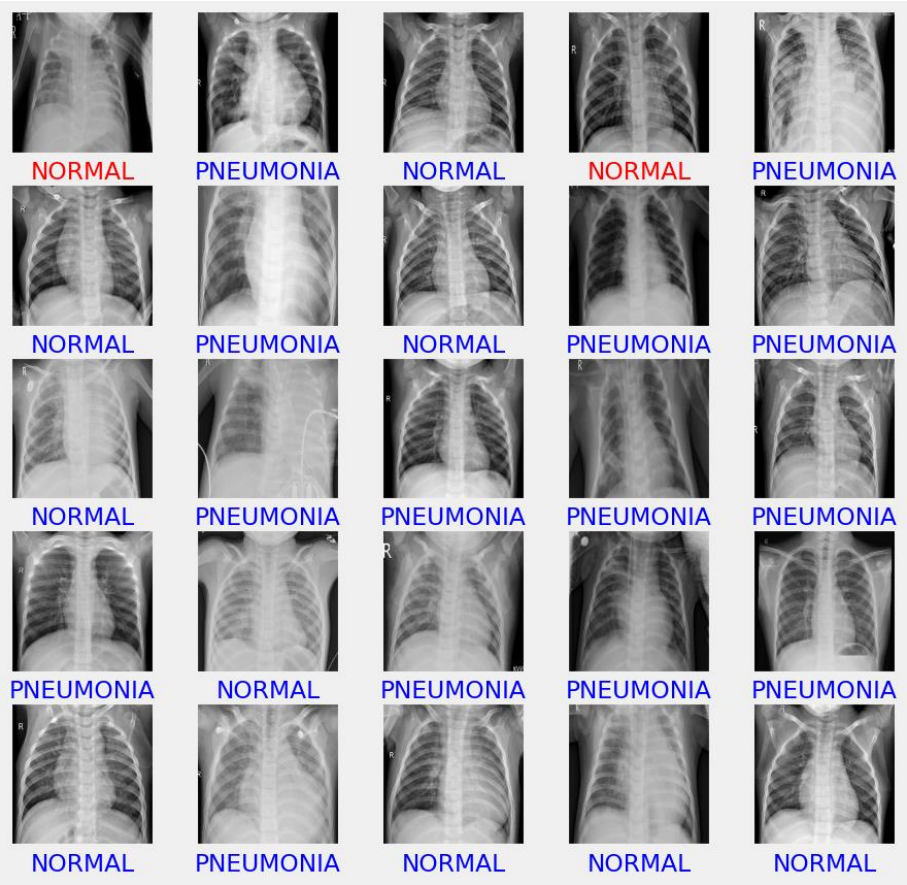


Рисунок 3.26 – Прогнозування 25 тестових зображень

3.2.8 Результати моделі з використання пригнічення кісткової тканини

На рисунку 3.27 зображений процес навчання моделі. Під час якого було зроблено 11 кроків і кожен крок займав близько 965 секунд (приблизно 16 хвилин). Загальна тривалість навчання складала близько 3.5 годин. Найкраще значення було на кроці 8. Модель на даному кроці збережена.

```
Epoch 1/20
118/118 [==============================] - ETA: 0s - loss: 0.4542 - accuracy: 0.7961
Epoch 1: val_loss improved from inf to 0.70379, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 971s 8s/step - loss: 0.4542 - accuracy: 0.7961 - val_loss: 0.7038 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 2/20
118/118 [==============================] - ETA: 0s - loss: 0.2532 - accuracy: 0.8940
Epoch 2: val_loss did not improve from 0.70379
118/118 [==============================] - 964s 8s/step - loss: 0.2532 - accuracy: 0.8940 - val_loss: 0.8500 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 3/20
118/118 [==============================] - ETA: 0s - loss: 0.2097 - accuracy: 0.9157
Epoch 3: val_loss improved from 0.70379 to 0.66836, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 961s 8s/step - loss: 0.2097 - accuracy: 0.9157 - val_loss: 0.6684 - val_accuracy: 0.
7407 - lr: 1.0000e-04
Epoch 4/20
118/118 [==============================] - ETA: 0s - loss: 0.2035 - accuracy: 0.9234
Epoch 4: val_loss improved from 0.66836 to 0.30390, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 962s 8s/step - loss: 0.2035 - accuracy: 0.9234 - val_loss: 0.3039 - val_accuracy: 0.
8367 - lr: 1.0000e-04
Epoch 5/20
118/118 [==============================] - ETA: 0s - loss: 0.1653 - accuracy: 0.9367
Epoch 5: val_loss improved from 0.30390 to 0.21644, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 961s 8s/step - loss: 0.1653 - accuracy: 0.9367 - val_loss: 0.2164 - val_accuracy: 0.
9104 - lr: 1.0000e-04
Epoch 6/20
118/118 [==============================] - ETA: 0s - loss: 0.1629 - accuracy: 0.9416
Epoch 6: val_loss improved from 0.21644 to 0.14100, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 962s 8s/step - loss: 0.1629 - accuracy: 0.9416 - val_loss: 0.1410 - val_accuracy: 0.
9530 - lr: 1.0000e-04

Epoch 6: val_loss improved from 0.21644 to 0.14100, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 962s 8s/step - loss: 0.1629 - accuracy: 0.9416 - val_loss: 0.1410 - val_accuracy: 0.
9530 - lr: 1.0000e-04
Epoch 7/20
118/118 [==============================] - ETA: 0s - loss: 0.1481 - accuracy: 0.9402
Epoch 7: val_loss did not improve from 0.14100
118/118 [==============================] - 962s 8s/step - loss: 0.1481 - accuracy: 0.9402 - val_loss: 0.1443 - val_accuracy: 0.
9520 - lr: 1.0000e-04
Epoch 8/20
118/118 [==============================] - ETA: 0s - loss: 0.1420 - accuracy: 0.9442
Epoch 8: val_loss improved from 0.14100 to 0.12708, saving model to cnn_seg_weights.best.hdf5
118/118 [==============================] - 963s 8s/step - loss: 0.1420 - accuracy: 0.9442 - val_loss: 0.1271 - val_accuracy: 0.
9605 - lr: 1.0000e-04
Epoch 9/20
118/118 [==============================] - ETA: 0s - loss: 0.1236 - accuracy: 0.9530
Epoch 9: val_loss did not improve from 0.12708
118/118 [==============================] - 965s 8s/step - loss: 0.1236 - accuracy: 0.9530 - val_loss: 0.1310 - val_accuracy: 0.
9584 - lr: 1.0000e-04
Epoch 10/20
118/118 [==============================] - ETA: 0s - loss: 0.1209 - accuracy: 0.9530
Epoch 10: val_loss did not improve from 0.12708
118/118 [==============================] - 964s 8s/step - loss: 0.1209 - accuracy: 0.9530 - val_loss: 0.1329 - val_accuracy: 0.
9562 - lr: 1.0000e-04
Epoch 11/20
118/118 [==============================] - ETA: 0s - loss: 0.1319 - accuracy: 0.9496
Epoch 11: val_loss did not improve from 0.12708

Epoch 11: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
118/118 [==============================] - 964s 8s/step - loss: 0.1319 - accuracy: 0.9496 - val_loss: 0.1273 - val_accuracy: 0.
9584 - lr: 1.0000e-04
Epoch 11: early stopping
```

Рисунок 3.27 –Процес навчання моделі

Було побудовано графіки зміни Втрат(loss) та точності(accurancy)

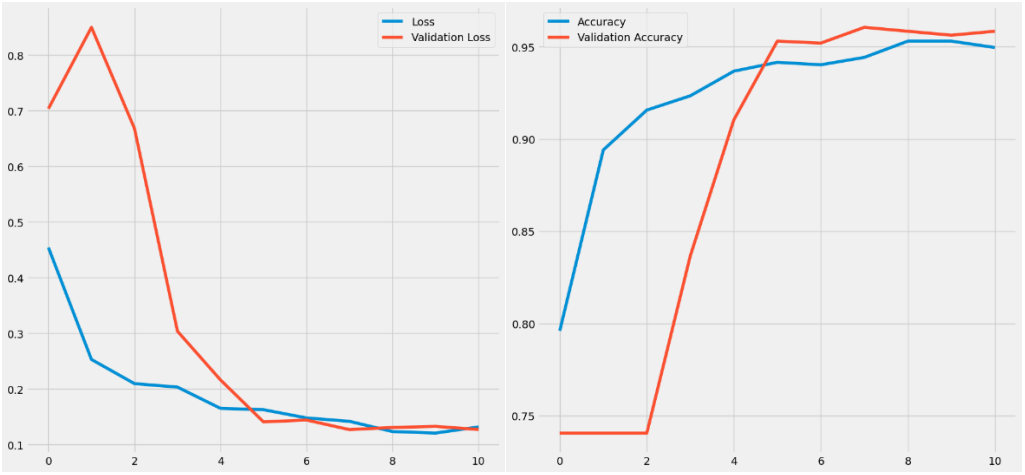


Рисунок 3.28 – Графіки Втрат (а) та Точності (б)

На рисунку 3.28 видно, що з кроком втрати зменшуються, а точність збільшується. З 5 кроку втрати та точність перевірочного набору менші(більші) ніж на навчального набору. Це значить, що дана модель добре підходить.

На рисунку і становлять 0.1161 та 0.0990 відповідно.

```
model.evaluate(X_test, Y_test)
37/37 [==============================] - 73s 2s/step - loss: 0.1161 - accuracy: 0.9590
[0.11613279581069946, 0.9590443968772888]
```

Рисунок 3.29 – Оцінка моделі

Загальні метрики для моделі на тренувальному наборі даних:

```
Accuracy on testing set: 0.9334470989761092
Precision on testing set: 0.9908814589665653
Recall on testing set: 0.8129675810473815
F1-score on testing set: 0.8931506849315067
```

Рисунок 3.30 – Основні метрики

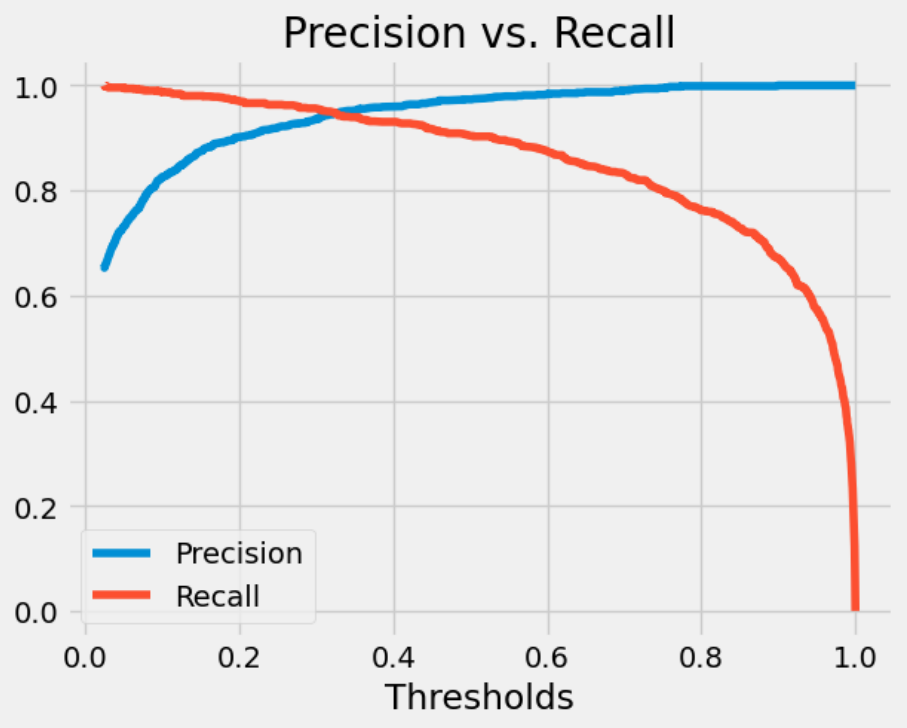


Рисунок 3.31 – Графік Precision-Recall

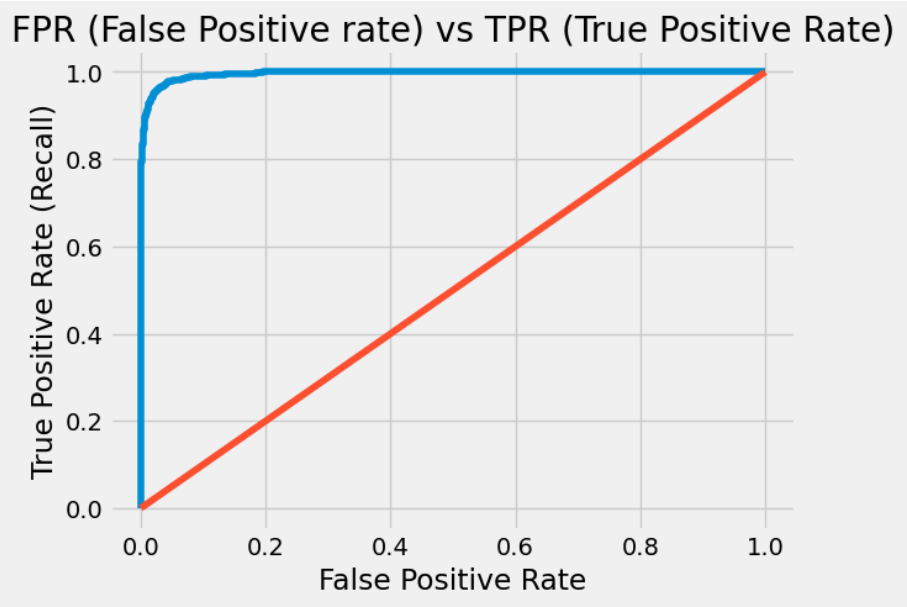


Рисунок 3.32 – Графік ROC

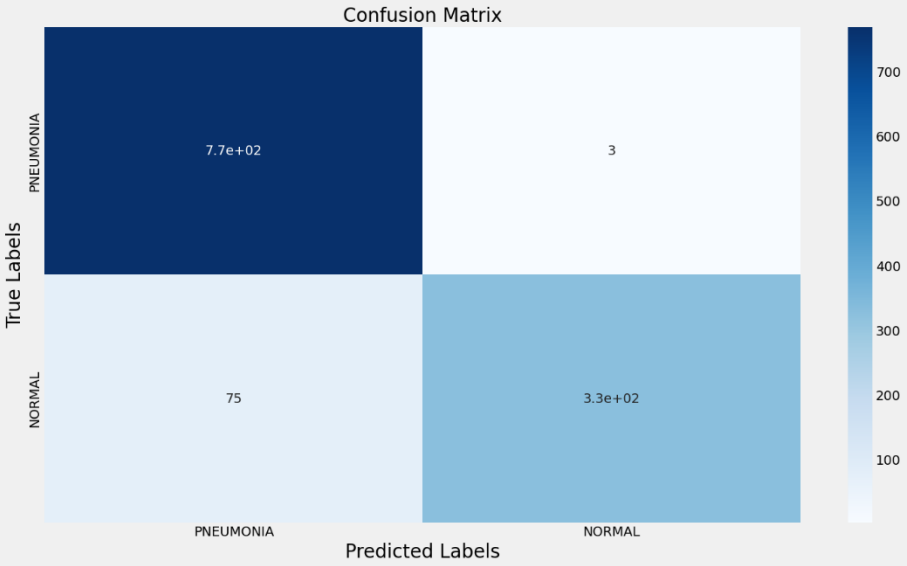


Рисунок 3.33 – Матриця Плутанини

Із рисунків 3.30 та 3.31 видно, що дана модель хибно ідентифікувала 3+75 зображень. Дана модель краще ідентифікує пневмонію, що не дивно знаючи коефіцієнт 3 до 1 без патологій.

На рисунку 3.34 зображені результати класифікації з тестового набору.

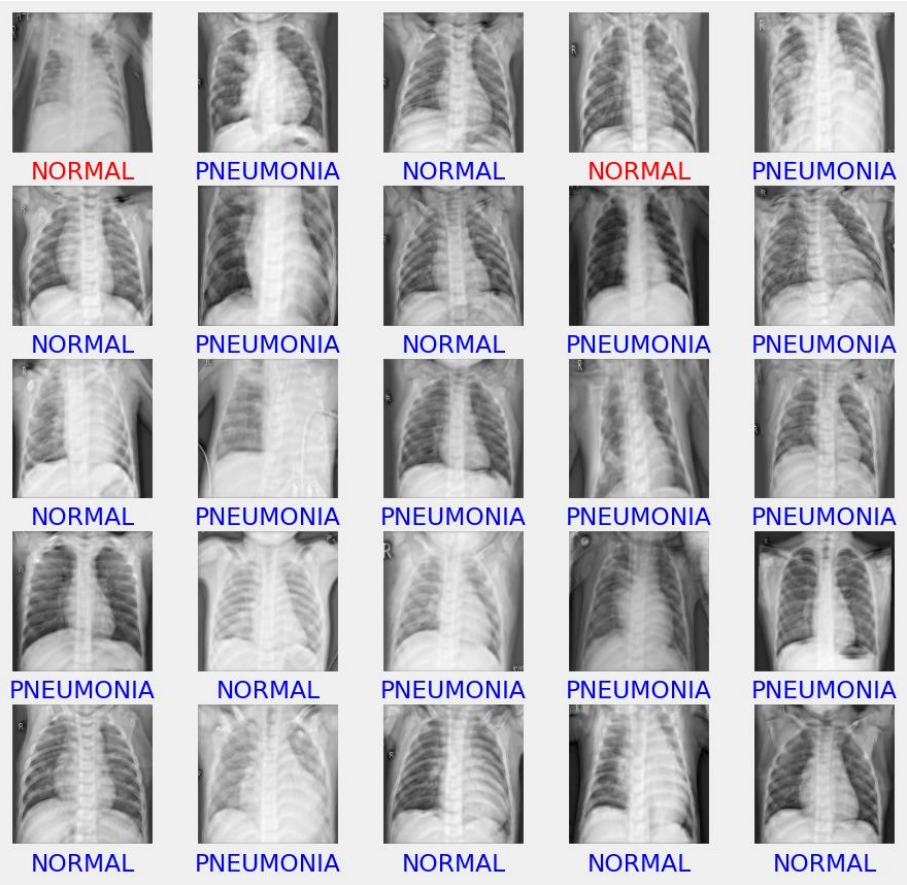


Рисунок 3.34 – Прогнозування 25 тестових зображень

3.2.9 Висновок

Модель з використання пригнічення кісткової тканини виявилася кращою за кількома метриками порівняно з моделлю без використання пригнічення кісткової тканини. Вона має вищу точність, точність, повноту і F1-показник. Це означає, що загальна якість передбачень моделі 2 є кращою. Крім того, модель 2 має менше хибно позитивних випадків, що є важливим аспектом з точки зору уникнення неправильного класифікації негативних випадків як позитивних.

Зважаючи на ці висновки, рекомендується використовувати модель 2 в подальших дослідженнях або застосуваннях, оскільки вона демонструє кращу здатність до класифікації з вищою точністю та повнотою. Проте, для демонстрації розбіжностей між моделями в чат-боті Telegram будуть застосовані обидві.

3.3 Розроблення чат–бота Telegram

3.3.1 Інформаційна модель

Для опису роботи коду та її складовий було схематично зображено принцип роботи (рис. 3.35).

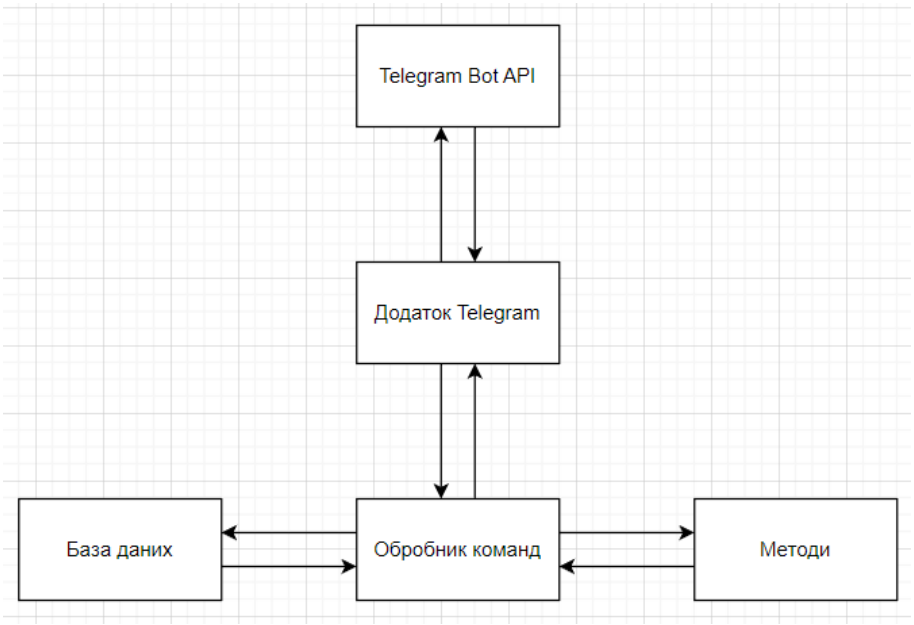


Рисунок 3.35 – Структура бота

Із рисунку видно, що:

• Telegram виконує роль платформи, де буде розміщений бот.

• Telegram Bot API виконує роль посередника між Telegram та нашими обробником команд. Він отримує повідомлення від користувача і в разі коректних вхідних даних відправляє відповідь користувачу.

• Обробник команд отримує дані від Telegram Bot API, знаходить сценарій та виконує його за допомогою Бази даних та описаних методів вирішення. Сформований результат відправляє назад користувачу через Telegram Bot API.

Для моделювання взаємодії між Користувачем та Ботом (Контролером) було використано діаграму варіантів використання (Use Case Diagram), яка зображена на рисунку 3.36.

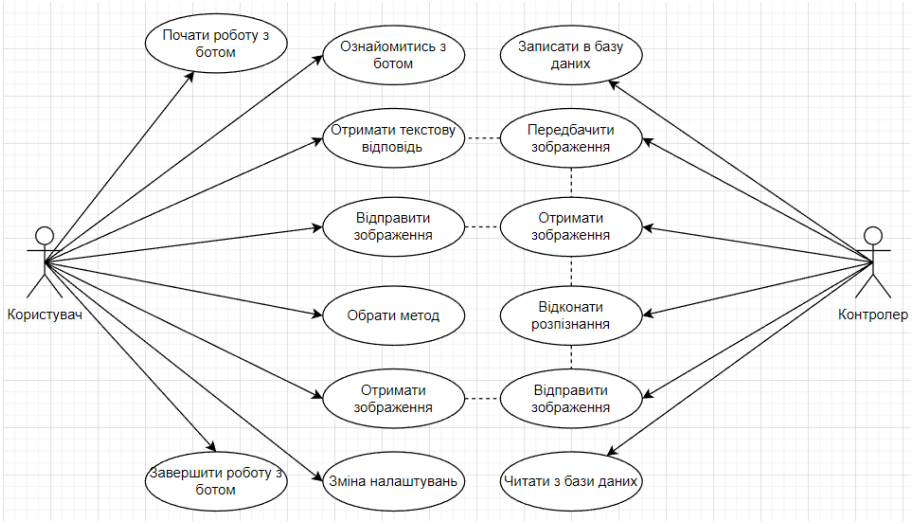


Рисунок 3.36 – Use Case Diagram

Діаграма варіантів використання допомагає визначити функціональність системи та її зовнішніх інтерфейсів, а також виявити основні сценарії взаємодії з користувачами або зовнішніми системами. Вона слугує засобом спілкування між розробниками, замовниками та іншими зацікавленими сторонами, допомагаючи узгодити спільне розуміння функціональних вимог до системи.

3.3.2 Створення бота

Для того щоб створити бота потрібно написати спеціальному боту @BotFather [14] який автоматизовано виконує ці дії. Обрати команду «/newbot». Далі вказати ім’я та username бота. Обов’язковою умовою є вказання закінчення bot, щоб користувачі могли розрізняти інших членів громади Telegram.

Результатом цих дій стане токен, за допомогою якого можна використовувати бота. Цей процес зображений на рисунку 3.37.

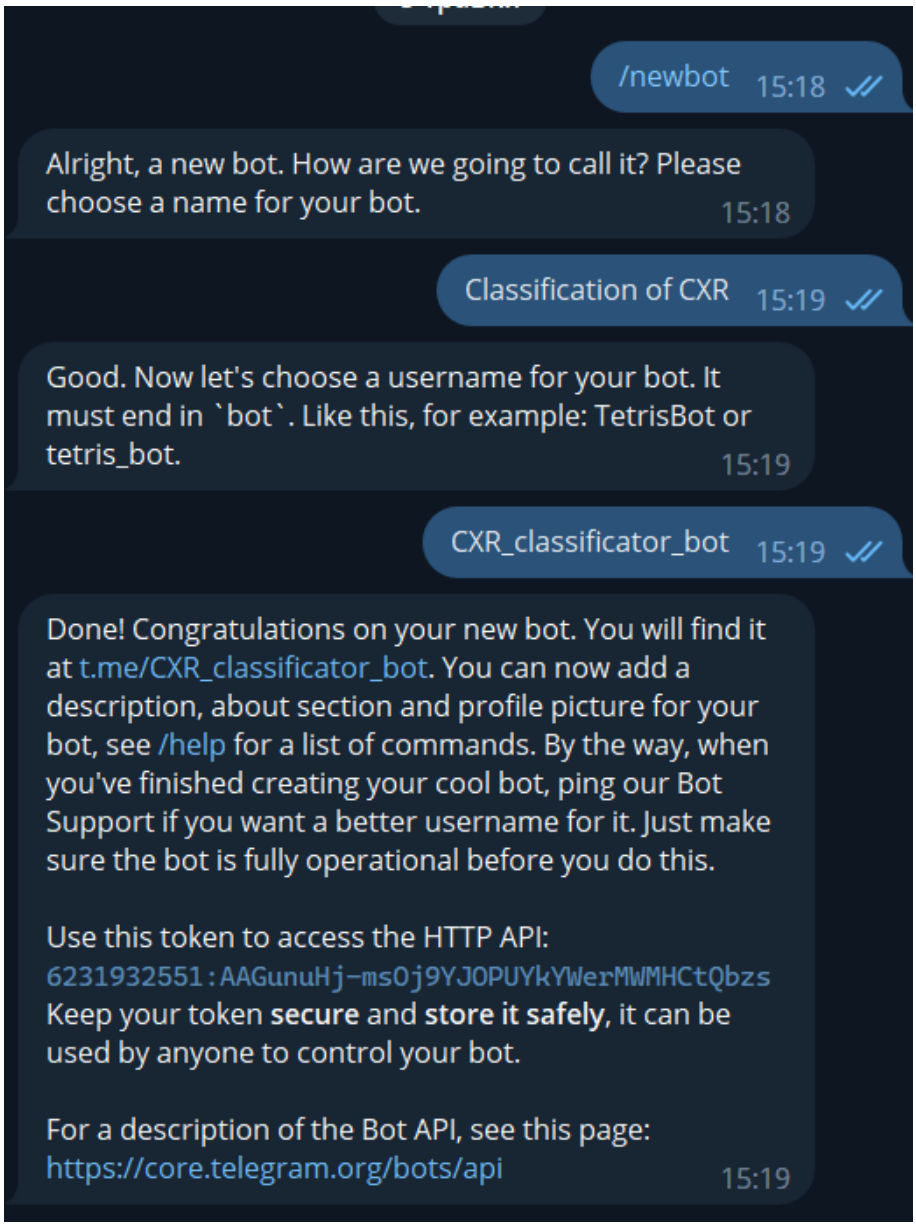


Рисунок 3.37 – Мінімальний набір команд для створення бота

3.3.3 Створення бази даних

Бувають аварійної ситуації та щоб користувачі відчували себе у комфорті, дані будуть зберігатись у базі даних. Це будуть параметри користувачів, такі як мова наприклад.

А також на основі вхідних зображень, можна побудувати нові набори даних, що в майбутньому допоможе краще розпізнавати рентгенівських зображень. Тому всі надіслані користувачами зображення краще зберігати.

На рисунку 3.38 зображена початкова сторінка СУБД DB Browser for SQLite.

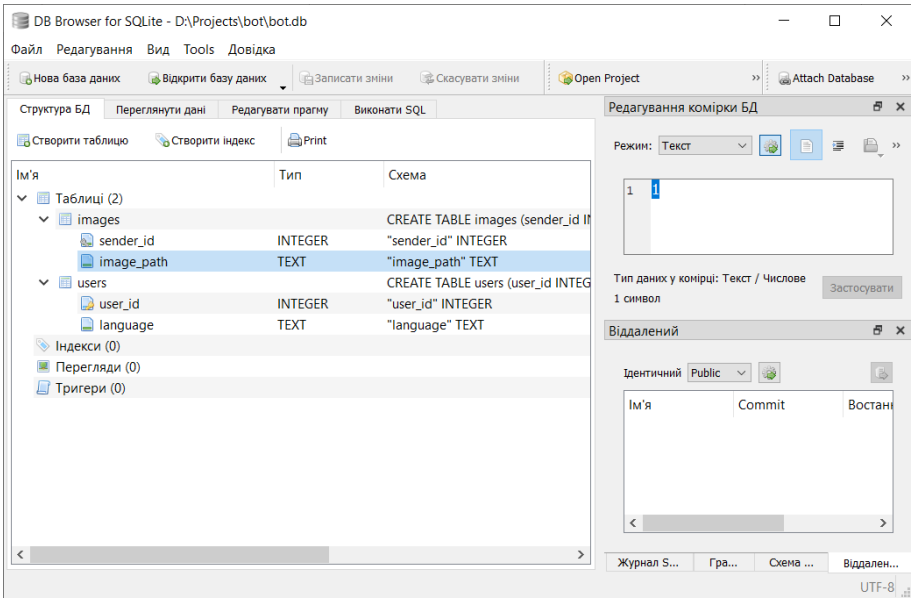


Рисунок 3.38 – Головна сторінка СУБД

На рисунку 3.39 показаний код таблиці для налаштування користувачів.

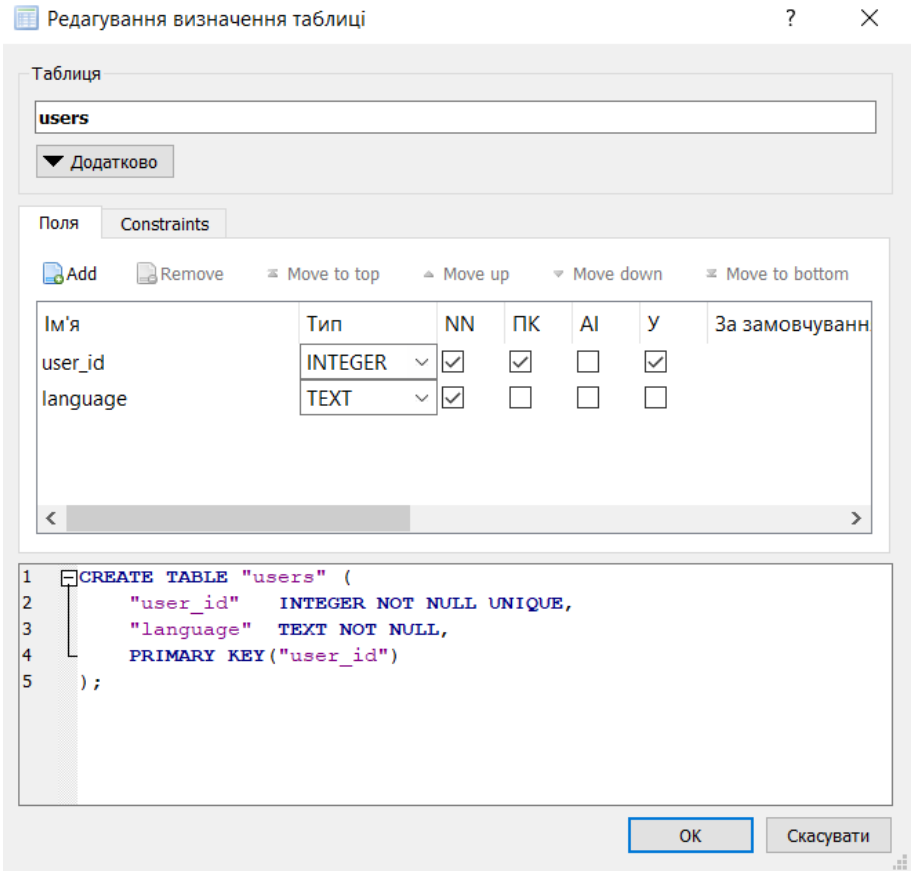


Рисунок 3.39 – Створення таблиці users

На рисунку 3.40 наведений код таблиці для відправників зображень.

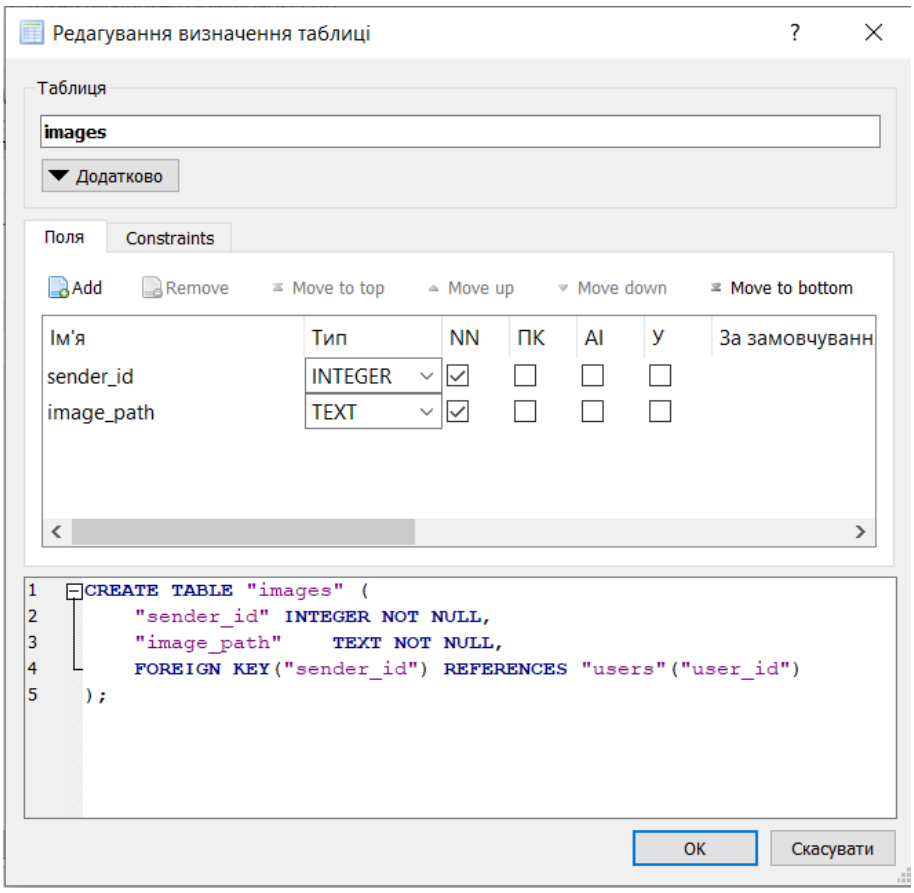


Рисунок 3.40 – Створення таблиці images

Зображення, що надсилають користувачі, можна не зберігати на пристрої. Усі ці дані можна читати з серверів Telegram. Це дає змогу зекономити на пам'яті пристроїв. Лиш у випадках використання користувачем моделі (методу) пригнічення кісткової тканини ми повинні зберегти спрогнозоване зображення, щоб потім його відправити.

3.3.4 Опис виконаної роботи

Лиш отримавши токен від @BotFather можна починати роботу. Структура створеного проекту зображена на рисунку 3.41.

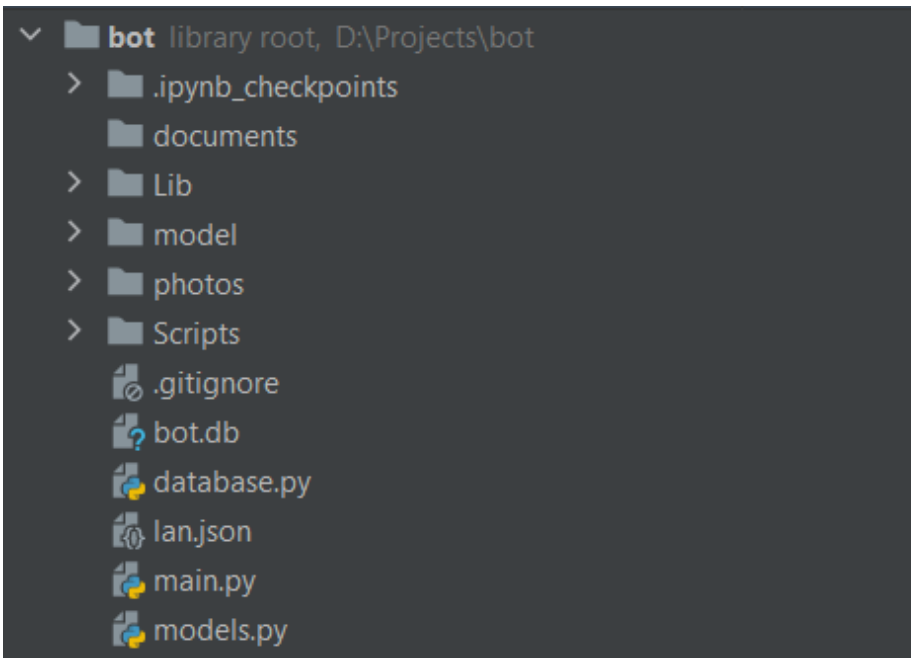


Рисунок 3.41 – Структура проекту

Оскільки користувач може надсилати повідомлення контролеру у вигляді зображення та файлів, створено 2 деки documents для зображень, які передали через файлів та photos для стиснених зображень. Тут зберігаються зображення, які будуть відправлятися користувачу.

Дека Model зберігає моделі з попереднього розділу, а саме bss, x_ray та x_ray_bss, а також їх ініціалізація. Користувач зможе користуватися ними.

Файл Bot.db є базою даних для даного проекту. Тут зберігають дані користувачів та зображення. В майбутньому буде доповнюватися іншими параметрами.

Для того, щоб залучити більшу кількість користувачів, написаний двомовний бот. Текст різних мов збережений у файлі lan.json.

Файл models.py описує взаємодію між моделями з попереднього розділу та ботом.

Основний файл, з якого запускається проект, це main.py. Тут знаходить Токен бота та все меню користувача.

На рисунку 3.42 зображений код ініціалізації бота з файлу main.py. В даний момент токен виступає як приклад коду. Він замінений заради безпеки користувачів.

```
# створення змінних для назв кнопок та повідомлень для кожної мови
TOKEN = '6231932551:AAGunuHj-msOj9YJOPUYkYWerMWMHCtQbzs'
bot = telebot.TeleBot('6231932551:AAGunuHj-msOj9YJOPUYkYWerMWMHCtQbzs')
```

Рисунок 3.42 – Ініціалізація бота

Також присутнє привітання нового користувача. На рисунку 3.43 зображений код, з якого видно, що мова бота залежить від обраної мови додатку. Якщо додаток на українській мові, то і бот відповідає українською мовою, інакше англійська мова. Змінити Мову можна у параметрах.

```
# обробник команди /start
@bot.message_handler(commands=['start'])
def start(message):
    language = message.from_user.language_code
    if language == 'uk':
        bd.set_language(message.chat.id, 'uk')
        action_user_lan[message.chat.id] = 'uk'
    else:
        bd.set_language(message.chat.id, 'en')
        action_user_lan[message.chat.id] = 'en'

    main_menu(message.chat.id, data[language]["bot_description"])
```

Рисунок 3.43 –Програмна реалізації першої взаємодії з користувачем

У разі, якщо виникне аварійна ситуація, параметр мови зберігається в базі даних.

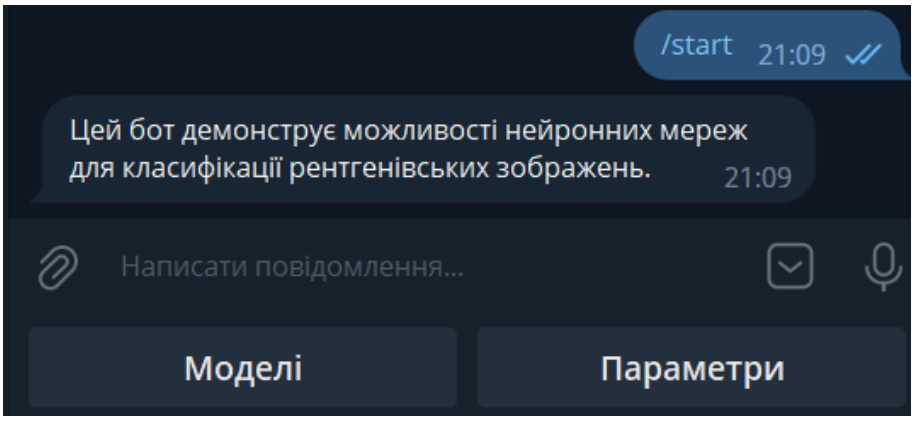


Рисунок 3.44 – Вигляд привітання у користувача

На рисунку 3.44 видно, що під час привітання з’являється перше меню вибору. Це моделі та параметри. Код даного меню зображений на рисунку 3.45.

```python
# основне меню з кнопками "Модель" та "Параметри"
def main_menu(user_id, text = None):
    lan = user_lan(user_id)
    keyboard = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    model_button = telebot.types.KeyboardButton(data[lan]['models'])
    params_button = telebot.types.KeyboardButton(data[lan]['parameters'])
    keyboard.add(model_button, params_button)
    bot.send_message(user_id, text=text, reply_markup=keyboard)
```

Рисунок 3.45 – Програмна реалізації першого меню

На початку йде перевірка мови користувача. На основі мови формуються кнопки-переходи з обраною мовою, які через Telegram Bot API передаються користувачу.

Для прикладу змінимо мову бота. Для цього натиснемо на Параметри. З’явиться нове меню, яке зображено на рисунку 3.46.

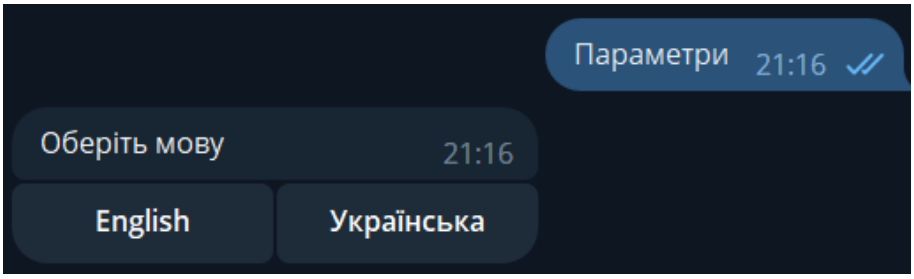


Рисунок 3.46 – Вигляд меню зміни мови

На рисунку 3.47 зображений код меню Параметри.

```python
# обробник кнопки "Параметри"
@bot.message_handler(func=lambda message: message.text ==  'Parameters' or message.text == 'Параметри')
def parameters(message):
    # меню з кнопками вибору мови
    lan = user_lan(message.chat.id)
    language_menu = telebot.types.InlineKeyboardMarkup()
    en_button = telebot.types.InlineKeyboardButton('English', callback_data='en')
    ua_button = telebot.types.InlineKeyboardButton('Українська', callback_data='uk')
    language_menu.add(en_button, ua_button)
    bot.send_message(message.chat.id, data[lan]['choose_language'], reply_markup=language_menu)
```

Рисунок 3.47 – Програмна реалізації меню зміни параметру мови

Оберемо English, оскільки в даний момент доступні лиш 2 мови, і Українська – за замовчуванням.

Рисунок 3.48 демонструє зміну мови. На рисунку 3.49 зображений код, завдяки якому можна зрозуміти, що також відбувся запис у базу даних параметра мови.

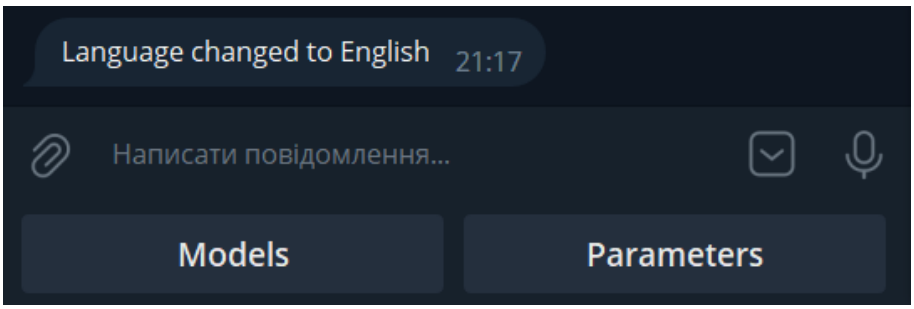


Рисунок 3.48 – Оновлений вигляд меню зі зміненою мовою

```
# обробник вибору мови
@bot.callback_query_handler(func=lambda call: call.data in ['en', 'uk'])
def language_choice(call):
    bd.set_language(call.message.chat.id, call.data)
    action_user_lan[call.message.chat.id] = call.data
    main_menu(call.message.chat.id,data[action_user_lan[call.message.chat.id]]["language_changed"])
```

Рисунок 3.49 – Програмна реалізації зміни мови

Перейдемо в Models. На рисунку 3.50 зображене нове меню вибору моделей. Доступні 3 моделі. Це просте розпізнавання рентгенівських зображень (X-ray), розпізнавання рентгенівських зображень з використання методу пригнічення кісткової тканини (X-ray BBS) та сам метод пригнічення кісткової тканини (BSS).

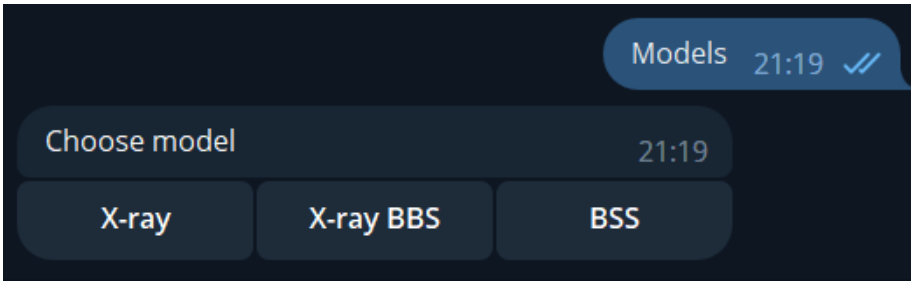


Рисунок 3.50 – Вигляд меню з доступними моделями

На рисунку 3.51 зображений код меню вибору моделей. Кнопки мають параметри callback_data.

```
# меню з кнопками вибору моделі
def model_menu(user_id):
    lan = user_lan(user_id)
    keyboard = telebot.types.InlineKeyboardMarkup()
    xray_button = telebot.types.InlineKeyboardButton(data[lan]['model_xray'], callback_data='xray')
    xray_bss_button = telebot.types.InlineKeyboardButton(data[lan]['model_xray_bone_shadow'], callback_data='xray_bss')
    bss_button = telebot.types.InlineKeyboardButton(data[lan]['model_bone_shadow'], callback_data='bss')
    keyboard.add(xray_button, xray_bss_button, bss_button)
    bot.send_message(user_id, text=data[lan]['choose_model'], reply_markup=keyboard)
```

Рисунок 3.51 – Програмна реалізація меню моделей

На рисунку 3.52 зображений код який при виборі однієї з моделі тригериться в передаванні дані в call.data, які відповідає за ці моделі.

```python
# обробник вибору моделі
@bot.callback_query_handler(func=lambda call: call.data in ['xray', 'xray_bss', 'bss'])
def model_choice_menu_all(call):
    one_of_models(call.message.chat.id, call.data)
```

Рисунок 3.52 – Програмна реалізація тригерування на параметри меню моделей

Код функції наведений на рисунку 3.52. Ця функція відповідає за створення динамічного меню.

```python
# меню для 1 з модель
def one_of_models(user_id, model):
    lan = user_lan(user_id)
    keyboard = telebot.types.InlineKeyboardMarkup()
    print(f'{model}_model')
    model_button = telebot.types.InlineKeyboardButton(data[lan]['classify_model'], callback_data= f'{model}_model')
    description_button = telebot.types.InlineKeyboardButton(data[lan]['model_description'], callback_data=f'{model}_description')
    back_button = telebot.types.InlineKeyboardButton(data[lan]['back'], callback_data='back_model')
    keyboard.add(model_button, description_button, back_button)
    bot.send_message(user_id, data[lan]['choose_option'], reply_markup=keyboard)
```

Рисунок 3.53 – Програмна реалізація динамічного меню моделей

Для прикладу візьмем Bss модель. На рисунку 3.54 видно, що незалежно від вибору, створиться меню з 3 кнопок. Це модель, Опис моделі та Назад. Це все звичайно на обраній мові користувача.

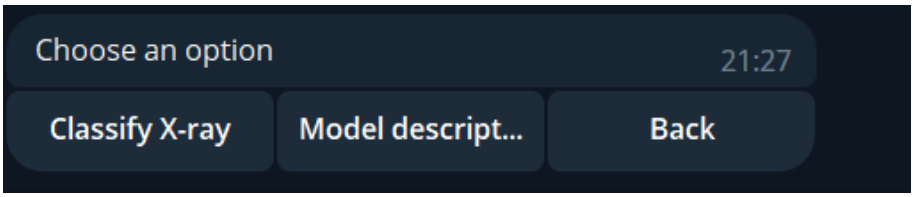


Рисунок 3.54 – Вигляд меню моделі Bss

Якщо натиснути на 1 кнопку, то отримаємо відповідь, яка зображена на рисунку 3.55. Бот просить надіслати зображення.

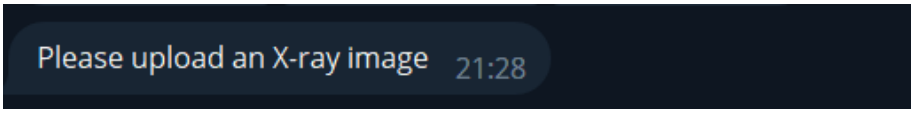


Рисунок 3.55 – Повідомлення користувачу

Надсилаємо зображення. За лічені секунди отримуємо відповідь. Це зображення з придушенням кісткової тканини (рис. 3.56).

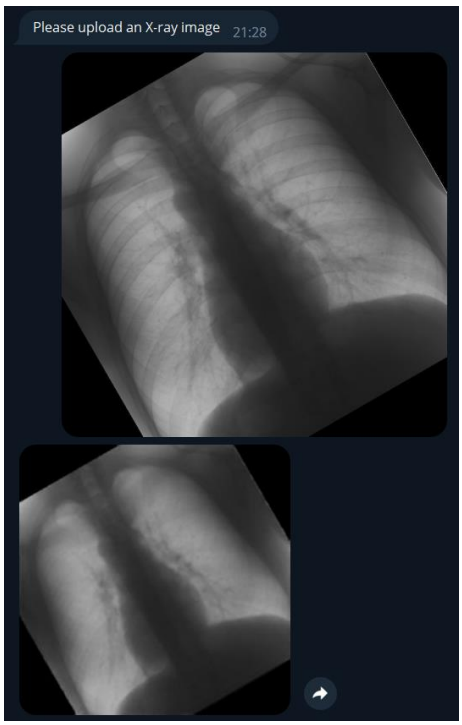


Рисунок 3.56 – Отримання відповіді від бота

Хоч зображення і передалося, але це стиснений варіант переданого зображення, повністю через файл. Для цього при надсиланні прибираємо галочку на стисканні зображення (рис. 3.57).

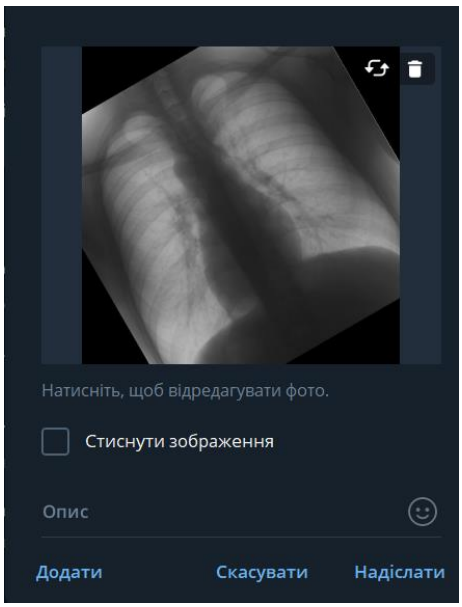
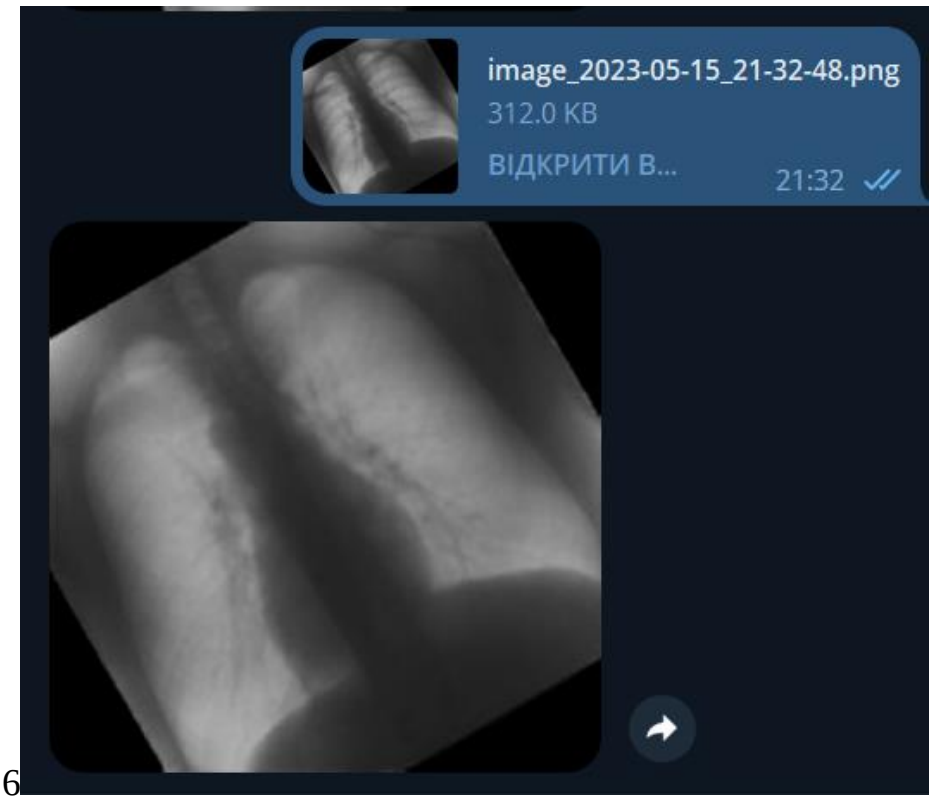


Рисунок 3.57 – Демонстрація відправки зображення без стиснення

На рисунку 3.58 зображений механізм отримання відповіді, коли відправляються зображення без стискання файлом.



6

Рисунок 3.58 – Демонстрація відправки зображення без стиснення

На рисунку 3.59 зображений коду обробки та відправлення відповіді користувачу. Спочатку йде перевірка як користувач надіслав зображення. Від цього залежить як нам його дістати. Далі йде запис шляху зображення у базу даних. Завантаження файлу відбувається шлях передачі файлу без його збереження на пристрій, адже він зберігається на сервері Telegram.

```python
@bot.message_handler(content_types=["document","photo"])
def handle_photo(message):

    if message.chat.id in action_user:

        file_info = None
        if message.photo is not None:
            file_info = bot.get_file(message.photo[-1].file_id)
        elif message.document is not None:
            file_info = bot.get_file(message.document.file_id)
        file_url = f'{file_info.file_path}'

        bd.set_image_path(message.chat.id, file_url)
        downloaded_file = bot.download_file(file_url)
        lan = user_lan(message.chat.id)

        if action_user[message.chat.id] == 'bss_model':
            bss_choise(downloaded_file, file_url)
            bot.send_photo(message.chat.id, open(file_url, 'rb'))
```

Рисунок 3.59 – Програмна реалізація отримання та взаємодії з файлами

Якщо це модель придушення кісткової тканини, то спрацьовує певний if, який викликає функцію bss_choise, яка передає параметр зображення та його шлях. Функція зображена на рисунку 3.60.

```python
def bss_choise(images, file_url):
    im_np = read_image(images)
    pred = pred_bss(im_np)
    save_bss(pred, file_url)
```

Рисунок 3.60 – Програмна реалізація функції bss_models

З коду видно, що спочатку зображення передається функції read_image (рис. 3.60), де змінюють розмір та нормалізують зображення.

```python
def read_image(images):
    im = cv2.imdecode(np.frombuffer(images, dtype=np.uint8), cv2.IMREAD_GRAYSCALE)
    im = cv2.resize(im, (256, 256))[:, :]
    im = im / 255
    return np.array(im).reshape(-1, 256, 256, 1)
```

Рисунок 3.61 – Програмна реалізація функції read_image

Далі зображення передається функції pred_bss (рис. 3.62), яка повертає предикт зображення. Також є 2 подібні функції, для решти моделей.

```
def pred_bss(im_np):
    return bss.predict(im_np)
```

Рисунок 3.62 – Програмна реалізація функції pred_bss

Далі зображення зберігається і передається користувачу (рис.3.57–3.58).

Оберемо іншу модель, для цього використаємо кнопку Back (Назад). Зміни зображені на рисунку 3.63. Меню обраної моделі нічим не відрізняється від попередньої.

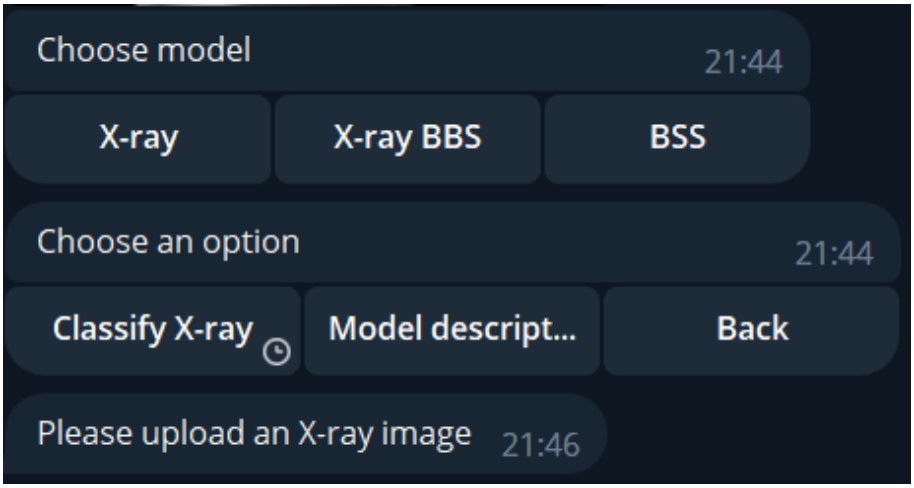


Рисунок 3.63 – Зміна моделі

Завантажимо зображення здорової людини.

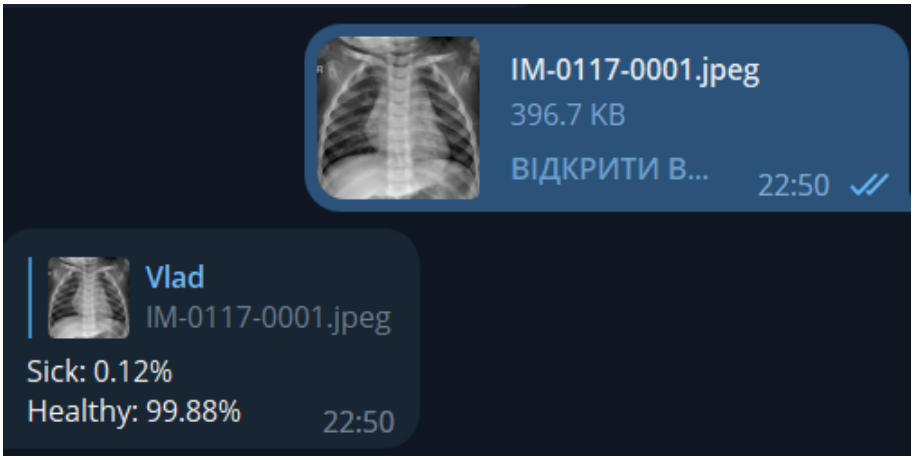


Рисунок 3.64 – Отримання відповіді з ймовірністю для зображення без хвороб

На рисунку 3.64 зображена отримана відповідь, в якій сказано, що існує ймовірність того, що дане рентгенівське зображення належить здоровому пацієнту з відсотком 98.76%.

Завантажимо зображення хворого пацієнта.

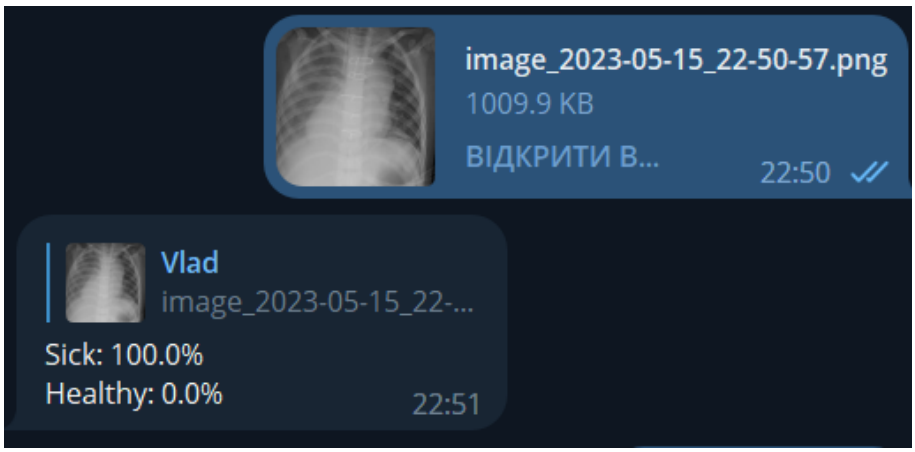


Рисунок 3.65 – Отримання відповіді з ймовірністю для зображення з хворобою

На рисунку 3.65 зображено отримання відповіді, в якій сказано, що існує ймовірність у 100%, що на даному зображенні, яке скинув користувач є хвороба.

Завантажимо ці зображення у модель розпізнавання рентгенівських зображень з використанням методу пригніченням кісткової тканини (3 модель). Для цього повернемось в меню до моделей та виберемо потрібно. Для перевірки використає ті самі зображення. Відповідь бота зображено на рисунку 3.66.

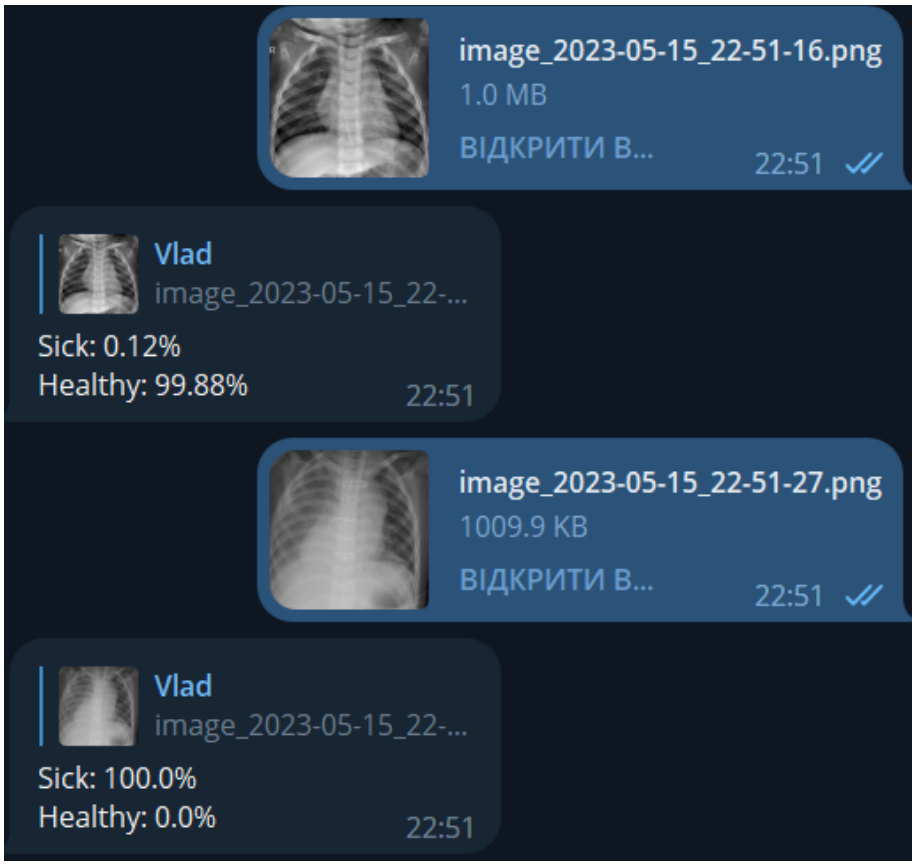


Рисунок 3.66 – Отримання відповідей з використання моделі з пригніченням

З рисунку 3.66 видно, що ця модель розпізнала зображення так само. Можливо причина в тому, що для демонстрації можливостей бота були використані зображення з тестового набору і це зображення відноситься до точно прогнозованого.

На рисунку 3.67 код виклику прогнозуючого коду та відправки відповідей користувачу. Цей код є продовження коду з рисунка 3.59. З коду видно, що ці дві моделі мають однакову відповідь користувачу. Але існує різниця.

Рисунок 3.67 – Програмна реалізація виклику прогнозуючого коду та відправки відповідей користувачу

На рисунку 3.68 зображений код викликаючих функцій x_ray_bss_choise та x_ray_choise. Вони відрізняються в додатковому використанні виклику моделі придушення кісткової тканини для пре обробки відправленого користувачем зображення.

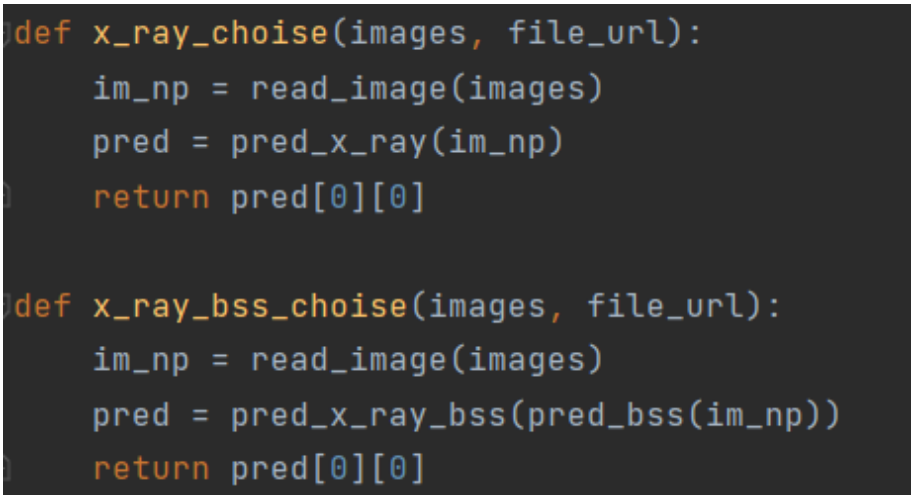


Рисунок 3.68 – Програмна реалізація функції для 2 моделей

```
bss = bss_model()
bss.load_weights('model/bss_model.h5')

x_ray = x_ray_model()
x_ray.load_weights('model/x_ray.hdf5')

x_ray_bss = x_ray_model()
x_ray_bss.load_weights('model/x_ray_bss.hdf5')
```

Рисунок 3.69 – Програмна реалізація функції для 2 моделей

А самі моделі були ініційовані за допомогою коду зображеному на рисунку 3.69. Де останні моделі мають однакову основу, але завантажені різні ваги з минулого розділу. Код моделей було описано в попередньому розділі.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи була створена інформаційна система для розпізнання рентгенівських зображень. Розглянуті доступні можливості розробки зазначеної системи та обрані оптимальні інструменти для виконання поставленого завдання. Під час розробки проекту були враховано усі вимоги до програмної реалізації системи.

У ході виконання кваліфікаційної роботи магістра

- модернізована та натренована модель пригнічення кісткової тканини для придушення кістковою тканиною на рентгенівських зображеннях;
- адаптовані та натреновані моделі розпізнавання рентгенівських зображень для класифікації зображень залежно від наявності патологій;
- розроблений чат-бот Telegram, який дозволяє користувачам надсилати рентгенівські зображення та отримувати результати розпізнавання.

Для цього були виконані такі завдання:

1. Проведений аналіз предметної області та визначено актуальність розробки даної системи.
2. Обрані мови та бібліотеки для розробки програмного продукту.
3. Реалізовано роботу інформаційної системи, що дозволяє отримувати та відправляти результати класифікації рентгенівських зображень.

За результатами роботи інформаційної системи можна зробити висновок, що застосування розробленої системи збільшить точність та ефективність розпізнання рентгенівських зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Renfrew D.L. et al. Error in radiology: Classification and lessons in 182 cases presented at a problem case conference // Radiology. 1992. Vol. 183, № 1. P. 145–150.
2. Bruno M.A., Walker E.A., Abujudeh H.H. Understanding and confronting our mistakes: The epidemiology of error in radiology and strategies for error reduction // Radiographics. Radiological Society of North America Inc., 2015. Vol. 35, № 6. P. 1668–1676.
3. Brook O.R. et al. Anatomy and pathophysiology of errors occurring in clinical radiology practice // Radiographics. 2010. Vol. 30, № 5. P. 1401–1410.
4. Satia I. et al. Assessing the accuracy and certainty in interpreting chest X-rays in the medical division // Clin. Med. (Northfield. Il). Royal College of Physicians, 2013. Vol. 13, № 4. P. 349.
5. Gordienko Y. et al. Deep Learning with Lung Segmentation and Bone Shadow Exclusion Techniques for Chest X-Ray Analysis of Lung Cancer // Adv. Intell. Syst. Comput. Springer Verlag, 2017. Vol. 754. P. 638–647.
6. Telegram Messenger [Electronic resource].
7. Telegram APIs [Electronic resource].
8. Balaji K., Lavanya K. Medical Image Analysis With Deep Neural Networks // Deep Learn. Parallel Comput. Environ. Bioeng. Syst. Elsevier, 2019. P. 75–97.
9. Yamashita R. et al. Convolutional neural networks: an overview and application in radiology // Insights Imaging. Springer Verlag, 2018. Vol. 9, № 4. P. 611–629.
10. Welcome to Python.org [Electronic resource].
11. TensorFlow [Electronic resource].
12. Scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation [Electronic resource].
13. Home - OpenCV [Electronic resource].
14. NumPy [Electronic resource].
15. Matplotlib — Visualization with Python [Electronic resource].

16. Waskom M. seaborn: statistical data visualization // J. Open Source Softw. The Open Journal, 2021. Vol. 6, № 60. P. 3021.
17. Project Jupyter | Home [Electronic resource].
18. Download PyCharm: Python IDE for Professional Developers by JetBrains [Electronic resource].
19. DB Browser for SQLite [Electronic resource].
20. Rajaraman S. et al. Chest X-ray Bone Suppression for Improving Classification of Tuberculosis-Consistent Findings // Diagnostics 2021, Vol. 11, Page 840. Multidisciplinary Digital Publishing Institute, 2021. Vol. 11, № 5. P. 840.
21. Kermany D.S. et al. Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning // Cell. Cell Press, 2018. Vol. 172, № 5. P. 1122-1131.e9.
22. Tf.keras.preprocessing.image.ImageDataGenerator | TensorFlow v2.12.0 [Electronic resource].
23. Видриган В. О. Шовкопляс О.А. Класифікація рентгенівських знімків з використанням алгоритмів глибокого навчання // Інформатика, математика, автоматика (IMA – 2023) : матеріали та програма міжнародної науково-технічної конференції, м. Суми, 24–28 квітня 2023 р. Суми : СумДУ, 2023. С. 120.

ДОДАТКИ

Додаток А. Лістинг програмного коду

Модель пригнічення кісткової тканини:

```
import os
import numpy as np
import pandas as pd
import cv2

import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow.keras
import tensorflow.keras.losses

from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras import backend as K
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint,
LearningRateScheduler, EarlyStopping, ReduceLROnPlateau



from sklearn.model_selection import train_test_split
from sklearn.metrics import *

image_path = '../X-ray Bone Shadow Supression/augmented/augmented/source'
target_path= '../X-ray Bone Shadow Supression/augmented/augmented/target'

def getData():
    im_with_array = []
    im_without_array = []
```

```
    for i in image_list:
        im_with                                                    =
cv2.imread(os.path.join(image_path,i),cv2.IMREAD_GRAYSCALE)
        im_with = cv2.resize(im_with, (256, 256))[:, :]
        im_with = im_with/255

        im_without         =         cv2.imread(os.path.join(target_path,
i),cv2.IMREAD_GRAYSCALE)
        im_without = cv2.resize(im_without, (256, 256))[:, :]
        im_without = im_without/255



        im_with_array.append(im_with)
        im_without_array.append(im_without)

    return im_with_array, im_without_array

image_array, target_array = getData()

image_array = np.array(image_array).reshape(-1, 256, 256, 1)
target_array = np.array(target_array).reshape(-1, 256, 256, 1)

X_train,           X_val,           Y_train,           Y_val           =
train_test_split(image_array,target_array,
                                                test_size = 0.2,
                                                random_state = 42)
X_val, X_test, Y_val,Y_test = train_test_split(X_val,Y_val,
                                                test_size = 0.1,
                                                random_state = 42)
print(X_train.shape,              X_val.shape,X_test.shape,Y_train.shape,
Y_val.shape,Y_test.shape)

def tf_log10(x):
    numerator = tf.math.log(x)
```

```
    denominator = tf.math.log(tf.constant(10, dtype=numerator.dtype))
    return numerator / denominator

def psnr(y_true, y_pred):
    max_pixel = 1.0
    return 10.0 * tf_log10((max_pixel ** 2) / (K.mean(K.square(y_pred -
y_true))))

def mae(y_true, y_pred):
    return tf.keras.metrics.mean_absolute_error(y_true, y_pred)

def ssim(y_true, y_pred):
    return tf.reduce_mean(tf.image.ssim(y_true, y_pred, 1.0))

def ssim_loss(y_true, y_pred):
    return 1-tf.reduce_mean(tf.image.ssim(y_true, y_pred, 1.0))

def ssim_multi(y_true, y_pred):
    return tf.reduce_mean(tf.image.ssim_multiscale(y_true, y_pred, 1.0))

def ssim_multi_loss(y_true, y_pred):
    return   1-tf.reduce_mean(tf.image.ssim_multiscale(y_true,   y_pred,
1.0))

def loss_mix_multi_084(y_true, y_pred):
    return 0.16 * mae(y_true, y_pred) + \
            0.84 * (1-ssim_multi(y_true, y_pred))

def BSS():

  input = Input((256,256,1))

  #encoder
  conv1 = Conv2D(16, (5, 5), activation='relu', padding='same')(input)
```

```
  conv2 = Conv2D(32, (5, 5), activation='relu', padding='same',
strides=2)(conv1)
  conv3 = Conv2D(64, (5, 5), activation='relu', padding='same',
strides=2)(conv2)
  #decoder
  conv4 = Conv2D(32, (5, 5), activation='relu', padding='same')(conv3)
  up1 = UpSampling2D((2,2))(conv4)
  conv5 = Conv2D(16, (5, 5), activation='relu', padding='same')(up1)
  up2 = UpSampling2D((2,2))(conv5)
  decoded = Conv2D(1, (5, 5), activation='sigmoid', padding='same')(up2)

  return  Model(input, decoded)

model = BSS()
model.compile(optimizer=Adam(lr=0.001), loss=loss_mix_multi_084,
              metrics=[mae, ssim_multi_loss,
                       psnr, ssim, ssim_multi])
model.summary()

weight_path="bss.best.hdf5"

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss',
                             verbose=1, save_weights_only=True,
                             save_best_only=True, mode='min')
earlyStopping = EarlyStopping(monitor='val_loss',
                              patience=10, verbose=1, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10,
                              verbose=1, mode='min', min_lr=0.00001)
callbacks_list = [checkpoint, earlyStopping, reduce_lr]

h = model.fit(X_train, Y_train,
            validation_data=(X_val, Y_val),
            epochs=200,
            callbacks= callbacks_list)
```

```
hist_df = pd.DataFrame(h.history)
hist_csv_file = 'history-bss-model.csv'
with open(hist_csv_file, mode='ab') as f:
    hist_df.to_csv(f)

model.load_weights('bss.best.hdf5')
model.summary()

preds = bss.predict(X_test)

fig, axs = plt.subplots(nrows=10, ncols=3, figsize=(10, 20))

for i in range(10):
    for j in range(3):
        if j == 1:
            axs[i, j].imshow(np.squeeze(X_test[i]), cmap='gray')
            axs[i, j].set_title('Original')
        elif j==2:
            axs[i, j].imshow(np.squeeze(Y_test[i]), cmap='gray')
            axs[i, j].set_title('Target')
        else:
            axs[i, j].imshow(np.squeeze(preds[i]), cmap='gray')
            axs[i, j].set_title('predicted')

model.save("bss-model")

model.save("bss-model.h5")

fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(h.history['loss'], '-', label = 'Loss')
ax1.plot(h.history['val_loss'], '-', label = 'Validation Loss')
ax1.legend()
```

```
fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(h.history['ssim'], '-', label = 'ssim')
ax1.legend()

fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(h.history['mae'], '-', label = 'mae')
ax1.legend()

fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(h.history['PSNR'], '-', label = 'PSNR')
ax1.legend()
```

Моделі розпізнання рентгенівських зображень:

```
import os
import numpy as np
import pandas as pd
import cv2

import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
import tensorflow.keras.losses

from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras import backend as K
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ModelCheckpoint,
LearningRateScheduler, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import *

plt.style.use('fivethirtyeight')
%matplotlib inline

def BSS():

  input = Input((256,256,1))

  #encoder
  conv1 = Conv2D(16, (5, 5), activation='relu', padding='same')(input)
  conv2  =  Conv2D(32,  (5,  5),  activation='relu',  padding='same',
strides=2)(conv1)
  conv3  =  Conv2D(64,  (5,  5),  activation='relu',  padding='same',
strides=2)(conv2)
  #decoder
  conv4 = Conv2D(32, (5, 5), activation='relu', padding='same')(conv3)
  up1 = UpSampling2D((2,2))(conv4)
  conv5 = Conv2D(16, (5, 5), activation='relu', padding='same')(up1)
  up2 = UpSampling2D((2,2))(conv5)
  decoded = Conv2D(1, (5, 5), activation='sigmoid', padding='same')(up2)

  return  Model(input, decoded)

bss = BSS()

bss.load_weights('bss-model.h5')

labels = ['PNEUMONIA', 'NORMAL']
def getData(data_dir):
    im_array = []
    target_array = []
    for label in labels:
        path = os.path.join(data_dir, label)
```

```
        target_numer = labels.index(label)
        for img in os.listdir(path):
            image_array = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
            image_array = cv2.resize(image_array, (256, 256))
            image_array = image_array / 255

            im_array.append(image_array)
            target_array.append(target_numer)
    return im_array, target_array


image_array_train, target_array_train = getData('chest_xray/train')
image_array_test, target_array_test = getData('chest_xray/test')
image_array_val, target_array_val = getData('chest_xray/val')
X = []
y = []

for feature, label in zip(image_array_train, target_array_train):
    X.append(feature)
    y.append(label)

for feature, label in zip(image_array_test, target_array_test):
    X.append(feature)
    y.append(label)

for feature, label in zip(image_array_val, target_array_val):
    X.append(feature)
    y.append(label)

X = np.array(X).reshape(-1, 256, 256, 1)
Y = np.array(y)

#Залежить чи є використання моделі придушення кісткової тканини
```

```
#X = bss.predict(X)

X_train, X_val, Y_train, Y_val = train_test_split(X, y,
                                                  test_size=0.2,
                                                  random_state=42)
X_train, X_test, Y_val, Y_val = train_test_split(X_train, y_train,
                                                 test_size=0.20,
                                                 random_state=42)
print(X_train.shape,              X_val.shape,X_test.shape,Y_train.shape,
Y_val.shape,Y_test.shape)

datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=90,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        vertical_flip=True)

datagen.fit(X_train)

model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=(256,256,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(64, (3, 3), padding='same'))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(BatchNormalization(axis=1))

model.add(Flatten())

model.add(Dropout(0.2))
model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer=Adam(learning_rate=0.0
001),metrics=['accuracy'])

weight_path="cnn-bss.best.hdf5"

checkpoint = ModelCheckpoint(weight_path, monitor='val_loss',
                             verbose=1, save_weights_only=True,
                             save_best_only=True, mode='min')
earlyStopping = EarlyStopping(monitor='val_loss',
                              patience=3, verbose=1, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
                              verbose=1, mode='min', min_lr=0.00001)

callbacks_list = [checkpoint, earlyStopping, reduce_lr]
```

```
h = model.fit(X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=20,
            callbacks= callbacks_list)

model.evaluate(X_test, Y_test)

fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(h.history['loss'], '-', label = 'Loss')
ax1.plot(h.history['val_loss'], '-', label = 'Validation Loss')
ax1.legend()

fig, (ax1) = plt.subplots(1, 1, figsize = (10, 10))
ax1.plot(history.history['accuracy'], '-', label = 'Accuracy')
ax1.plot(history.history['val_accuracy'],  '-',  label  =  'Validation
Accuracy')
ax1.legend()
pred = model.predict(X_train)
precisions, recalls, thresholds = precision_recall_curve(Y_train, pred)
fpr, tpr, thresholds2 = roc_curve(Y_train, pred)

def plot_precision_recall(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1])
    plt.plot(thresholds, recalls[:-1])
    plt.title('Precision vs. Recall')
    plt.xlabel('Thresholds')
    plt.legend(['Precision', 'Recall'], loc='best')
    plt.show()

def plot_roc(fpr, tpr):
    plt.plot(fpr, tpr)
    plt.plot([0, 1], [0, 1])
    plt.title('FPR (False Positive rate) vs TPR (True Positive Rate)')
```

```
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate (Recall)')
    plt.show()

plot_precision_recall(precisions, recalls, thresholds)
plot_roc(fpr, tpr)

predictions = model.predict(X_test)
predictions

binary_predictions = []
threshold = thresholds[np.argmax(precisions >= 0.80)]
for i in predictions:
    if i >= threshold:
        binary_predictions.append(1)
    else:
        binary_predictions.append(0)

print('Accuracy on testing set:', accuracy_score(binary_predictions,
y_test))
print('Precision on testing set:', precision_score(binary_predictions,
y_test))
print('Recall on testing set:', recall_score(binary_predictions, y_test))
print('F1-score on testing set:', f1_score(binary_predictions, y_test))

matrix = confusion_matrix(binary_predictions, y_test)
plt.figure(figsize=(16, 9))
ax= plt.subplot()
sns.heatmap(matrix, cmap= "Blues", linecolor = 'black', annot=True, ax =
ax)

# labels, title and ticks
ax.set_xlabel('Predicted Labels', size=20)
ax.set_ylabel('True Labels', size=20)
```

```
ax.set_title('Confusion Matrix', size=20)
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train.reshape(-1, img_size, img_size)[i], cmap='gray')
    if(binary_predictions[i]==y_test[i]):
        plt.xlabel(labels[binary_predictions[i]], color='blue')
    else:
        plt.xlabel(labels[binary_predictions[i]], color='red')
plt.show()
```

Телеграм бот

Main.py :

```
import telebot
from database import *
from models import *
import json

data = json.loads(open('lan.json', 'r', encoding='utf-8').read())
bd = SQLighter()
action_user = {}
action_user_lan = {}

# створення змінних для назв кнопок та повідомлень для кожної мови
TOKEN = '6231932551:AAGunuHj-msOj9YJOPUYkYWerMWMHCtQbzs'
bot = telebot.TeleBot('6231932551:AAGunuHj-msOj9YJOPUYkYWerMWMHCtQbzs')
```

```
def user_lan(user_id):
    if user_id in action_user_lan:
        return action_user_lan[user_id]
    else:
        lan = bd.get_language(user_id)
        action_user_lan[user_id] = lan
        return lan


# основне меню з кнопками "Модель" та "Параметри"
def main_menu(user_id, text = None):
    lan = user_lan(user_id)
    keyboard = telebot.types.ReplyKeyboardMarkup(resize_keyboard=True)
    model_button = telebot.types.KeyboardButton(data[lan]['models'])
    params_button = telebot.types.KeyboardButton(data[lan]['parameters'])
    keyboard.add(model_button, params_button)
    bot.send_message(user_id, text=text, reply_markup=keyboard)



# меню з кнопками вибору моделі
def model_menu(user_id):
    lan = user_lan(user_id)
    keyboard = telebot.types.InlineKeyboardMarkup()
    xray_button                                                    =
telebot.types.InlineKeyboardButton(data[lan]['model_xray'],
callback_data='xray')
    xray_bss_button                                                =
telebot.types.InlineKeyboardButton(data[lan]['model_xray_bone_shadow'],
callback_data='xray_bss')
    bss_button                                                     =
telebot.types.InlineKeyboardButton(data[lan]['model_bone_shadow'],
callback_data='bss')
    keyboard.add(xray_button, xray_bss_button, bss_button)
```

```
    bot.send_message(user_id,             text=data[lan]['choose_model'],
reply_markup=keyboard)

# меню для 1 з модель
def one_of_models(user_id, model):
    lan = user_lan(user_id)
    keyboard = telebot.types.InlineKeyboardMarkup()
    print(f'{model}_model')
    model_button                                                    =
telebot.types.InlineKeyboardButton(data[lan]['classify_model'],
callback_data= f'{model}_model')
    description_button                                              =
telebot.types.InlineKeyboardButton(data[lan]['model_description'],
callback_data=f'{model}_description')
    back_button  =  telebot.types.InlineKeyboardButton(data[lan]['back'],
callback_data='back_model')
    keyboard.add(model_button, description_button, back_button)
    bot.send_message(user_id,                 data[lan]['choose_option'],
reply_markup=keyboard)


# обробник команди /start
@bot.message_handler(commands=['start'])
def start(message):
    language = message.from_user.language_code
    if language == 'uk':
        bd.set_language(message.chat.id, 'uk')
        action_user_lan[message.chat.id] = 'uk'
    else:
        bd.set_language(message.chat.id, 'en')
        action_user_lan[message.chat.id] = 'en'

    main_menu(message.chat.id, data[language]["bot_description"])
```

```
# обробник кнопки "Моделі"
@bot.message_handler(func=lambda message: message.text == 'Models' or
message.text == 'Моделі')
def model_menu_all(message):
    model_menu(message.chat.id)


# обробник кнопки "Параметри"
@bot.message_handler(func=lambda message: message.text == 'Parameters' or
message.text == 'Параметри')
def parameters(message):
    # меню з кнопками вибору мови
    lan = user_lan(message.chat.id)
    language_menu = telebot.types.InlineKeyboardMarkup()
    en_button = telebot.types.InlineKeyboardButton('English',
callback_data='en')
    ua_button = telebot.types.InlineKeyboardButton('Українська',
callback_data='uk')
    language_menu.add(en_button, ua_button)
    bot.send_message(message.chat.id, data[lan]['choose_language'],
reply_markup=language_menu)


# обробник вибору мови
@bot.callback_query_handler(func=lambda call: call.data in ['en', 'uk'])
def language_choice(call):
    bd.set_language(call.message.chat.id, call.data)
    action_user_lan[call.message.chat.id] = call.data

main_menu(call.message.chat.id,data[action_user_lan[call.message.chat.id]
]["language_changed"])
```

```
# обробник кнопки "Назад"
@bot.callback_query_handler(func=lambda call: call.data == 'back_model')
def model_choice(call):
    model_menu(call.message.chat.id)


# обробник вибору моделі
@bot.callback_query_handler(func=lambda call: call.data in ['xray',
'xray_bss', 'bss'])
def model_choice_menu_all(call):
    one_of_models(call.message.chat.id, call.data)


# обробник вибору моделі або перегляду її опису моделі
@bot.callback_query_handler(func=lambda call: call.data in ['xray_model',
'xray_bss_model','bss_model'])
def model_choice_button(call):
    lan = user_lan(call.message.chat.id)
    action_user[call.message.chat.id] = call.data
    bot.send_message(call.message.chat.id, data[lan]['please'])

# обробник вибору моделі або перегляду її опису моделі
@bot.callback_query_handler(func=lambda call: call.data in
['xray_description', 'xray_bss_description','bss_description'])
def model_choice_description(call):
    pass


@bot.message_handler(content_types=["document","photo"])
def handle_photo(message):

    if message.chat.id in action_user:
```

```
        file_info = None
        if message.photo is not None:
            file_info = bot.get_file(message.photo[-1].file_id)
        elif message.document is not None:
            file_info = bot.get_file(message.document.file_id)
        file_url = f'{file_info.file_path}'

        bd.set_image_path(message.chat.id, file_url)
        downloaded_file = bot.download_file(file_url)
        lan = user_lan(message.chat.id)

        if action_user[message.chat.id] == 'bss_model':
            bss_choise(downloaded_file, file_url)
            bot.send_photo(message.chat.id, open(file_url, 'rb'))

        elif action_user[message.chat.id] == 'xray_bss_model':
            pred = x_ray_bss_choise(downloaded_file, file_url)
            bot.reply_to(message, f"{data[lan]['sick']} {round(100 - pred
* 100,2)}% \n{data[lan]['healthy']} {round(pred * 100,2)}%")

        elif action_user[message.chat.id] == 'xray_model':
            pred = x_ray_choise(downloaded_file, file_url)
            bot.reply_to(message, f"{data[lan]['sick']} {round(100 -  pred
* 100,2)}% \n{data[lan]['healthy']} {round(pred * 100,2)}%")

    else:
        bot.reply_to(message.chat.id,  data[lan]['sick']['please_model'],
reply_markup=model_menu)


bot.polling()
```

models.py :

```
import numpy as np
import os
import random
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *

def bss_model():
    input = Input((256, 256, 1))

    conv1 = Conv2D(16, (5, 5), activation='relu', padding='same')(input)
    conv2 = Conv2D(32, (5, 5), activation='relu', padding='same',
strides=2)(conv1)
    conv3 = Conv2D(64, (5, 5), activation='relu', padding='same',
strides=2)(conv2)

    conv4 = Conv2D(32, (5, 5), activation='relu', padding='same')(conv3)
    up1 = UpSampling2D((2, 2))(conv4)
    conv5 = Conv2D(16, (5, 5), activation='relu', padding='same')(up1)
    up2 = UpSampling2D((2, 2))(conv5)
    decoded = Conv2D(1, (5, 5), activation='sigmoid', padding='same')(up2)

    return Model(input, decoded)

def x_ray_model():
    model = Sequential()

    model.add(Conv2D(256, (3, 3), input_shape=(256, 256, 1),
padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
```

```
    model.add(BatchNormalization(axis=1))

    model.add(Conv2D(64, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
    model.add(BatchNormalization(axis=1))

    model.add(Conv2D(16, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
    model.add(BatchNormalization(axis=1))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(64))
    model.add(Activation('relu'))

    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    return model

def read_image(images):
    im     =     cv2.imdecode(np.frombuffer(images,     dtype=np.uint8),
cv2.IMREAD_GRAYSCALE)
    im = cv2.resize(im, (256, 256))[:, :]
    im = im / 255
    return np.array(im).reshape(-1, 256, 256, 1)

def pred_bss(im_np):
    return bss.predict(im_np)

def pred_x_ray(im_np):
```

```
    return x_ray.predict(im_np)

def pred_x_ray_bss(im_np):
    return x_ray_bss.predict(im_np)

def save_bss(im_np,file_url):
    cv2.imwrite(file_url, im_np[0] * 255)

def bss_choise(images, file_url):
    im_np = read_image(images)
    pred = pred_bss(im_np)
    save_bss(pred, file_url)

def x_ray_choise(images, file_url):
    im_np = read_image(images)
    pred = pred_x_ray(im_np)
    return pred[0][0]

def x_ray_bss_choise(images, file_url):
    im_np = read_image(images)
    pred = pred_x_ray_bss(pred_bss(im_np))
    return pred[0][0]

bss = bss_model()
bss.load_weights('model/bss_model.h5')

x_ray = x_ray_model()
x_ray.load_weights('model/x_ray.hdf5')

x_ray_bss = x_ray_model()
x_ray_bss.load_weights('model/x_ray_bss.hdf5')
```

database.py :

```
import sqlite3
```

```
class SQLighter:

    def __init__(self):
        self.connection = sqlite3.connect('bot.db',
check_same_thread=False)
        self.cursor = self.connection.cursor()

        self.cursor.execute("""CREATE TABLE IF NOT EXISTS users
                        (user_id INTEGER PRIMARY KEY,
                         language TEXT)""")
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS images
                        (sender_id INTEGER,
                         image_path TEXT,
                         FOREIGN KEY (sender_id) REFERENCES
users(user_id))""")

    # Функція для зміни мови користувача
    def set_language(self, user_id, language):
        print( user_id, language)
        self.cursor.execute(f"""INSERT OR REPLACE INTO users (user_id,
language) VALUES ('{user_id}', '{language}')""")
        self.connection.commit()


    # Функція для отримання мови користувача
    def get_language(self, user_id):
        return self.cursor.execute(f"""SELECT language FROM users WHERE
user_id = '{user_id}'""").fetchall()[0][0]


    # Функція для збереження шляху до зображення користувача
    def set_image_path(self, sender_id, image_path):
        self.cursor.execute(f"""INSERT INTO images (sender_id, image_path)
VALUES ('{sender_id}', '{image_path}')""")
```

```
        self.connection.commit()
```

lan.json :

```
{
    "en" : {
        "bot_description": "This bot demonstrates the capabilities of neural networks for classifying X-ray images.",
        "models": "Models",
        "parameters": "Parameters",
        "model_xray": "X-ray",
        "model_xray_bone_shadow": "X-ray BBS",
        "model_bone_shadow": "BSS",
        "choose_model": "Choose model",
        "choose_option": "Choose an option",
        "classify_model": "Classify X-ray",
        "model_description": "Model description",
        "choose_language": "Choose language",
        "language_changed": "Language changed to English",
        "back" : "Back",
        "please": "Please upload an X-ray image",
        "please_model": "Please Choose a model",
        "healthy" : "Healthy:",
        "sick" : "Sick:"

    },
    "uk" : {
        "bot_description": "Цей бот демонструє можливості нейронних мереж для класифікації рентгенівських зображень.",
        "models": "Моделі",
        "parameters": "Параметри",
        "model_xray": "Рентген",
        "model_xray_bone_shadow": "Рентген з пригніченням кісткової тіні",
        "model_bone_shadow": "Пригнічення кісткової тіні",
        "choose_model": "Оберіть модель:",
        "choose_option": "Виберіть варіант",
```

```
        "classify_model": "Класифікувати",
        "model_description": "Опис моделі",
        "choose_language": "Оберіть мову",
        "language_changed": "Мову змінено на Українську",
        "back" : "Назад",
        "please": "Будь ласка, завантажте рентгенівське зображення",
        "please_model": "Будь ласка, виберіть модель",
        "healthy" : "Здоровий:",
        "sick" : "Хворий:"
    }
}
```