МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

« » травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система діагностування шкірних захворювань з використанням чат боту»

здобувачки групи КНд-01с Щербань Дар'ї Олександрівни

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

_____ Дар'я ЩЕРБАНЬ

(підпис)

Старший викладач, доктор філософії _____ Микола ЗАРЕЦЬКИЙ

(підпис)

Суми – 2024

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

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

здобувачки групи КНд-01с Щербань Дар'я Олександрівна

1. Тема роботи: «Інформаційна система діагностування шкірних захворювань з використанням чат боту»

затверджую наказом по СумДУ від *«26» квітня 2024 року № 0438-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до «5» червня 2024 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми, перегляд наявних рішень та алгоритмів. 2) Формування задачі та визначення цілі дослідження. 3) Вибір методології для розв'язання задачі. 4) Розробка та проєктування системи. 5) Здійснення практичної реалізації системи.

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

6. Консультанти до проєкту (роботи), із значенням розділів проєкту, 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 дипломного проекту (роботи)	Термін виконання	Примітка
1.	*Аналіз проблеми. Огляд аналогів розроблюваного програмного продукту. Огляд алгоритмів.*		
2.	*Формулювання завдань проєктута вибір методів реалізації.*		
3.	*Розробка моделі та створення проєкту чат-боту.*		
4.	*Практична реалізація проєкту.*		
5.	*Оформлення пояснювальної записки до дипломного проєкту.*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 66 стор., 35 рис., 4 додатків, 22 джерел.

Об'єкт дослідження — процеси діагностики шкірних захворювань, процеси детектування шкірних захворювань з використанням підходів машинного навчання

Предмет дослідження — моделі та методи діагностування шкірних захворювань з використанням алгоритмів машинного навчання.

Мета роботи — розробка та аналіз ефективності чат-боту, який дозволяє користувачам отримувати первинну діагностику шкірних захворювань на основі фотографій шкірних уражень.

Результати — були обрані ефективні підходи для розв’язання задачі; за допомогою Python та моделей машинного навчання створена система, здатна проводити діагностику шкірних захворювань на підставі знімків.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, НЕЙРОННІ МЕРЕЖІ, ЕКСТРАКТОР ОЗНАК, TELEGRAM BOT API, МАШИННЕ НАВЧАННЯ, PYTHON, ШКІРНІ ЗАХВОРЮВАННЯ.

ЗМІСТ

ВСТУП

У сучасному світі швидкого розвитку технологій значно зросла роль інформаційних систем (ІС) у медицині, зокрема в діагностиці та лікуванні захворювань. Серед широкого спектра медичних проблем особливе місце займають шкірні захворювання, що поширені серед усіх вікових груп населення і можуть бути як незначними косметичними дефектами, так і симптомами серйозних патологій. Швидка та точна діагностика таких захворювань є ключовим фактором ефективного лікування та попередження їх прогресування. Втім, не завжди існує можливість негайно звернутися до фахівця через географічні, фінансові або інші бар'єри. У такому контексті виникає потреба у розробці доступних, зручних і надійних інструментів для попередньої діагностики, що здатні надати оперативну допомогу і пораду.

Розвиток штучного інтелекту (ШІ) і машинного навчання відкриває нові перспективи у медичній діагностиці. Зокрема, створення інформаційної технології діагностування шкірних захворювань з використанням чат-боту, інтегрованого з алгоритмами ШІ, може стати революційним рішенням, яке зробить первинну діагностику широко доступною для всіх верств населення. Такий підхід дозволить не тільки зменшити навантаження на фахівців, але й забезпечити швидкий доступ до кваліфікованої медичної консультації безпосередньо з дому.

Це дослідження має на меті вивчення можливостей використання чат-ботів та штучного інтелекту в ідентифікації шкірних захворювань, оцінку наявних рішень та створення авторського прототипу такої системи. Актуальність цього проекту випливає з постійної потреби у передових медичних технологіях, які здатні забезпечити якісну діагностику та лікування, оптимізуючи час і ресурси.

Створення ефективної системи передбачає інтегрований підхід, який об'єднує не тільки розробку алгоритмів на основі штучного інтелекту, але й детальне розуміння особливостей шкірних захворювань, їх симптомів та

проявів. Тому важливою частиною роботи стане співпраця з медичними спеціалістами та дерматологами, що дозволить гарантувати точність та ефективність діагностики.

Потреба у такому дослідженні також виникає через зростання числа людей з шкірними проблемами і збільшення самодіагностики та самолікування, що не завжди призводить до бажаних результатів, а й навіть, до негативних наслідків. Розробка доступної та надійної діагностичної системи сприятиме зростанню обізнаності щодо здоров'я шкіри і заохочуватиме людей своєчасно звертатися за медичною допомогою.

В кінцевому результаті, це дослідження має на меті не тільки розробку нової інформаційної технології, але й внесення вагомого вкладу у поліпшення якості медичного обслуговування, зробивши його більш доступним, ефективним та зручним для широкого кола користувачів. Це дослідження відкриває нові перспективи для використання штучного інтелекту у медицині та покращує розуміння потенціалу інформаційних технологій у боротьбі зі складними медичними викликами.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд та аналіз проблеми діагностування шкірних захворювань

Шкіра, як найбільший орган людського організму, забезпечує критично важливий бар’єр проти екзогенних загроз, включаючи мікроорганізми (бактерії, віруси) та токсичні агенти. Дерматологічні розлади представляють собою значний спектр патологій, що вражають осіб різного віку, та можуть бути ініційовані широким діапазоном етіологічних факторів, включно з генетичною предиспозицією, життєвим стилем та екологічним впливом. Серед найчастіше діагностованих дерматологічних захворювань слід виділити акне, карциноми шкіри, себорейний кератоз, псоріаз, меланому та вітіліго

(див. рис. 1.1). Враховуючи прогресивний та рецидивуючий характер багатьох шкірних захворювань, їхній негативний вплив на фізіологічне та психоемоційне становище пацієнтів може бути значним. Отже, своєчасне та адекватне діагностичне втручання є критично необхідним для визначення оптимальних стратегій терапевтичного менеджменту та підтримання оптимального рівня здоров’я [1].

Рисунок 1.1 — Найпоширеніші типів шкірних захворювань.

Традиційні методи діагностики у дерматології зазвичай залежать від візуальної інспекції дерматологічних ознак та суб’єктивного аналізу, заснованого на клінічному досвіді. Цей метод може не включати точні, об'єктивні та кількісні параметри для оцінки, що іноді призводить до ризику постановки некоректного діагнозу навіть досвідченими дерматологами [2].

На щастя, комп’ютерне розпізнавання зображень на основі технологій ШІ відкриває перспективні можливості для ідентифікації дерматологічних захворювань. Алгоритми штучного інтелекту можна навчити за допомогою великих наборів даних зображень шкіри, щоб дізнатися закономірності, пов’язані з різними захворюваннями шкіри, що дозволяє забезпечити більшу точність у діагностиці порівняно з традиційними методами, особливо на ранніх стадіях хвороби. Додатково, завдяки детальній розробці та налаштуванню, алгоритми ШІ здатні уникнути людських упереджень, забезпечуючи більшу об’єктивність у встановленні діагнозів. Таким чином, інструменти діагностики, засновані на штучному інтелекті, пропонують рішення для подолання деяких проблем, пов’язаних із діагностикою шкірних захворювань [3, 4, 5]. Сучасні алгоритми ШІ, зокрема, техніки машинного (ML) та глибокого навчання (DL), ефективно виявляють і узагальнюють характеристики шкірних уражень, дозволяючи точно розрізняти доброякісні та злоякісні утворення. DL зазвичай демонструє високу ефективність у роботі з об’ємними даними та складними задачами, в той час як ML методи можуть бути використані в сценаріях з обмеженими даними. Дані підходи можна використовувати в системах комп’ютерної діагностики (CAD), пропонуючи точні результати класифікації для дерматологів. Крім того, для тих, хто не є дерматологом, ці системи можуть допомогти зменшити кількість помилок, спричинених обмеженим досвідом.

Інноваційні розробки в області штучного інтелекту, включно з алгоритмами машинного навчання та технологіями комп’ютерного зору, відкрили значні перспективи для обробки візуальної інформації. В

дерматології, застосування ШІ сприяє значному прогресу у вирішенні діагностичних завдань, пов’язаних із розпізнаванням шкірних патологій:

1. Збільшення діагностичної точності: ШІ ефективно проаналізовує обширні бази даних дерматологічних знімків, виявляючи специфічні особливості та мінімальні відхилення, недоступні для виявлення людиною, що сприяє підвищенню точності та консистентності постановки діагнозів.

2. Раннє виявлення захворювань: ШІ має здатність ідентифікувати на ранніх етапах рак шкіри та інші патологіїна ранніх стадіях, коли лікування є найбільш ефективним. Це допомагає реєструвати зміни, які можуть залишитися непоміченими людським оком.

3. Поліпшення доступності діагностики: інструменти ШІ можуть бути ефективно застосовані медичними спеціалістами, незалежно від їх спеціалізації в галузі дерматології, значно розширюючи доступність кваліфікованих діагностичних послуг.

4. Оптимізація швидкості та ефективності діагностики: Завдяки здатності ШІ швидко обробляти та аналізувати дерматологічні зображення, вдається скоротити час, необхідний для встановлення діагнозу, що має особливе значення в умовах високої завантаженості клінічних закладів.

5. Узгодженість: Використання ШІ сприяє стандартизації процесу діагностики, мінімізуючи ризик помилкових діагнозів та сприяючи покращенню лікувальних результатів для пацієнтів.

1.2 Огляд існуючих рішень

Завдяки постійному розвитку обчислювальних технологій та алгоритмів машинного навчання, штучний інтелект змінив підходи до аналізу дерматологічних зображень, пропонуючи нові можливості для виявлення та класифікації шкірних утворень.

Далі розглянемо рішення, засновані на штучному інтелекті:

1. DermEngine (https://www.dermengine.com/) — це інтелектуальна платформа для управління дерматологічними даними, яка використовує ШІ

для аналізу зображень шкіри та ідентифікації підозрілих змін. Платформа дозволяє дерматологам ефективно вести записи пацієнтів, здійснювати теледерматологічні консультації та отримувати автоматизовані рекомендації щодо діагнозу (див. рис. 1.1).

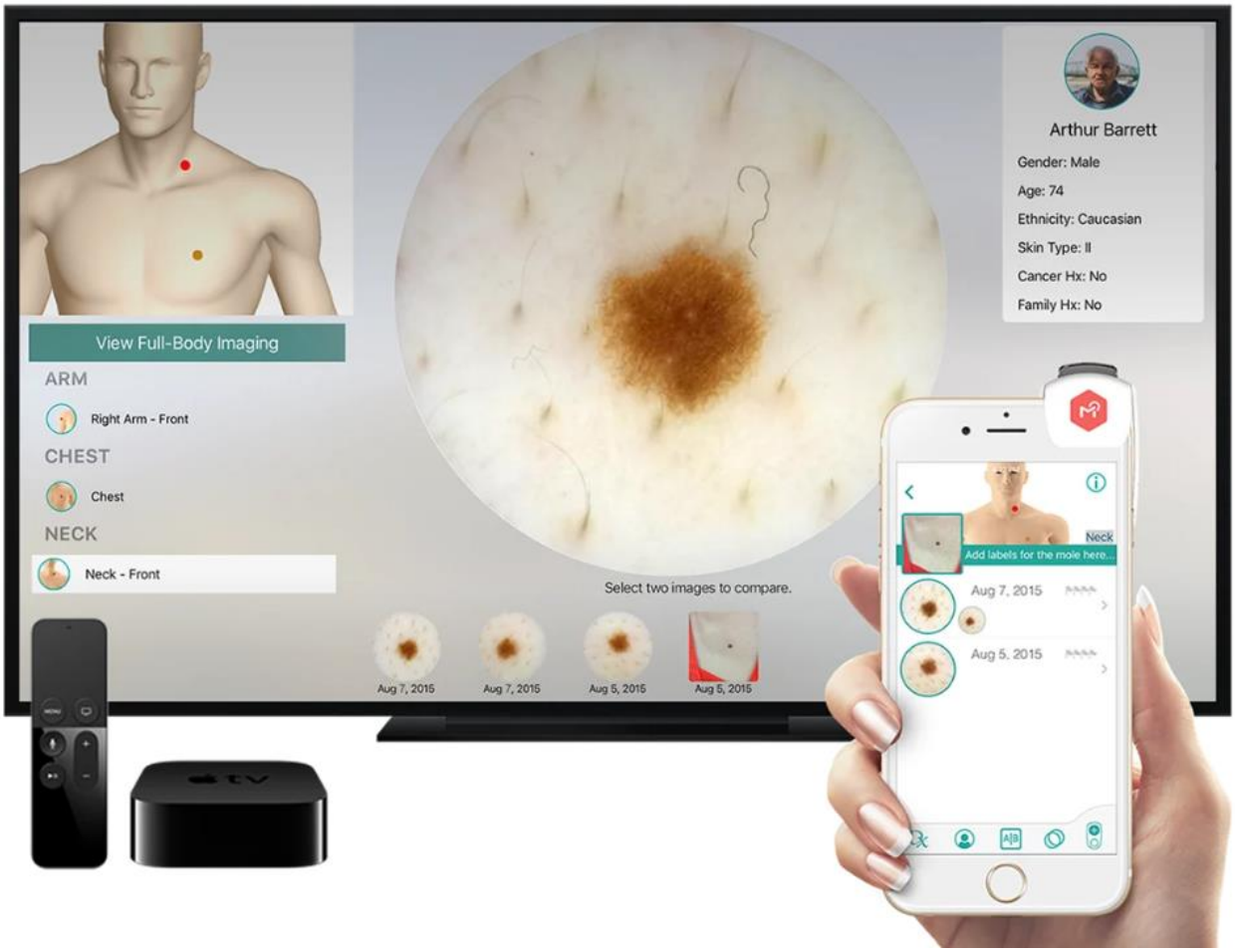

Рисунок 1.1 — Інтерфейс DermEngine

Переваги:

– розширений візуальний пошук зображень;

– автоматизовані алгоритми зіставлення родимок і відображення (зображення всього тіла);

– еволюційний трекер (дермоскопія).

Недоліки:

– вартість та доступність;

– не завжди висока точність діагностики.

2. SkinVision (https://www.skinvision.com/) — мобільний додаток, який дозволяє користувачам фотографувати шкірні ураження та використовувати алгоритми AI для оцінки ризику меланоми та інших видів раку шкіри (див. рис. 1.2). Додаток надає рекомендації щодо подальших дій, засновані на аналізі зображень.

Переваги:

– можливість раннього виявлення шкірного раку, зокрема меланоми, з високою точністю;

– додаток забезпечує легкий доступ до інструментів діагностики шкіри безпосередньо зі смартфона, що зручно для користувачів;

– надання користувачам інформації про шкірні захворювання та їх симптоми, підвищуючи обізнаність щодо стану шкіри.

Недоліки:

– обмеження за типами шкірних захворювань (можливі обмеження у здатності додатку ідентифікувати певні типи шкірних утворень або захворювань, які не включені до навчальних даних);

– вартість та доступність;

– залежність від якості зображення.

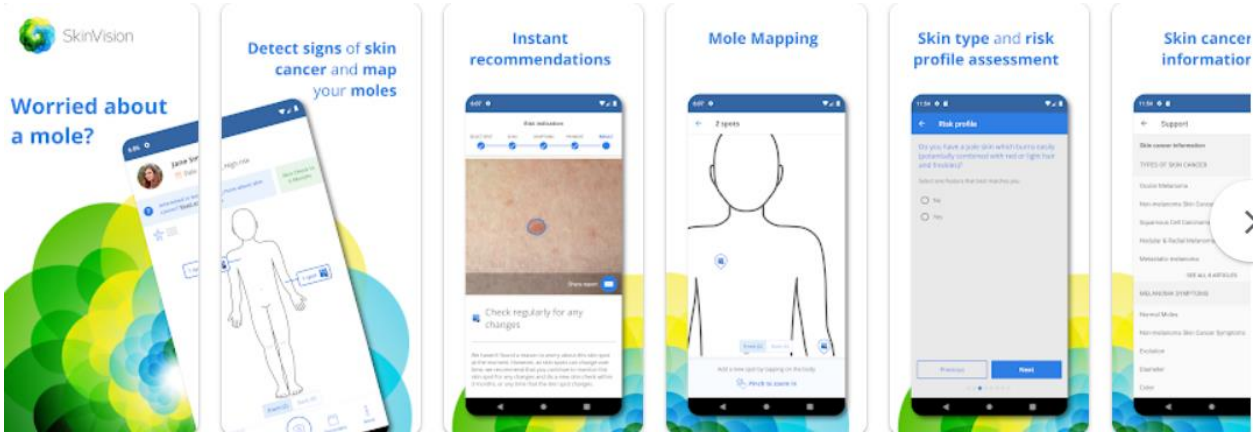

Рисунок 1.2 — Функціонал додатку SkinVision

3. Miiskin (https://miiskin.com/) — мобільний додаток, який використовує технології ШІ для моніторингу змін шкірних утворень та раннього виявлення ознак раку шкіри (див. рис. 1.3). Додаток дозволяє користувачам вести довгостроковий моніторинг та порівняння зображень з часом.

Переваги:

– використання додатка може дозволити користувачам відчувати більшу приватність при документуванні шкірних уражень, ніж при безпосередньому показі їх медичному фахівцеві;

– освітній компонент.

Недоліки:

– незручний інтерфейс;

– вартість та доступність;

– обмежений функціонал.

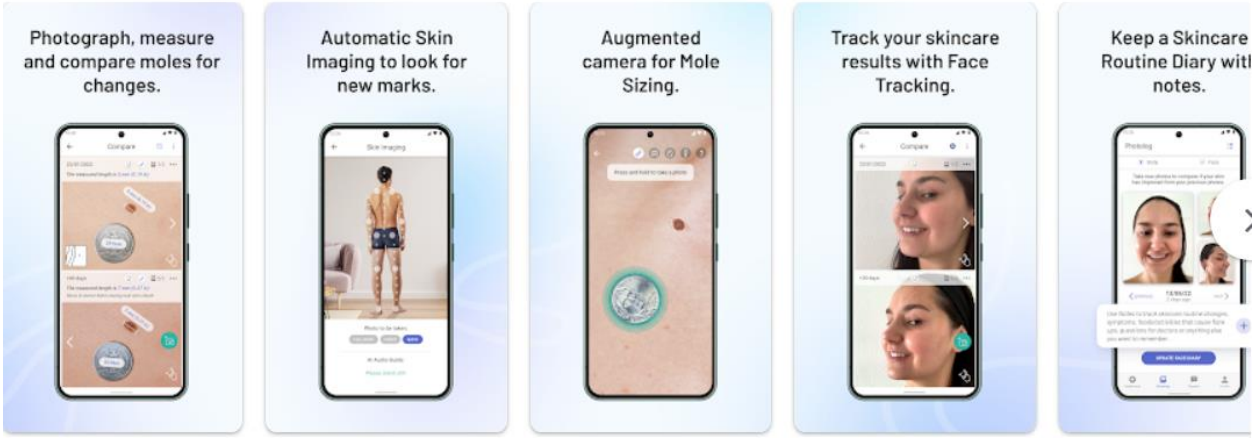

Рисунок 1.3 — Функціонал додатку Miiskin

4. AI Dermatologist: Skin Scanner (https://aidermatologist.com) — проєкт Google, який розробляє інструменти на основі штучного інтелекту для допомоги в діагностиці шкірних захворювань. Ця ініціатива спрямована на створення точних і доступних інструментів для виявлення широкого спектру дерматологічних станів (див. рис. 1.4).

Переваги:

– інноваційність, точність;

– підвищення обізнаності про здоров'я шкіри.

Недоліки:

– потреба у верифікації медичними фахівцями;

– конфіденційність та безпека даних.

Рисунок 1.4 — Функціонал додатку AI Dermatologist

Отже, аналіз програмних рішень у сфері діагностики шкірних захворювань, особливо тих, що базуються на технологіях ШІ, показує, що хоч і існуючі рішення зосереджуються на визначенні станів шкіри та їх моніторингу з метою сприяння здоров’ю, вони не завжди доступні для всіх категорій користувачів через вартість підписки або одноразові платежі.

У цьому контексті, розробка чат-боту, який буде не тільки інтегрувати алгоритми ШІ для точної діагностики, але й буде цілком безкоштовним для кінцевих користувачів. Такий підхід не тільки сприятиме ранньому виявленню та ефективному лікуванню шкірних захворювань, але й зможе знизити загальне навантаження на систему охорони здоров’я, надаючи користувачам інструменти для профілактики та попередньої самодіагностики.

1.3 Огляд алгоритмів

У контексті нашої задачі, детектування означає процес ідентифікації та класифікації шкірних захворювань за допомогою штучного інтелекту, зокрема, методів машинного та глибокого навчання. Цей процес включає аналіз зображень шкірних уражень з метою виявлення патологічних змін, які можуть вказувати на конкретні захворювання. Детектування допомагає автоматизувати процес первинної діагностики, забезпечуючи швидке та ефективне виявлення потенційних проблем із здоров’ям шкіри.

Для цих цілей можуть бути застосовані декілька алгоритмів та підходів.

Одним з підходів є використання алгоритмів машинного навчання (Machine Learning, ML), що використовують статистичні методи для дозволу комп'ютерам «вчитися» з даних, без явного програмування для кожної специфічної задачі. У контексті діагностики шкірних захворювань, це може включати навчання моделей на великих наборах зображень шкірних уражень, дозволяючи системі ідентифікувати паттерни, що вказують на конкретні захворювання.

Ще одним підходом можна виділити алгоритми глибокого навчання (Deep Learning, DL). DL є підмножиною ML і використовує багатошарові нейронні мережі для моделювання складних абстракцій. CNN (Convolutional Neural Network), спеціалізований тип нейронних мереж, є особливо потужним у візуальному аналізі зображень. Вони автоматично та ефективно виявляють ключові особливості зображень без потреби в ручному витягненні ознак, що робить їх ідеальними для розпізнавання шкірних захворювань.

Ці алгоритми поділяються на дві окремі, але взаємопов’язані області: сегментація та класифікація шкірних захворювань.

У сфері медичних зображень традиційне розпізнавання зображень в основному покладається на сегментацію країв і методи виділення ознак [6]. Для всіх шкірних захворювань точна сегментація зображень шкіри є важливою для виявлення та локалізації уражень.

Методи сегментації зображень шкіри в машинному та глибокому навчанні дозволяють точно виділити області інтересу на зображеннях, що є критично важливим для діагностики та подальшого лікування шкірних захворювань.

Далі буде проведено аналіз методів сегентації в ML та DL для вирішення нашої задачі.

Методи сегментації в машинному навчанні:

1. Кластеризація (наприклад, K-середніх): Розділяє зображення на сегменти (кластери) на основі схожості кольорів або текстур, де K – це задана кількість кластерів [7]. Це простий і швидкий метод, але може бути не дуже точним у складних випадках. Приклад сегментації зображення при K=6, наведено на рисунку 1.5.

Рисунок 1.5 — Сегментація зображення методом K-середніх

2. Методи, засновані на границях (наприклад, детектори країв Собеля): Алгоритм країв Собеля використовує два ядра фільтра для обчислення приблизних значень градієнтів яскравості зображення, що дозволяє виявити краї та орієнтацію країв. Один фільтр виявляє зміни яскравості по горизонталі, інший – по вертикалі. Це може бути корисним для виявлення контурів уражень.

3. Водорозділ (Watershed): Алгоритм, що симулює розлив води для визначення границь об'єктів. Ефективний для виділення окремих уражень, але може викликати перерозділ при наявності шуму [8].

Методи сегментації в глибокому навчанні:

1. Згорткові нейронні мережі (CNN): Хоча CNN переважно використовуються для класифікації, вони також можуть бути адаптовані для сегментації за допомогою повнозв’язних шарів [9].

2. U-Net: Спеціалізована архітектура для сегментації, що має симетричну структуру для точної локалізації об’єктів на зображенні. Часто використовується в медичній сегментації зображень. U-Net є розвитком стандартних CNN і оптимізована для задач біомедичної сегментації зображень. У цій архітектурі використовується симетрична структура з «стисненням» (зниження розміру зображення та підвищенням глибини) і «розширенням» (збільшення розміру зображення та зменшенням глибини), яка дозволяє точно локалізувати та класифікувати різні частини зображення. Архітектура мережі наведена на рисунку 1.6. Вона складається з шляху, що звужується (ліворуч) і розширюється шляху (праворуч). Звуження шлях – типова архітектурі згорткової нейронної мережі. Він складається з повторного застосування двох згорток 3×3, за якими слідують ініт ReLU та операція максимального об'єднання (2×2 ступеня 2) для зниження дозволу. На кожному етапі знижувальної дискретизації канали властивостей подвоюються. Кожен крок у шляху, що розширюється, складається з операції підвищує дискретизації карти властивостей, за якою слідують: згортка 2×2, яка зменшує кількість каналів властивостей; об'єднання з відповідним чином обрізаною картою властивостей зі стягується шляху; дві 3×3 згортки, за якими слідує ReLU [10].

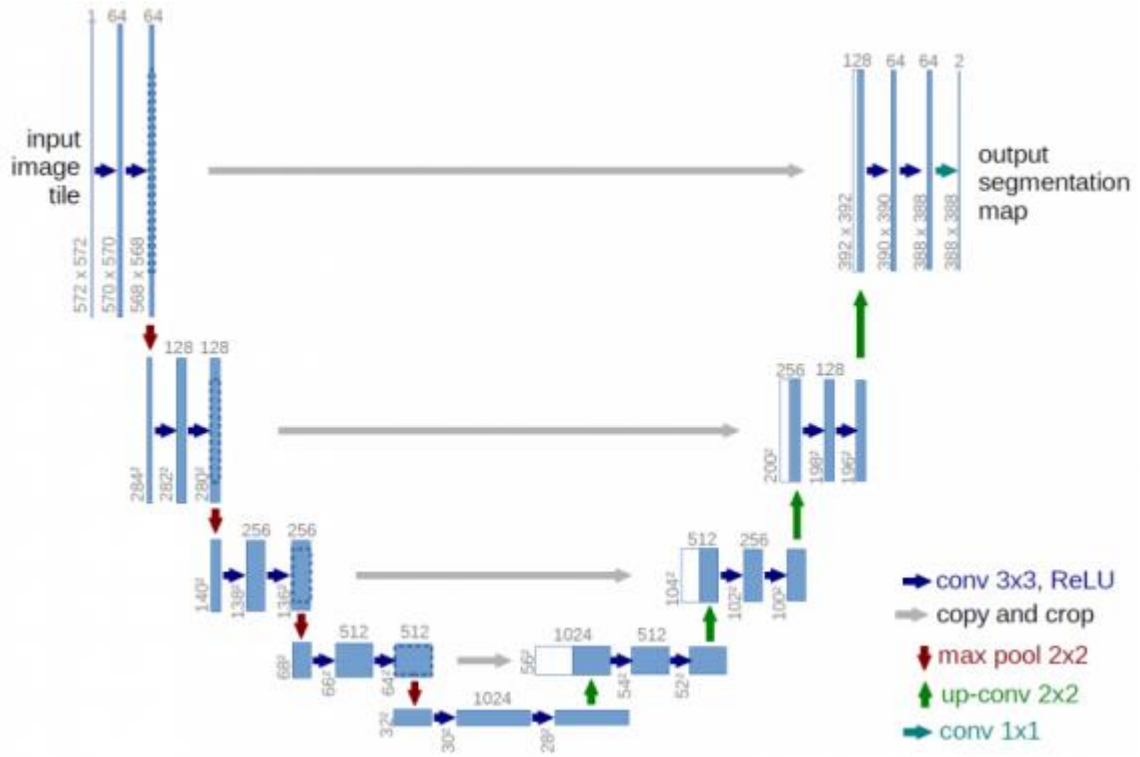

Рисунок 1.6 — Архітектура U-net

3. SegNet: Архітектура схожа на U-Net, але з оптимізованою структурою декодера для ефективнішої обробки зображень. SegNet — це архітектура згорткової нейронної мережі (CNN), призначена для семантичної сегментації в комп'ютерному зорі (рис. 1.7). Семантична сегментація — це завдання класифікації кожного пікселя зображення до певної категорії чи класу, наприклад визначення об'єктів та їхніх меж. SegNet є однією з багатьох архітектур глибокого навчання, розроблених для вирішення цієї проблеми. SegNet базується на архітектурі кодера-декодера. Він призначений для отримання зображення як вхідних даних і створення піксельної карти міток як вихідних даних. Кодер фіксує функції високого рівня, застосовуючи серію згорткових шарів і шарів об'єднання. Ключовим нововведенням у SegNet є мережа декодера, яка використовує індекси просторового об'єднання, створені під час максимального об'єднання на етапі кодування, для підвищення дискретизації та створення карти сегментації. Це форма підвищення дискретизації, яка одночасно ефективна для пам'яті та допомагає

зберегти дрібні деталі на виході [11].

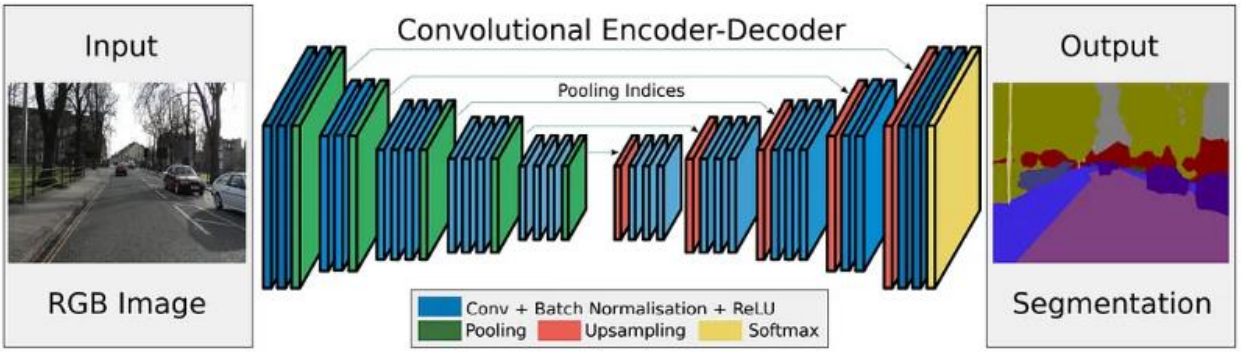

Рисунок 1.7 — Архітектура SegNet

4. DeepLab: Використовує атрофовану (dilated) конволюцію для покращення здатності захоплення контексту на зображенні та покращення точності сегментації. DeepLab — це архітектура семантичної сегментації. Спочатку вхідне зображення проходить по мережі за допомогою розширених звивин. Потім вихідні дані з мережі білінійно інтерполюються та проходять через повністю підключену CRF для точного налаштування результату [12].

Отже, у контексті сегментації зображень, глибоке навчання має кілька ключових переваг порівняно з традиційними методами машинного навчання:

– Автоматичне навчання ознак: Моделі глибокого навчання автоматично витягують розширені ознаки з даних, на відміну від традиційних методів, що спираються на ознаки, розроблені вручну.

– Краще узагальнення: Моделі глибокого навчання ефективно працюють з меншими наборами даних і мають кращі узагальнювальні здібності порівняно з методами машинного навчання, які потребують об'ємних даних і великої ручної роботи по створенню ознак.

– Масштабування: Глибоке навчання дозволяє ефективно обробляти масштабні набори даних і користується перевагами розподілених обчислень, на відміну від методів машинного навчання, обмежених через необхідність ручного створення ознак.

– Адаптивність: Глибоке навчання забезпечує велику гнучкість,

адаптуючись до різних завдань і типів даних без потреби в специфічному проектуванні під кожне завдання, на відміну від традиційних методів.

Враховуючи завдання сегментації зображень шкіри для ідентифікації та локалізації уражень, U-Net, здається, кращим вибором. Цей вибір обумовлений спеціалізацією U-Net на задачах сегментації та її здатністю ефективно працювати навіть з обмеженими даними, що часто є випадком у медичному зображенні. Архітектура U-Net, розроблена спеціально для завдань сегментації, робить її вправною в точному окресленні зацікавлених областей, що є необхідним для такої задачі. Однак, якщо задача включає роботу з дуже складними сценаріями сегментації та обчислювальні ресурси не є обмеженням, DeepLab може забезпечити кращий результат завдяки його складному підходу до вловлення дрібних деталей.

Далі буде проведено аналіз методів класифікації в ML та DL для вирішення нашої задачі.

Машина опорних векторів (Support Vector Machine, SVM) — є алгоритмом машинного навчання, який використовується в класифікації та регресії. Основна ідея полягає в знаходженні гіперплощини, яка найкращим чином розділяє набір навчальних даних на класи [13].

Алгоритм класифікації Random Forest є ансамблевим методом машинного навчання, який використовує множину дерев рішень для вирішення задачі класифікації або регресії (див. рис. 1.8). Він побудований на принципі «мудрості натовпу», де кілька моделей (у цьому випадку дерев рішень) об'єднуються для вироблення загального рішення, яке зазвичай є більш точним і стабільним, ніж рішення від будь-якої окремої моделі. Застосування Random Forest у контексті діагностики шкірних захворювань:

– Екстракція ознак. З зображень шкірних захворювань можна екстрагувати різноманітні ознаки, такі як текстурні, колірні характеристики, контури ураження та інші. Random Forest добре справляється з великою кількістю ознак і може визначати їх важливість.

– Класифікація захворювань. Після екстракції ознак модель Random Forest може бути навчена розпізнавати та класифікувати різні типи шкірних захворювань, використовуючи зазначені ознаки.

– Оцінка важливості ознак. Однією з переваг Random Forest є можливість оцінки важливості ознак, що дозволяє визначити, які ознаки найбільш значущі для класифікації шкірних захворювань. Це може допомогти у виборі найбільш інформативних ознак для подальшого аналізу та діагностики [14].

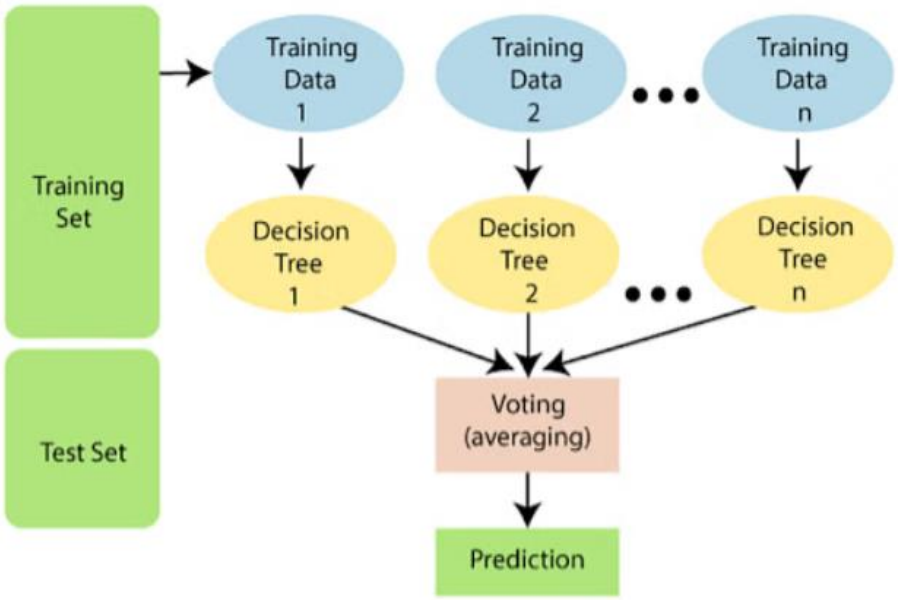

Рисунок 1.8 — Робота алгоритму Random Forest

Алгоритм класифікації MobileNet — це архітектура глибокої згорткової нейронної мережі (CNN), розроблена з метою забезпечення високої ефективності в умовах обмежених обчислювальних ресурсів, таких як мобільні пристрої або вбудовані системи (рис. 1.9). MobileNet використовує глибокі згортки з роздільними фільтрами, що дозволяє зменшити кількість параметрів та обчислювальну складність без значної втрати точності моделі [15].

Основні компоненти MobileNet:

– Глибокі згортки з роздільними фільтрами: MobileNet спочатку застосовує глибоку згортку для кожного каналу вхідного зображення окремо, а потім використовує 1x1 згортки (точкові згортки) для комбінування вихідних каналів. Цей підхід значно зменшує кількість необхідних мультиплікацій та додавань порівняно з традиційними згортковими операціями.

– Гіперпараметри для масштабування ширини та роздільної здатності: MobileNet дозволяє легко адаптувати архітектуру до різних обчислювальних обмежень, змінюючи ширину мережі (кількість фільтрів у згортках) та роздільну здатність вхідних зображень.

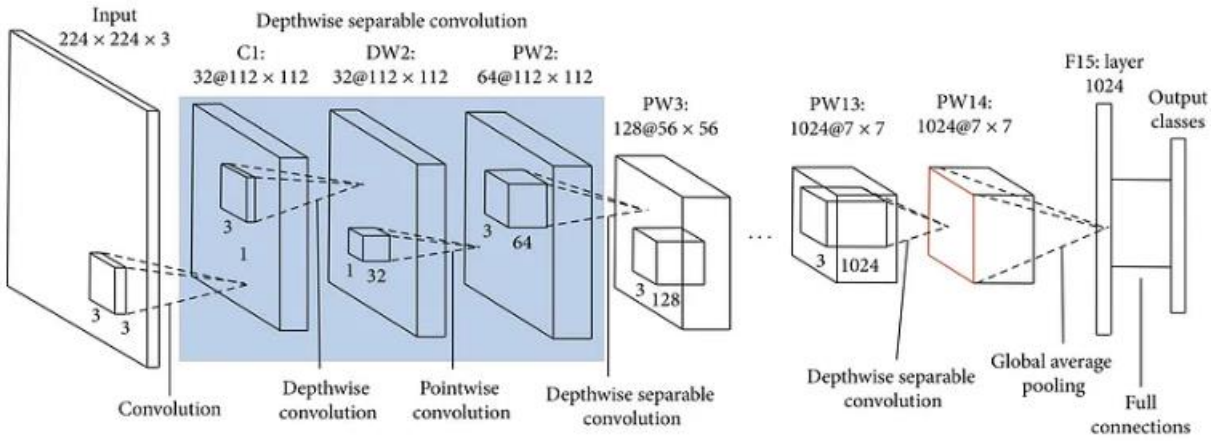

Рисунок 1.9 — Архітектура MobileNet

EfficientNet — це архітектура CNN від дослідників з Google, яка дозволяє підвищити продуктивність за допомогою методу масштабування, що називається складовим масштабуванням (рис. 1.10). Цей метод масштабування рівномірно масштабує всі розміри глибини/ширини/дозвіл на фіксовану величину (складовий коефіцієнт) [16].

Застосування EfficientNet у контексті діагностики шкірних захворювань:

– Висока точність. EfficientNet забезпечує високу точність у класифікації зображень, що робить його ідеальним вибором для завдань діагностики шкірних захворювань, де важливо точно розрізнити між різними

типами уражень та станів шкіри.

– Ефективність використання ресурсів. Оскільки EfficientNet оптимізований для ефективного використання обчислювальних ресурсів, його можна застосовувати не тільки на потужних серверах, але й на мобільних пристроях або в умовах з обмеженими обчислювальними можливостями. Це робить його придатним для мобільних додатків з діагностики шкірних захворювань.

– Масштабованість. Завдяки своїй унікальній здатності ефективно масштабуватися, EfficientNet може бути налаштований для оптимальної продуктивності залежно від доступних обчислювальних ресурсів і потреб додатка. Це дозволяє розробникам тонко налаштувати модель для максимальної точності чи швидкодії залежно від конкретних вимог.

– Гнучкість застосування. EfficientNet може бути застосований не тільки для класифікації зображень, але й для більш складних завдань, таких як детектування об’єктів (наприклад, локалізація уражень на зображенні шкіри) та сегментація зображень (наприклад, точне виділення контурів уражень).

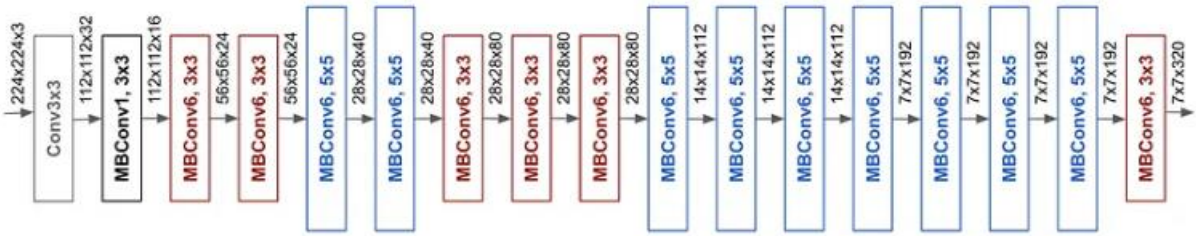

Рисунок 1.10 — Архітектура EfficientNet

Алгоритм VGG (Visual Geometry Group) – це одна з моделей глибокого навчання, що була розроблена групою VGG з Оксфордського університету. Ця модель стала відомою завдяки своєму виступу в змаганні ImageNet Large Scale Visual Recognition Challenge (ILSVRC) у 2014 році.

Основною характеристикою архітектури VGG є її простота, що базується на використанні великої кількості однакових згорткових блоків, кожен з яких містить згортку з ядром 3x3, за якою слідує функція активації ReLU, і кілька повністю з'єднаних шарів на кінці. Існує два варіанти моделі VGG: 16- та 19-рівнева мережа, причому VGG-19 (19-рівнева мережа) є удосконаленням моделі VGG-16 (16-рівнева мережа) [17].

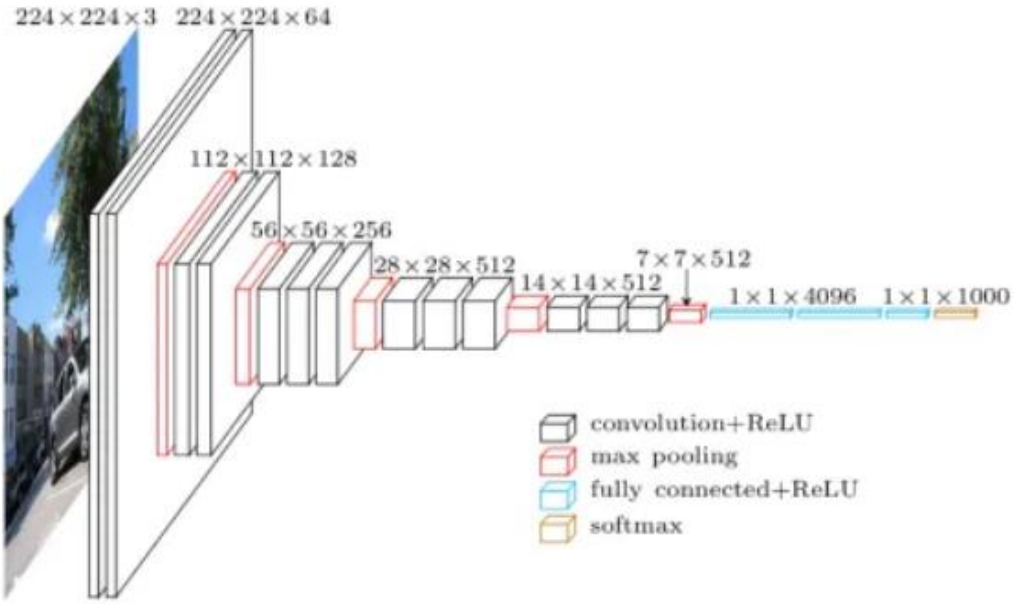

Рисунок 1.11 — Архітектура VGG

За результатами порівняння, EfficientNet видається найкращим варіантом для нашої задачі діагностики шкірних захворювань. Це пояснюється його високою точністю, ефективністю та здатністю до масштабування, що дозволяє оптимально використовувати доступні обчислювальні ресурси. Також EfficientNet може бути легко адаптований до різних умов застосування, що робить його ідеальним вибором для мобільних додатків і вбудов.

1.4 Постановка задачі

Функціональні вимоги до чат-боту для діагностування шкірних захворювань:

1. Введення даних:

– завантаження фотографій шкірних уражень: користувачі мають змогу завантажити фотографії уражень шкіри зі свого пристрою або використовувати камеру для створення нових знімків.

2. Сегментація зображення:

– автоматичне виявлення об'єктів (сегментація зображення) на шкірі.

3. Класифікація сегментів та ідентифікація шкірних захворювань:

– система повинна класифікувати сегменти, локалізувати шкірні ураження на фотографії та ідентифікувати тип шкірного захворювання.

4. Відображення результатів:

– показ результатів користувачу: чат-бот має надавати користувачеві інформацію про діагностовані шкірні ураження, їх можливі типи.

5. Інтерактивність та користувацький досвід:

– діалоговий інтерфейс: розробка зрозумілого для користувача діалогового інтерфейсу, що сприяє легкій взаємодії під час завантаження фотографій та отримання діагностичних висновків.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір засобів програмування

У контексті розробки інформаційної системи для діагностики шкірних захворювань з використанням чат-бота, ключовим елементом є вибір програмних мов та фреймворків. Python, завдяки своїй широкій підтримці в наукових та дослідницьких спільнотах, особливо у сфері машинного та глибокого навчання, вибирається як основна мова програмування. Це забезпечує доступ до великої кількості бібліотек, таких як TensorFlow або PyTorch, які є фундаментальними для розробки та тренування моделей глибокого навчання.

Для розробки чат-бота, що інтегрується з месенджерами, такими як Telegram або Facebook Messenger, може використовуватися фреймворк, такий як python-telegram-bot або PyMessenger. Ці інструменти спрощують розробку ботів, забезпечуючи гнучке управління повідомленнями та інтерактивністю з користувачем [18].

Для обробки та аналізу зображень шкіри важливою є бібліотека OpenCV, що дозволяє реалізувати предварительну обробку зображень, їх сегментацію та підготовку до класифікації. А Keras, високорівневий API, що працює на основі TensorFlow, забезпечує зручність у створенні нейронних мереж, спрощуючи експериментування з різними архітектурами.

Також важливою є інтеграція з веб-інтерфейсом для зручності користувачів, що може бути реалізована через Flask або Django, які забезпечують гнучкість у створенні веб-сервісів.

Вибір цих інструментів створює потужну та гнучку основу для розробки інформаційної системи діагностики шкірних захворювань, що забезпечує не лише технічну ефективність та надійність, але й відмінний користувацький досвід.

2.2 Вибір методів та алгоритмів розроблення

Для розробки системи діагностики шкірних захворювань, ретельний вибір методів та алгоритмів обробки даних є критично важливим. Наступні інструменти та фреймворки були обрані для кожного ключового етапу розробки:

1. Попередня обробка та аналіз зображень шкіри:

– OpenCV: Ця бібліотека використовується для обробки зображень, включаючи фільтрацію, перетворення та виділення ключових ознак. Вона дозволить підготувати зображення для подальшого аналізу.

– Scikit-image: Інший інструмент для обробки зображень, який допомагає у визначенні текстури, кольору та форми уражень на шкірі.

2. Детектування та класифікація уражень шкіри:

– TensorFlow або PyTorch: Обидва фреймворки забезпечують потужні можливості для глибокого навчання, включаючи розробку, тренування та впровадження нейронних мереж, спеціалізованих на діагностиці шкірних захворювань.

– Keras: Високорівневий API, що спрощує процес розробки та тренування моделей глибокого навчання, працює з TensorFlow, дозволяє швидко експериментувати з різними архітектурами.

3. Оптимізація та вдосконалення моделей:

– Optuna або Hyperopt: Ці бібліотеки призначені для оптимізації гіперпараметрів, що дозволяє досягти кращої продуктивності моделей глибокого навчання за рахунок вибору оптимальних налаштувань.

4. Інтеграція з чат-ботом:

– python-telegram-bot: Фреймворк для створення чат-ботів у Telegram. Забезпечує зручні інструменти для взаємодії з користувачами, приймання зображень і надсилання результатів діагностики.

– Flask: Для розробки веб-сервісу, що взаємодіє з чат-ботом, Flask використовується як легкий фреймворк, що дозволяє швидко створювати та

розгортати веб-застосунки.

5. Аналіз та обробка даних:

– Pandas і NumPy: Обробка та аналіз даних, зокрема, для статистичного аналізу результатів діагностики та управління даними пацієнтів.

Ці бібліотеки забезпечують швидкі та ефективні інструменти для роботи з великими наборами даних, що дозволяє здійснювати глибокий аналіз результатів діагностики [19].

Обрані інструменти та технології забезпечують міцну основу для розробки інноваційної системи діагностики шкірних захворювань. Використання передових бібліотек машинного та глибокого навчання, таких як TensorFlow і Keras, дозволяє ефективно реалізувати складні моделі нейронних мереж для точної класифікації уражень шкіри. Фреймворки для створення чат-ботів, як-от python-telegram-bot, та веб-серверні рішення, такі як Flask, забезпечують зручний інтерфейс для кінцевих користувачів, дозволяючи їм легко отримувати консультації та результати діагностики. Паралельно, інструменти для аналізу та обробки даних, включаючи Pandas та NumPy, сприяють ефективній роботі з даними, що є ключовим для підвищення точності діагностики. Такий комплексний підхід до вибору засобів програмування та алгоритмів не тільки підвищує ефективність розроблюваної системи, але й забезпечує високий рівень адаптивності та масштабованості проєкту [20].

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Структурно-функціональне моделювання

IDEF0 — це стратегія моделювання, що сприяє створенню графічних зображень процесів та взаємодій в складних проєктах, таких як розробка системи для діагностики шкірних захворювань з використанням алгоритмів машинного навчання. Цей метод застосовується для детального аналізу та оптимізації функціональності системи. Використання IDEF0 у розробці нашої системи дозволяє візуалізувати та структурувати процеси виявлення шкірних захворювань, аналізу зображень та інтеракції з користувачем.

Застосування IDEF0 у нашому проєкті допомагає розділити систему на чіткі модулі, які відображають специфічні задачі, такі як аналіз дерматологічних зображень, класифікація захворювань і зворотний зв'язок з користувачем. Рисунок 3.1 демонструє спрощену функціональну модель, яка відтворює структуру системи для аналізу шкірних зображень та надання діагностичних даних.

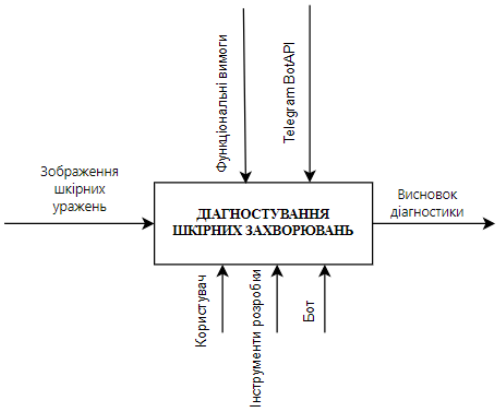

Рисунок 3.1 — IDEF0

У рамках методології IDEF1 для моделювання інформації, ми застосовуємо детальний підхід до візуалізації потоків даних між елементами системи, що використовуються для діагностики шкірних уражень. Ось як

IDEF1 може бути використана для структурування обміну інформацією в нашому проєкті (рис. 3.2):

1. Визначення інформаційних потоків:

– У методології IDEF1 потоки даних позначені стрілками, що демонструють напрям обміну інформацією між блоками. У нашій системі це може включати потік зображень уражень від користувача до процесу «Детектування та сегментація зображення».

2. Обробка інформації:

– Кожен функціональний блок обробляє інформацію згідно з визначеними алгоритмами. Наприклад, в блоку «Детектування та сегментація зображення» використовуються алгоритми ML та DL для аналізу зображень та ідентифікації характеристик уражень.

3. Розподіл інформації:

– IDEF1 використовується для ідентифікації маршрутизації інформаційних потоків. Результати процесу «Детектування та сегментація зображення» передаються до блоку «Бінарна класифікація сегментів» для подальшого аналізу та визначення типу захворювання.

Методологія IDEF1 полегшує вивчення та оптимізацію обміну інформацією у системі, забезпечуючи ясність в інформаційних взаємодіях та підвищуючи ефективність комунікації даних для точної та ефективної діагностики.

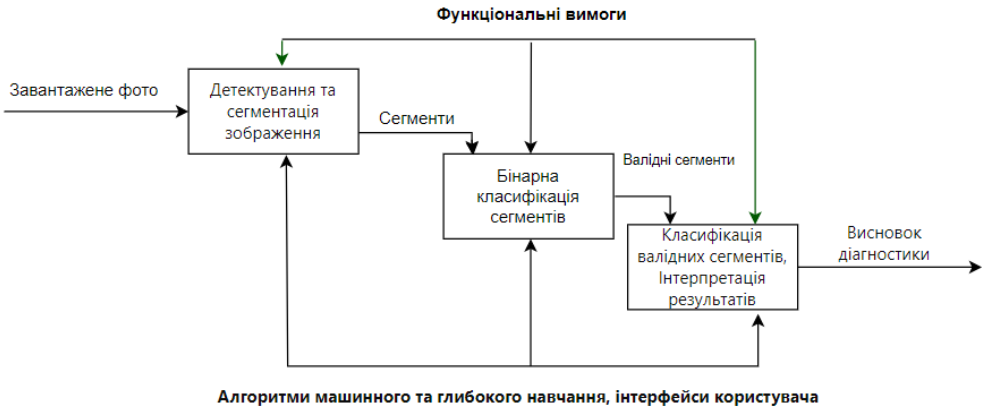

Рисунок 3.2 — IDEF1

3.2 Реалізації основних компонентів

Нейронні мережі можуть адаптуватися до різних задач за допомогою декількох спеціалізованих методів навчання, кожен із яких має унікальні особливості та призначення, відповідно до вимог задачі [21]:

1. Навчання з учителем (Supervised Learning): Цей метод передбачає тренування моделі на даних, які вже містять правильні відповіді, з метою навчити модель точно прогнозувати результати на основі вхідних даних.

2. Навчання без учителя (Unsupervised Learning): Під час такого тренування модель опрацьовує нерозмічені дані, щоб самостійно виявити приховані закономірності або структури, що можуть включати завдання кластеризації або зниження розмірності.

3. Слабокероване навчання (Semi-Supervised Learning): Цей підхід об'єднує елементи обох попередніх методів, використовуючи частково розмічені дані, де тільки деякі зразки мають мітки, що корисно в умовах обмеженої розмітки.

4. Навчання з підкріпленням (Reinforcement Learning): У цьому методі модель або агент навчається, взаємодіючи з середовищем, щоб виконувати дії, які максимізують нагороду, часто застосовується в задачах, де потрібно вчитися складним стратегіям.

5. Перенос знань (Transfer Learning): Застосування цього підходу включає використання моделі, навченої на одному завданні, для вирішення іншого, схожого завдання, що є особливо ефективним, коли доступна невелика кількість даних для нової задачі.

Зважаючи на успішний досвід інших дослідників і компаній у діагностуванні шкірних захворювань, було прийнято рішення застосувати перенос знань для нашої задачі. Цей метод дозволяє максимально ефективно використати наявні розробки і швидко досягти високих результатів у розв'язанні поставленої задачі, використовуючи передові технології і попередній досвід у цій області.

Було вибрано фреймворк Tensorflow для реалізації проєкту, оскільки він є одним з найбільш відомих та широко використовуваних інструментів для машинного навчання. В ході дослідження в інтернеті була виявлена модель для діагностики шкірних захворювань, яка була попередньо навчена на значному масиві даних — близько 5000 тисяч оригінальних зображень та включала 10 класів. (рис. 3.3).

```
acne  vulgaris        -  720
cellulitis            -  430
decubitus  ulcer      -  510
eczema                -  600
pityriasis versicolor -  480
psoriasis             -  550
scabies               -  390
tinea  corporis       -  410
tinea  pedis          -  470
urticaria             -  440
```

Рисунок 3.3 — Перелік класів моделі класифікатора шкірних уражень

Так як класифікатор досить чутливий до вхідного зображення, то було вирішено додати кроки для фільтрації зображення, а саме виділення ділянок із шкірою. Для цього було навчено простий бінарний класифікатор на основі MobileNetV2 з ємністю мережі 0.25 та розширенням рецептивного поля 192x192 пікселів. Для виділення (сегментації) ділянок, які потрібно класифікувати, можна використати готову та досить швидку модель на основі Faster Segment Anything [22]. Вона сегментує будь-яке зображення.

SAM (Segment Anything Model) — це модель для сегментації, яку розробила Meta AI навесні 2023 року. Ця модель швидко набула популярності серед AI-моделей. Модель SAM виконує сегментацію об’єктів на фотографіях відповідно до заданих користувачем умов, які можуть бути у формі точки, прямокутника або тексту. За словами розробників, завдяки

попередньо обчисленому ембедингу зображення, SAM може працювати в реальному часі на CPU у веббраузері. Крім того, модель можна застосовувати без додаткового навчання для інших завдань, які не стосуються безпосередньо сегментації, наприклад, для визначення країв та позиціонування об'єктів.

SAM складається з двох компонентів: кодувальника зображень на основі ViT і декодера масок, керованого промптами, які працюють послідовно (див. рис. 3.4).

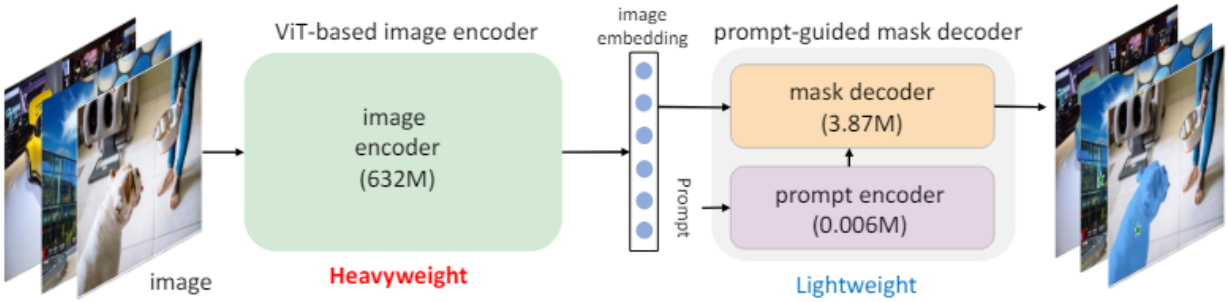

Рисунок 3.4 — Архітектура Segment Anything Model

Сегментаційні маски SAM, хоч і достатньо точні, але не завжди мають потрібний рівень деталізації, що послужило мотивацією для створення моделі Semantic-SAM. Semantic-SAM — це сегментаційна модель, яка дозволяє розпізнавати та сегментувати об’єкти на будь-якому рівні деталізації. Модель відрізняється гарним розумінням семантики об'єктів. На прикладі нижче Semantic-SAM за допомогою одного кліка від користувача передбачає ієрархічну послідовність сегментаційних масок різного розміру, починаючи від голови людини та закінчуючи цілим вантажівкою (див. рис. 3.5-3.6).

Рисунок 3.5 — Сегментація по Semantic-SAM

Для забезпечення глибокого розуміння семантики об'єктів модель проходила навчання на семи різних датасетах, у тому числі на SA-1B. Ці датасети включають сегментаційні маски з різними ступенями деталізації: деякі з них надають анотації тільки на рівні окремих об'єктів (наприклад, COCO), тоді як інші включають інформацію як про об'єкти, так і про їх частини (наприклад, Pascal Part).

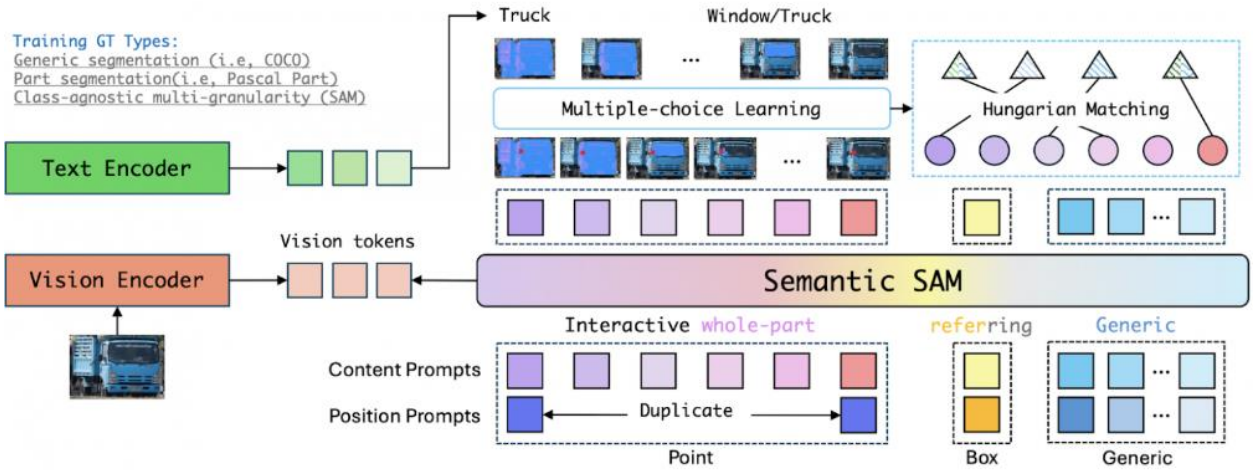

Рисунок 3.6 — Архітектура Semantic-SAM

HQ-SAM — це покращена версія моделі SAM, розроблена для кращої сегментації складних об'єктів. Розробники моделі вирішили внести декілька нововведень, не збільшуючи при цьому кількість параметрів моделі значно. Основні нововведення включають High-Quality Output Token (HQ-Output Token) і блок Global-Local Feature Fusion. HQ-Output Token допомагає моделі краще розпізнавати об'єкти, використовуючи спеціальні токени для керування увагою моделі, тоді як блок Fusion поєднує інформацію з різних частин зображення для точнішої сегментації (див. рис. 3.7-3.8).

Нові компоненти моделі навчаються на спеціально створеному датасеті HQSeg-44k, який містить 44 тисячі детальних масок для тренування. Завдяки тому, що основні ваги моделі залишаються незмінними, навчання нових

компонентів займає всього 4 години на 8 GPU, що робить процес більш ефективним і швидким.

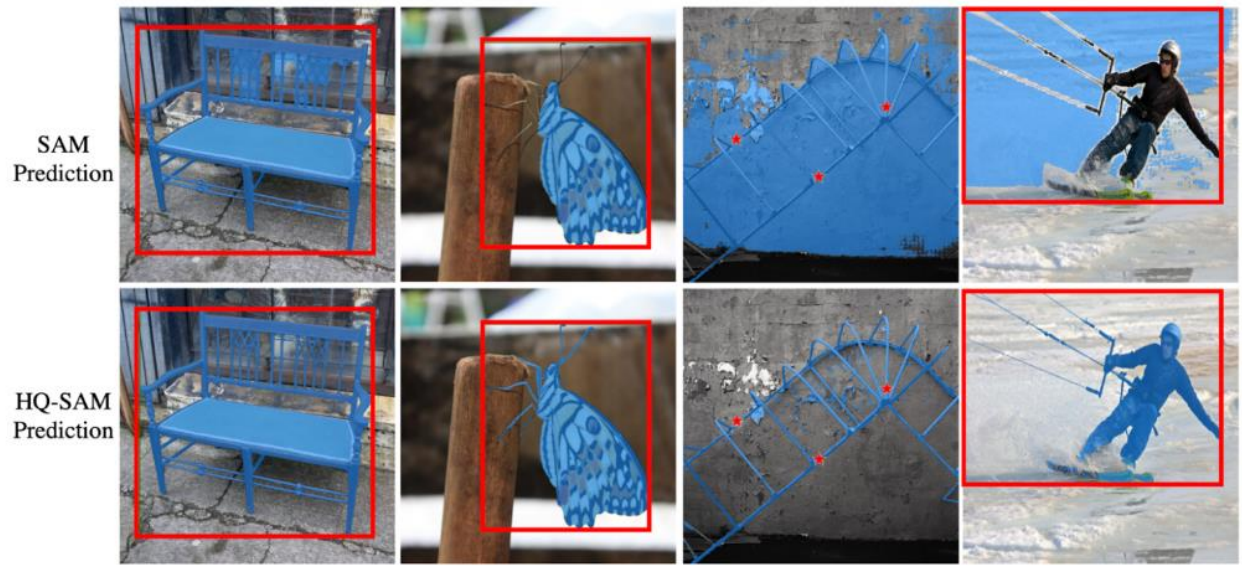

Рисунок 3.7 — Покращення якості сегментації HQ-SAM складних об'єктів

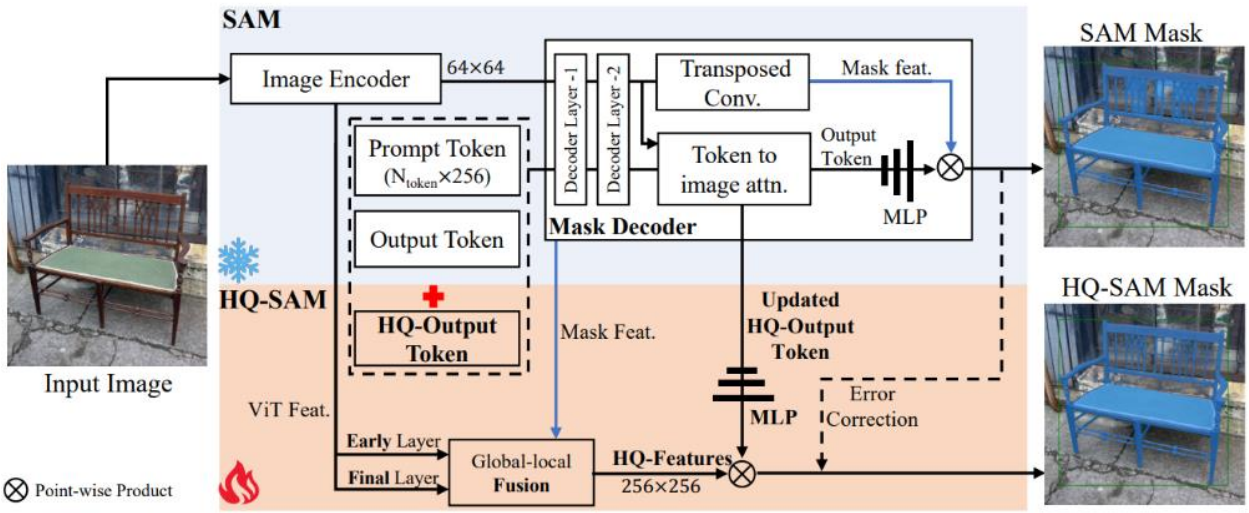

Рисунок 3.8 — Архітектура HQ-SAM

Через велику обчислювальну складність, спричинену використанням трансформерного енкодера, модель зазнавала обмежень у широкому застосуванні. Тому було прийнято рішення замінити складний енкодер ViT у SAM на більш простий згортковий детектор. Це дозволило прискорити

роботу моделі в 50 разів, при цьому зберігаючи якість передбачень на аналогічному рівні (див. рис. 3.9).

Метод FastSAM включає два основні етапи: спочатку відбувається сегментація всіх об'єктів на зображенні, а потім вибір конкретного об'єкта відповідно до заданого промпта.

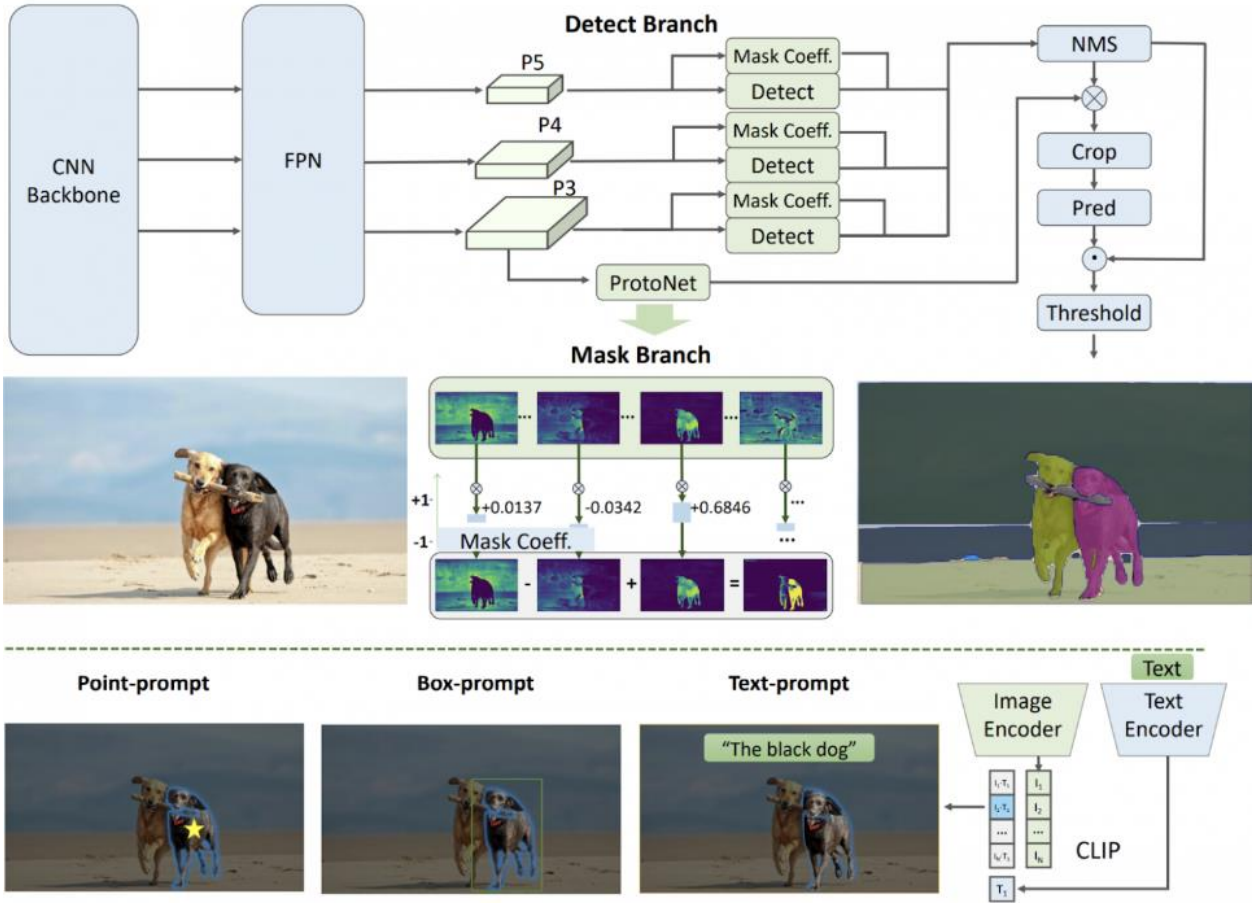

Рисунок 3.9 — Архітектура FastSAM

Якщо ми хочемо використовувати SAM на мобільному пристрої, існує Faster SAM — легша версія, оптимізована для мобільних телефонів. У оригінальному SAM енкодер з 600 мільйонами параметрів є найскладнішою частиною, тому для спрощення моделі в MobileSAM енкодер ViT-H замінений на ViT-S, що має менше параметрів і забезпечує кращу швидкість обчислень.

Заміна ViT-H на ViT-S може вплинути на продуктивність, оскільки енкодер і декодер у SAM взаємозалежні і навчаються разом (див. рис. 3.10). Проблему погіршення продуктивності вирішує метод роздільної дистиляції, який включає дистиляцію енкодера та донавчання декодера на двох окремих етапах.

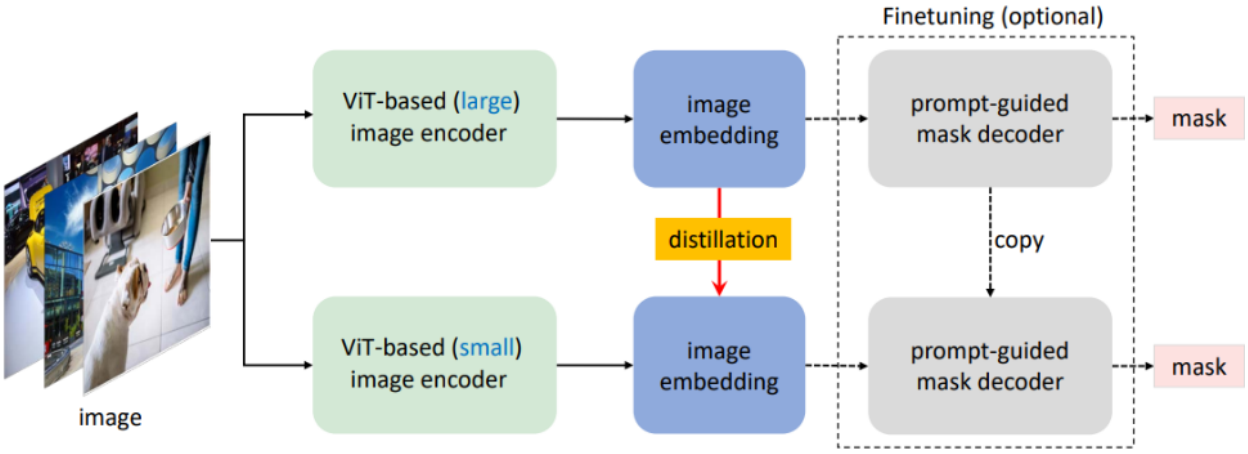

Рисунок 3.10 — Архітектура Faster SAM

Дистиляція — це метод, який застосовується для тренування нових моделей за допомогою передачі знань від попередньо навчених моделей. У рамках проекту MobileSAM, цей метод дозволяє тренувати легкий енкодер зображень, що ефективно співпрацює з декодером масок з оригінального SAM, імітуючи поведінку базового енкодера.

MobileSAM в 60 разів менший за оригінальний SAM і зберігає порівнянну якість. Водночас, він у 4 рази швидший і в 7 разів легший, ніж FastSAM, що робить його ідеальним для використання на мобільних пристроях з обмеженими ресурсами. MobileSAM може бути навчений на одному GPU менше ніж за один день, а обробка одного зображення займає всього 10 мілісекунд.

Приклад коду для завантаження моделі сегментації в форматі onnx:

```
import cv2
import numpy as np
from models.sam_onnx import SegmentAnythingONNX

model = SegmentAnythingONNX(
```

```
    "models/mobilesam.encoder.onnx",
    "models/mobilesam.decoder.onnx"
)

def segmentation(image):

    prompt = [{"type": "rectangle", "data": [0, 0, image.shape[1],
image.shape[0]]}]
    print(image.shape[0], image.shape[1])
    embedding = model.encode(image)
    image_area = image.shape[0]* image.shape[1]
    masks = model.predict_masks(embedding, prompt)
    mask = 1*(masks[0] > 0).astype(np.uint8).reshape(masks.shape[2],
masks.shape[3], 1)
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    boxes = []
    if contours:
        for c in contours:
            contour_area = cv2.contourArea(c)
            if contour_area > image_area*0.2:
                boxes.append(list(cv2.boundingRect(c)))

    return masks, boxes
```

Приклад сегментації зображень на основі Faster Segment Anything наведено на рисунку 3.11.

Рисунок 3.11 — Приклад сегментації на основі Faster SAM

Спочатку потрібно створити датасет і конвертувати його в формат tfrecord для оптимізації та прискорення процесу навчання.

Функція, яка виконує це завдання:

```
def build_tfrecords(dataset, output_path, images_annotations,
category_index):
    writer = tf.compat.v1.python_io.TFRecordWriter(output_path)
    label_map_dict = label_map_util.get_label_map_dict(LABEL_PATH)

    print("Building TFRecord files for dataset: %s" % dataset)
```

```
    for idx, (image, _annotations) in enumerate(images_annotations):
        if idx % 100 == 0:
            print('%d of %d annotations' % (idx,
len(images_annotations)))

        _, tf_example, num_annotations_skipped = create_tf_example(
            image=image,
            annotations_list=_annotations,
            image_dir=IMAGES_DIR,
            category_index=category_index,
            include_masks=False
        )

        writer.write(tf_example.SerializeToString())

    writer.close()
    print("Done!")
```

На рисунку 3.12 представлено зразки навчальних даних.

Рисунок 3.12 — Зразки навчальних даних

Далі потрібно зібрати детекторний модуль, розширивши архітектуру MobileNetV2, щоб адаптувати її для виявлення захворювань. Це включає додавання додаткових шарів до базової моделі MobileNetV2, які займатимуться розпізнаванням та класифікацією об’єктів у зображеннях.

Зазвичай MobileNetV2 використовується для ефективної екстракції ознак із зображень, проте для повноцінної детекції об’єктів необхідно інтегрувати додаткові компоненти.

Також для оптимізації процесу навчання налаштуємо збереження контрольних точок та введемо механізм раннього припинення навчання, щоб уникнути перетренування (overfitting).

Відповідний код для налаштувань:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=8,
verbose=2)

# Налаштування Model Checkpoint
model_checkpoint = ModelCheckpoint(
    'model_checkpoint.h5', monitor='val_loss', verbose=1,
save_best_only=True, save_weights_only=True)
```

Далі ми проводимо тренування моделі та аналізуємо результати на валідаційній вибірці. Графік, що демонструє зміну похибки в залежності від кількості ітерацій, представлений на рисунку 3.13.

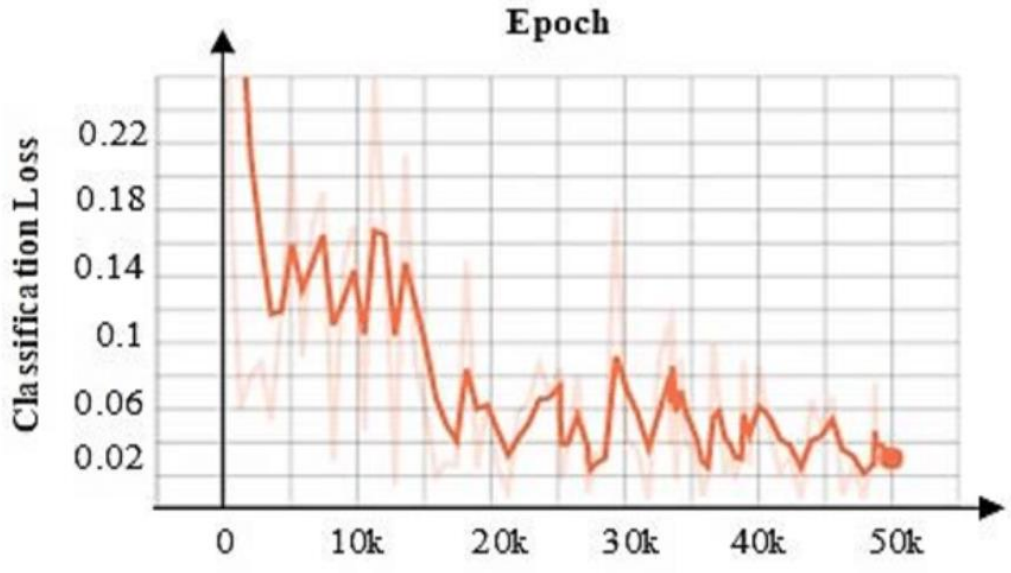

Рисунок 3.13 — Графік, який показує зміну похибки у залежності від кількості ітерацій

Оцінимо результати за допомогою кількісного аналізу, здійснивши прогнози на тестовій вибірці та створивши матрицю невідповідностей

(confusion matrix) для отриманих результатів. Відповідна матриця невідповідностей представлена на рисунку 3.14.

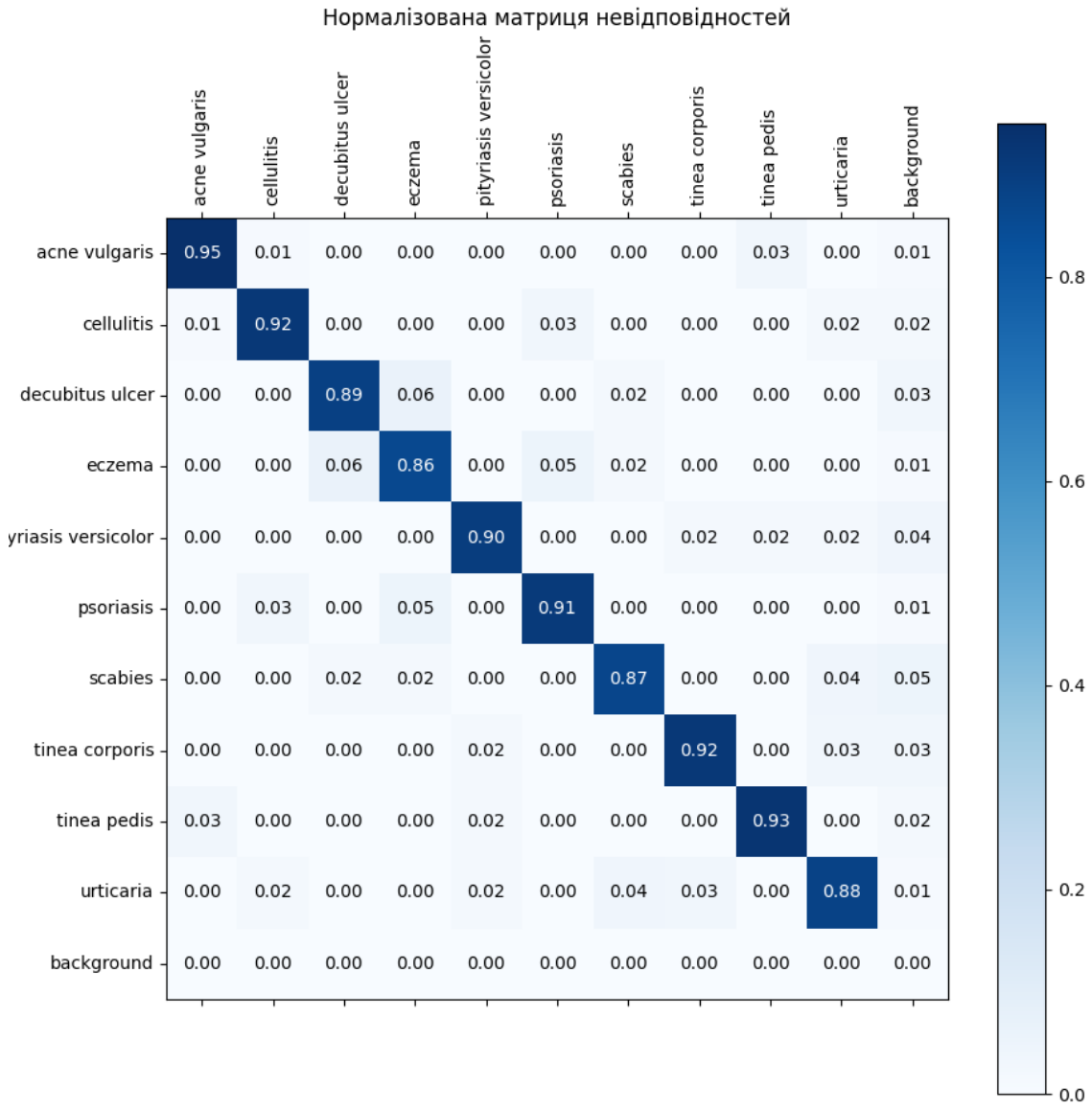

Рисунок 3.14 — Нормалізована матриця невідповідностей для тестового набору даних

Наступний крок у виконанні завдання полягає у створенні чат-бота в Telegram і здобутті необхідного токену.

Спочатку слід створити бота через @BotFather (див. рис. 3.15).

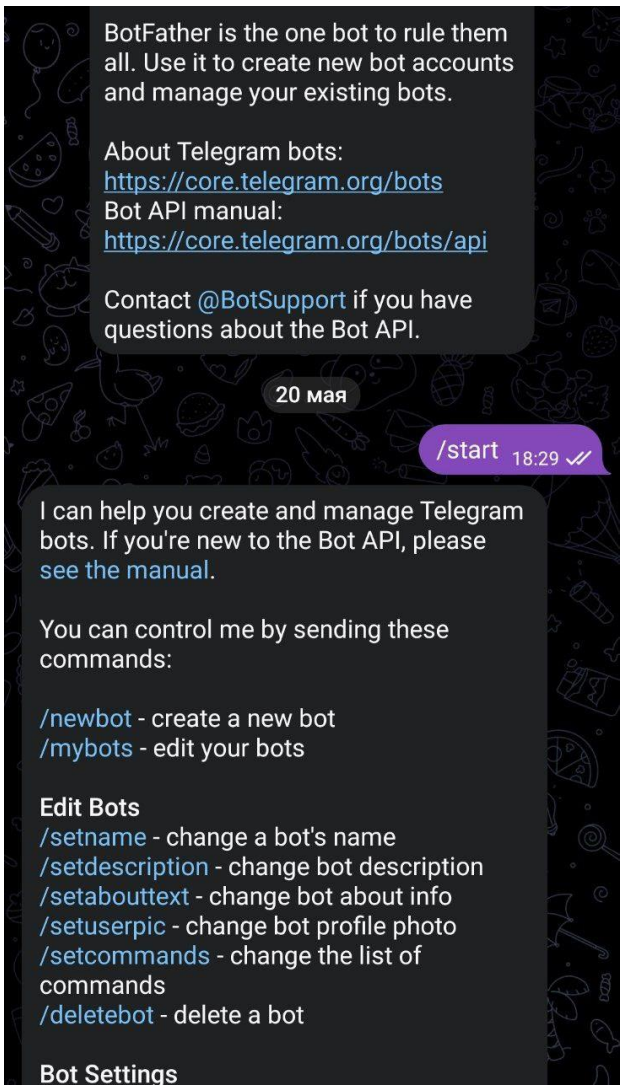

Рисунок 3.15— Старт реєстрації бота

Далі необхідно відправити команду «/newbot» у чаті з @BotFather, щоб зареєструвати нового бота, після чого буде запропоновано ввести назву та ім’я користувача для бота (див. рис. 3.16). Отримавши токен для Telegram, можна приступити до розробки програмного коду бота.

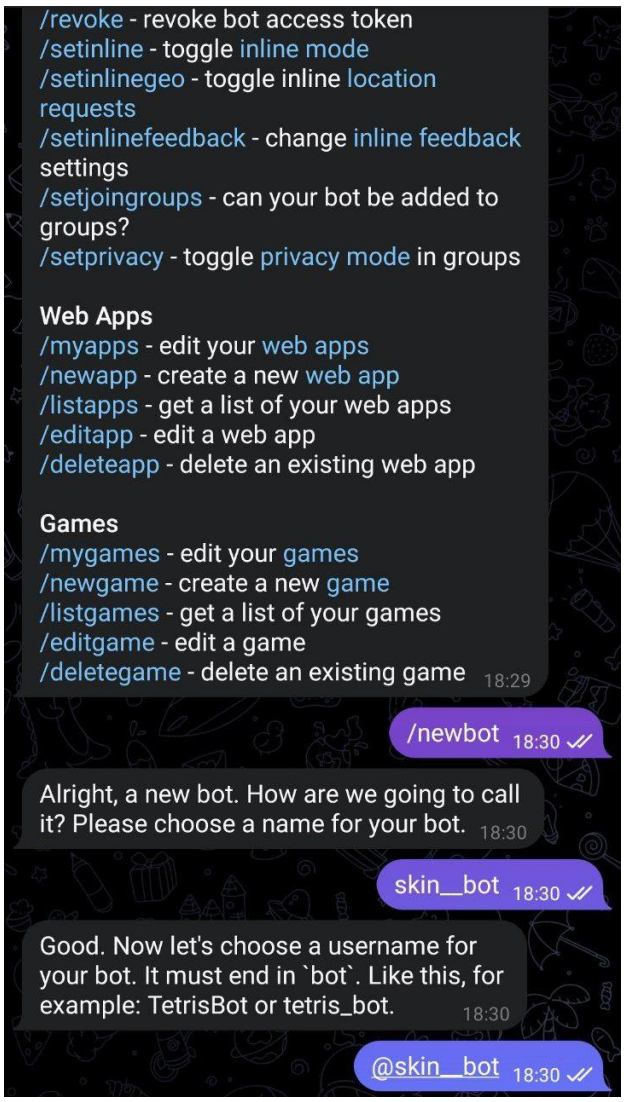

Рисунок 3.16 — Команди для створення бота

Опис основних елементів структури проєкту (рис. 3.17):

– dataset: Папка, що містить набори даних, які використовуються для тренування чи тестування моделей.

– images: Папка для зберігання зображень, які можуть використовуватися для обробки моделями.

– images_pred: Папка, де зберігаються зображення після їх обробки та класифікації моделями.

– models (mobilesam.decoder.onnx, mobilesam.encoder.onnx): Файли onnx, що містять серіалізовані моделі енкодера та декодера, що використовуються для інференції.

– model_classifier.pkl: Серіалізована модель класифікатора, збережена у форматі pickle.

– test: Папка для тестувальних скриптів або даних.

– tg_bot/: Ця папка містить всі файли та модулі, пов'язані з ботом.

– bot_main.py: Основний скрипт для запуску бота та сервісу розпізнавання шкірних захворювань.

– utils.py: Допоміжні утиліти, що використовуються різними частинами проєкту.

– config.txt: Конфігураційний файл для налаштувань бота.

– requirements.txt: Файл, який містить список залежностей Python для проєкту.

– main_project_file.py: Головний файл проєкту для імпорту та запуску бота.

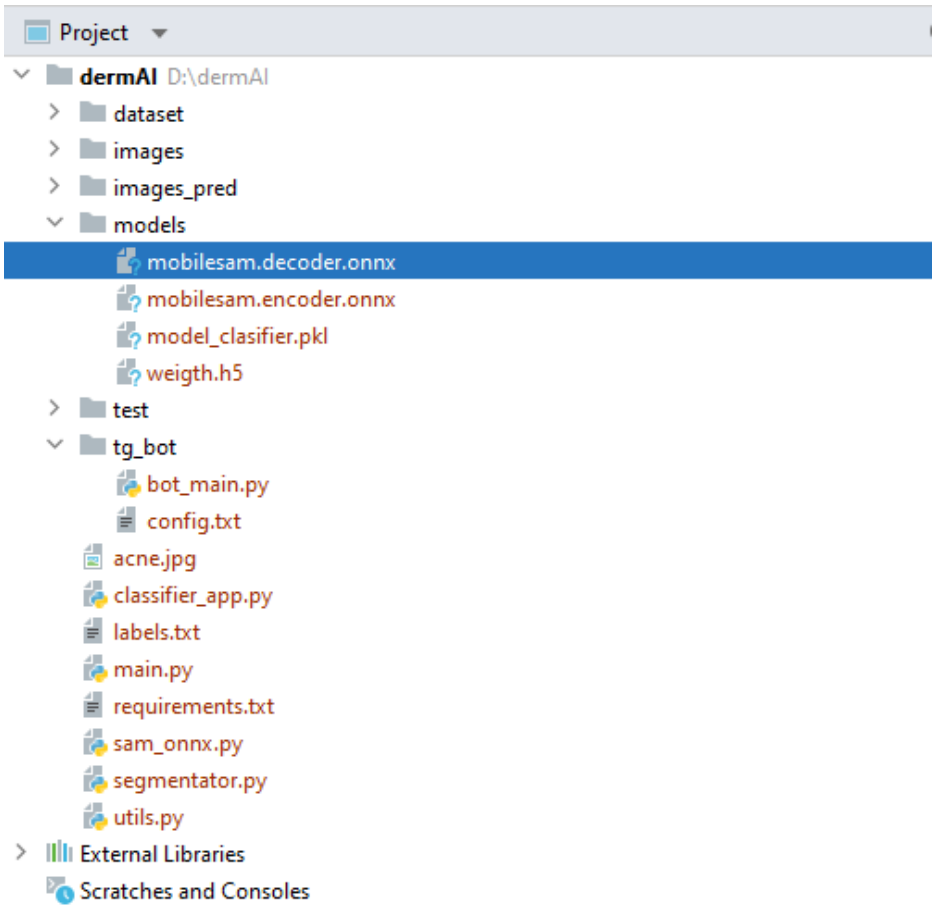

Рисунок 3.17 — Структура проєкту

Для полегшення взаємодії з API Telegram було використано бібліотеку «telebot», інстальовану через pip:

```
pip install pyTelegramBotAPI
```

Pip — це менеджер пакетів для Python, що спрощує процес встановлення та управління бібліотеками у Python-проектах. Акронім Pip означає «Pip Installs Packages» або «Pip Installs Python».

Деякі з основних команд pip:

- `pip install package_name`: встановлення пакету;
- `pip uninstall package_name`: видалення пакету;
- `pip freeze`: показ списку всіх встановлених пакетів з їхніми версіями;
- `pip install -r requirements.txt`: встановлення пакетів з файлу requirements.txt.

Наступним кроком є створення файлу main.py і написання коду для ініціації взаємодії з Telegram API:

```
TOKEN = '7124592295:AAFIe66iIrgq5L1ZKPClDKZEBPzmeylWsLA'

def main() -> None:
    updater = Updater(TOKEN)

    dispatcher = updater.dispatcher
    job_queue = updater.job_queue

    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(MessageHandler(Filters.photo, process_image))

    # Запуск бота
    updater.start_polling()

    updater.idle()

if __name__ == '__main__':
    main()
```

Після активації скрипта, бот реагуватиме на команду «/start» (див. рис. 3.18).

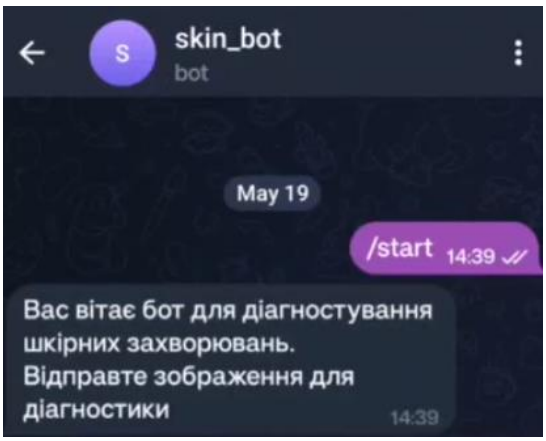

Рисунок 3.18 — Реакція на команду запуску

Скрипт для отримання фото від користувача та його подальшої обробки:

```
def process_image(update: Update, context: CallbackContext) -> None:
    if update.message.photo:
        photo_file = update.message.photo[-1].get_file()
        photo_bytes = io.BytesIO(photo_file.download_as_bytearray())
        image = Image.open(photo_bytes)
        time.sleep(1)

        context.bot.send_message(chat_id=update.message.chat_id, text =
"Обробка зображення")
        segments, boxes = segmentation(image)
        image_with_segments = draw_segments(image, segments)
        context.bot.send_photo(chat_id=update.message.chat_id,
photo=image_with_segments, caption='Сегментація зображення')

        valid_boxex = []
        for box in boxes:
            prediction = classify_box(model, image, box)
            prob = prediction[0][0]
            label = prediction[0][1]
            if prob >  0.7:
                valid_boxex.append(box)
                draw_boxes_and_labels(image, box, label)

        context.bot.send_photo(chat_id=update.message.chat_id, photo=image,
caption='Класифікація сегментів')

        union_boxes  = non_max_suppression(valid_boxex)
        for box in union_boxes:

            client = Client("http://127.0.0.1:7860/")
            result = client.predict(
                img=image[box[1]:box[1]+box[3], box[0]:box[0]+ box[2], :],
                api_name="/predict"
            )
            context.bot.send_message(chat_id=update.message.chat_id,
text=result)
```

Повний код класу для завантаження та запуску моделі сегментації та алгоритм роботи Telegram бота представлені в Додатках Б-В.

3.3 Використання програмного додатку

На стадії тестування основна увага приділяється оцінюванню та перевірці робочих функцій чат-бота. Тестування є критично важливим етапом у розробці, адже воно допомагає переконатися, що бот працює належним чином та відповідає встановленим критеріям.

У цьому розділі представлено результати тестування, які можна переглянути на рисунках 3.19-3.24. Процес тестування дозволяє виявити потенційні проблеми та покращити якість чат-бота перед його запуском.

Тестування боту при коректних даних:

1. Запуск бота (див. рис. 3.19).

2. Відправка боту коректного зображення шкіри. Після чого бот надішле сегментоване зображення, де окремі ділянки шкіри виділені різнокольоровими рамками (див. рис. 3.20), та процес класифікації сегментів (див. рис. 3.21).

Результати класифікації цих сегментів з вказівкою на діагноз та вірогідність наведено на рисунку 3.22. Наприклад, для однієї з виділених ділянок встановлено діагноз «псоріаз» з вірогідністю 0.971, що свідчить про високу впевненість у цьому висновку.

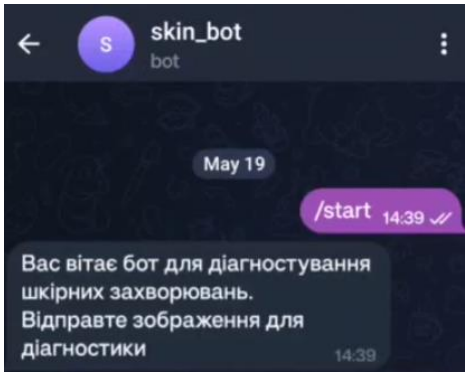

Рисунок 3.19 — Запуск бота

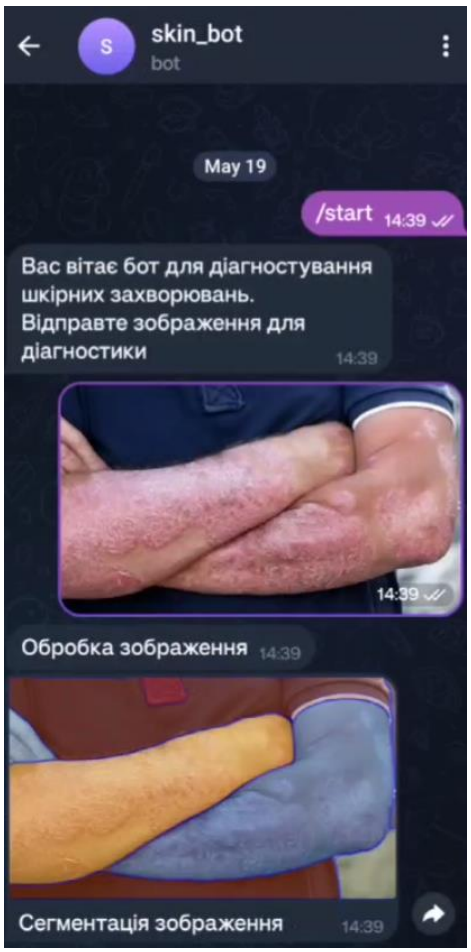

Рисунок 3.20 — Результат сегментації коректного зображення

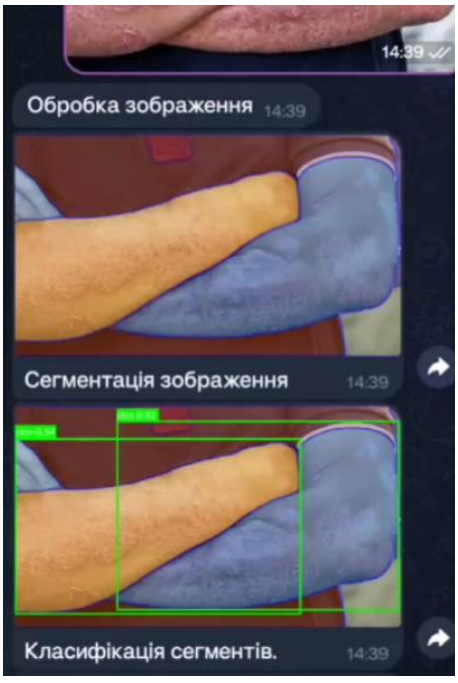

Рисунок 3.21 — Результат класифікації сегментів

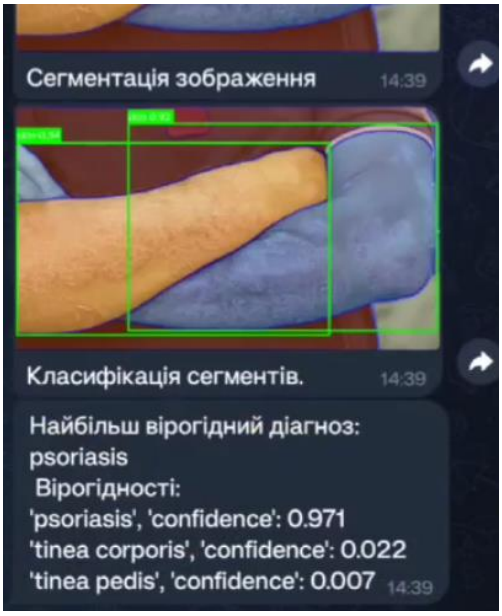

Рисунок 3.22 — Виведення результату діагностики

У випадку відправки зображення, що не містить ділянки шкіри, або файлів невірного формату, користувач отримає повідомлення про неможливість визначення захворювання (рис. 3.23-3.24).

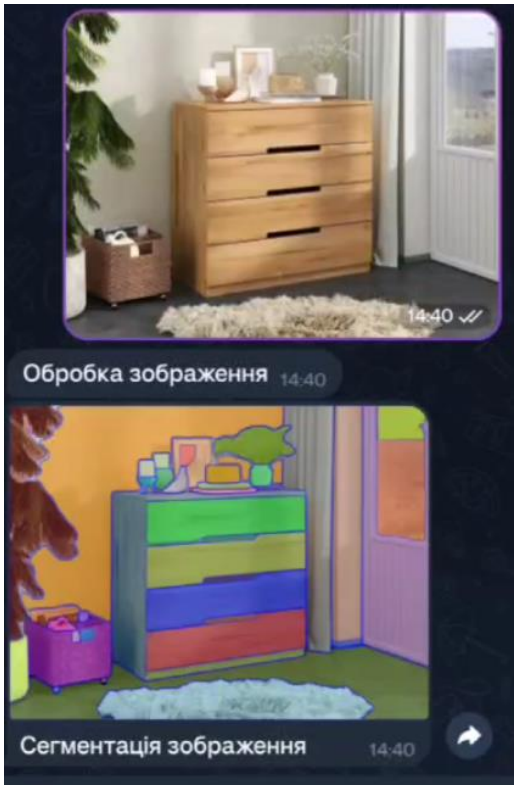

Рисунок 3.23 — Сегментація некоректного зображення

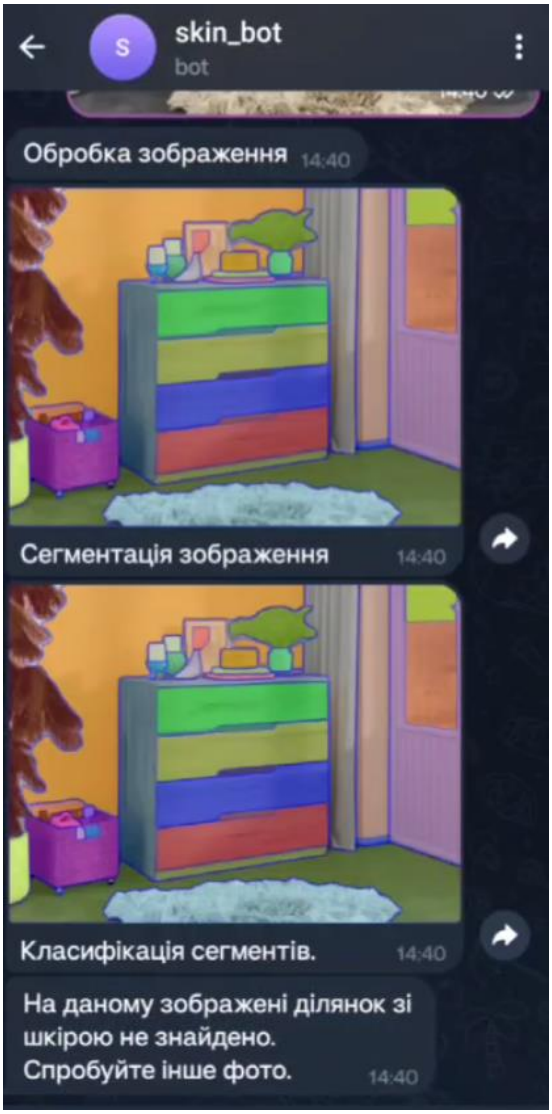

Рисунок 3.24 — Класифікація сегментів некоректного зображення

ВИСНОВКИ

У результаті виконання дипломного проєкту було досягнуто важливих результатів у розробці системи, що дозволяє користувачам ефективно здійснювати первинну діагностику шкірних захворювань. А саме:

- визначено основні напрямки дослідження, що включають об'єкти та предмети дослідження;
- проаналізовано актуальність розробки на основі існуючої проблематики в діагностиці шкірних захворювань;
- проведено аналіз аналогічних систем, виявлено їхні недоліки та переваги, що дало змогу визначити ключові напрямки для покращення власної системи;
- встановлено функціональні вимоги до чат-бота, зокрема, завантаження фотографій, детектування та класифікація уражень, відображення результатів діагностики;
- обрано програмні засоби та методики машинного навчання для реалізації чат-бота, включаючи застосування глибоких нейронних мереж та алгоритмів штучного інтелекту.

Програмна реалізація чат-боту була успішно виконана з використанням мови програмування Python, що дозволило створити ефективний інструмент для первинної діагностики шкірних захворювань на основі зображень.

Основною технічною складовою розробки була інтеграція технологій машинного навчання. Для детектування та класифікації шкірних уражень використано модель на основі Faster Segment Anything для сегментації та простий бінарний класифікатор на основі MobileNetV2 для ідентифікації типів захворювань. Ці технології дозволили чат-боту точно та швидко обробляти вхідні дані.

Завершення розробки та імплементації чат-боту відкриває нові перспективи для діагностики шкірних захворювань, пропонуючи зручний та

доступний спосіб для користувачів отримувати первинні діагностичні висновки. Проєкт підкреслює значення машинного навчання у сучасній медицині та може слугувати основою для подальших досліджень та розробок в цій області.

У подальшому, вдосконалення чат-бота може включати розширення бази даних для тренування, збільшення точності діагностики та покращення інтерактивних можливостей, що забезпечить ще більшу інтеграцію та користь від використання системи. Цей проєкт демонструє значний потенціал машинного навчання у вирішенні практичних медичних задач.

СПИСОК ЛІТЕРАТУРИ

1. Recent Advancements and Perspectives in the Diagnosis of Skin Diseases Using Machine Learning and Deep Learning: A Review [Електронний ресурс]. — Режим доступу: https://pubmed.ncbi.nlm.nih.gov/38066747/ – (дата звернення 04.03.2024) — Назва з титулу екрана.

2. Psoriasis, Vitiligo, and Biologic Therapy: Case Report and Narrative Review/ / Бурландо, М.; Мураччіолі, А.; Коццані, Е.; Пароді, А. — Клінічний випадок дерматології. 2021, 13, сторінки 372–378.

3. Що таке ШІ? Застосування штучного інтелекту в дерматології. / Ду-Харпур, Х.; Ватт, Ф.М.; Ласкомб, Н.М.; Лінч, М.Д. — Британський журнал дерматології. 2020, 183, 423–430.

4. Сільверберг, Н.Б. Епідеміологія вітиліго. Поточний дерматологічний звіт. 2015, 4, 36–43.

5. Пател, С.; Ванг, Дж.В.; Мотапарті, К.; Лі, Дж.Б. Штучний інтелект у дерматології для клініцистів. Клінічна дерматологія. 2021, 39, 667–672.

6. Срідхар, Б.; Бе, М.С.; Кумар, М.С. Порівняльне дослідження виявлення раку шкіри меланоми за допомогою традиційних і сучасних методів обробки зображень. У матеріалах Четвертої міжнародної конференції 2020 року з I-SMAC (IoT у соціальних, мобільних, аналітичних та хмарних мережах) (I-SMAC), Палладам, Індія, 7–9 жовтня 2020 р.; IEEE: Нью-Йорк, Нью-Йорк, США, 2020; с. 654–658.

7. Вступ до сегментації зображень за допомогою кластеризації K-Means [Електронний ресурс]. — Режим доступу: https://www.kdnuggets.com/2019/08/introduction-image-segmentation-k-means-clustering.html – (дата звернення 04.03.2024) — Назва з титулу екрана.

8. Introduction to Image Segmentation with K-Means clustering [Електронний ресурс]. — Режим доступу: https://www.kdnuggets.com/2019/08/introduction-image-segmentation-k-means-

clustering.html – (дата звернення 04.03.2024) — Назва з титулу екрана.

9. Згорткові нейронні мережі (CNN) у машинному навчанні [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/. – (дата звернення 04.03.2024) — Назва з титулу екрана.

10. U-Net Architecture Explained [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/u-net-architecture-explained/?ref=header_search – (дата звернення 04.03.2024) — Назва з титулу екрана.

11. SegNET [Електронний ресурс]. — Режим доступу: https://medium.com/@saba99/segnet-a139ce77b570 – (дата звернення 04.03.2024) — Назва з титулу екрана.

12. DeepLab [Електронний ресурс]. — Режим доступу: https://paperswithcode.com/method/deeplab – (дата звернення 04.03.2024) — Назва з титулу екрана.

13. Алгоритм машини опорних векторів (SVM) [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ – (дата звернення 04.03.2024) — Назва з титулу екрана.

14. Random Forest Algorithm [Електронний ресурс]. — Режим доступу: https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm – (дата звернення 04.03.2024) — Назва з титулу екрана.

15. Огляд MobileNet: Ефективна згорткова нейронна мережа для мобільних пристроїв [Електронний ресурс]. — Режим доступу: https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d – (дата звернення 04.03.2024) — Назва з титулу екрана.

16. Understanding EfficientNet — The most powerful CNN architecture [Електронний ресурс]. — Режим доступу: https://arjun-sarkar786.medium.com/understanding-efficientnet-the-most-powerful-cnn-

architecture– (дата звернення 04.03.2024) — Назва з титулу екрана.

17. Understanding VGG Neural Networks: Architecture and Implementation [Електронний ресурс]. — Режим доступу: https://thegrigorian.medium.com/understanding-vgg-neural-networks-architecture-and-implementation-400d99a9e9ba – (дата звернення 04.03.2024) — Назва з титулу екрана.

18. Building Telegram Bots: Develop Bots in 12 Programming Languages using the Telegram Bot API / Nicolas Modrzyk — Apress, December 2018.

19. Python Machine Learning: Unlock modern machine learning and deep learning techniques with Python / Sebastian Raschka, Vahid Mirjalili — Packt Publishing, Third Edition 2019. — 770 с.

20. Fluent Python: Clear, Concise, and Effective Programming / Luciano Ramalho — O'Reilly Media, Second Edition 2021. — 500 с.

21. Python Machine Learning - Third Edition: Unlock modern machine learning and deep learning techniques with Python / Sebastian Raschka, Vahid Mirjalili — Packt Publishing, December 2019. — 772 с.

22. FASTER SEGMENT ANYTHING: TOWARDS LIGHTWEIGHT SAM FOR MOBILE APPLICATIONS [Електронний ресурс]. — Режим доступу: https://arxiv.org/pdf/2306.14289

ДОДАТКИ

Додаток А

Код для трансферу вагових коефіцієнтів, створення моделі класифікатору, налаштування параметрів навчання моделі класифікації шкірних захворювань.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Гіперпараметри
num_classes = 10
batch_size = 32
learning_rate = 0.0005
num_epochs = 30
input_shape = (224, 224, 3)

train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=15,
    width_shift_range=0.22,
    height_shift_range=0.21,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    'data/test',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)

base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=input_shape, alpha=0.5)

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(num classes, activation='softmax')(x)


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

for layer in base_model.layers:
    layer.trainable = False


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


model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=test_generator,
    validation_steps=test_generator.samples // batch_size,
    epochs=num_epochs
)

for layer in base_model.layers[-2:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=learning_rate/10),
loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=test_generator,
    validation_steps=test_generator.samples // batch_size,
    epochs=num_epochs // 2
)

loss, accuracy = model.evaluate(test_generator, steps=test_generator.samples
// batch_size)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

Додаток Б

Реалізація на мові Python створеного телеграм боту

```
import io
import logging
from utils import draw_segments, classify_box, draw_boxes_and_labels,
non_max_suppression
from telegram import Update
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters,
CallbackContext, JobQueue
import time
from ..segmentator import segmentation
import tensorflow as tf
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2,
preprocess_input, decode_predictions
from PIL import Image, ImageDraw
import numpy as np
from gradio_client import Client, file


model = MobileNetV2(weights='imagenet')
weights_path = 'models/weigth.h5'
model.load_weights(weights_path)



logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)

logger = logging.getLogger(__name__)

# Команда /start
def start(update: Update, context: CallbackContext) -> None:
    update.message.reply_text('Вас вітає бот для діагностування шкірних
захворювань.\nВідправте зображення для діагностики')


def process_image(update: Update, context: CallbackContext) -> None:
    if update.message.photo:
        photo_file = update.message.photo[-1].get_file()
        photo_bytes = io.BytesIO(photo_file.download_as_bytearray())
        image = Image.open(photo_bytes)
        time.sleep(1)

        context.bot.send_message(chat_id=update.message.chat_id, text =
"Обробка зображення")
        segments, boxes = segmentation(image)
        image_with_segments = draw_segments(image, segments)
        context.bot.send_photo(chat_id=update.message.chat_id,
photo=image_with_segments, caption='Сегментація зображення')

        valid_boxex = []
        for box in boxes:
            prediction = classify_box(model, image, box)
            prob = prediction[0][0]
            label = prediction[0][1]
```

```
            if prob >  0.7:
                valid_boxex.append(box)
                draw_boxes_and_labels(image, box, label)

        context.bot.send_photo(chat_id=update.message.chat_id, photo=image,
caption='Класифікація сегментів')

        union_boxes  = non_max_suppression(valid_boxex)
        for box in union_boxes:

            client = Client("http://127.0.0.1:7860/")
            result = client.predict(
                img=image[box[1]:box[1]+box[3], box[0]:box[0]+ box[2], :],
                api_name="/predict"
            )
            context.bot.send_message(chat_id=update.message.chat_id,
text=result)


TOKEN = '7124592295:AAFIe66iIrgq5L1ZKPClDKZEBPzmeylWsLA'

def main() -> None:
    updater = Updater(TOKEN)

    dispatcher = updater.dispatcher
    job_queue = updater.job_queue

    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(MessageHandler(Filters.photo, process_image))

    # Запуск бота
    updater.start_polling()

    updater.idle()

if __name__ == '__main__':
    main()
```

Додаток В

Реалізація на мові Python класу для завантаження та запуску моделі сегментації.

```
import logging

logging.basicConfig(level=logging.DEBUG)

from copy import deepcopy

import cv2
import numpy as np
import onnxruntime


class SegmentAnythingONNX:
    """Segmentation model using SegmentAnything"""

    def __init__(self, encoder_model_path, decoder_model_path) -> None:
        self.target_size = 1024
        self.input_size = (684, 1024)

        # Load models
        providers = onnxruntime.get_available_providers()
        providers = [p for p in providers if p !=
"TensorrtExecutionProvider"]

        if providers:
            logging.info(
                "Available providers for ONNXRuntime: %s", ",
".join(providers)
            )
        else:
            logging.warning("No available providers for ONNXRuntime")
        self.encoder_session = onnxruntime.InferenceSession(
            encoder_model_path, providers=providers
        )
        self.encoder_input_name = self.encoder_session.get_inputs()[0].name
        self.decoder_session = onnxruntime.InferenceSession(
            decoder_model_path, providers=providers
        )

    def get_input_points(self, prompt):
        """Get input points"""
        points = []
        labels = []
        for mark in prompt:
            if mark["type"] == "point":
                points.append(mark["data"])
                labels.append(mark["label"])
            elif mark["type"] == "rectangle":
                points.append([mark["data"][0], mark["data"][1]])  # top left
                points.append(
                    [mark["data"][2], mark["data"][3]]
                )  # bottom right
                labels.append(2)
                labels.append(3)
        points, labels = np.array(points), np.array(labels)
        return points, labels
```

```
 def run_encoder(self, encoder_inputs):
    """Run encoder"""
    output = self.encoder_session.run(None, encoder_inputs)
    image_embedding = output[0]
    return image_embedding

@staticmethod
def get_preprocess_shape(oldh: int, oldw: int, long_side_length: int):
    """
    Compute the output size given input size and target long side length.
    """
    scale = long_side_length * 1.0 / max(oldh, oldw)
    newh, neww = oldh * scale, oldw * scale
    neww = int(neww + 0.5)
    newh = int(newh + 0.5)
    return (newh, neww)

def apply_coords(self, coords: np.ndarray, original_size, target_length):

    old_h, old_w = original_size
    new_h, new_w = self.get_preprocess_shape(
        original_size[0], original_size[1], target_length
    )
    coords = deepcopy(coords).astype(float)
    coords[..., 0] = coords[..., 0] * (new_w / old_w)
    coords[..., 1] = coords[..., 1] * (new_h / old_h)
    return coords


 def run_decoder(
     self, image_embedding, original_size, transform_matrix, prompt
 ):
     """Run decoder"""
     input_points, input_labels = self.get_input_points(prompt)

     # Add a batch index, concatenate a padding point, and transform.
     onnx_coord = np.concatenate(
         [input_points, np.array([[0.0, 0.0]])], axis=0
     )[None, :, :]
     onnx_label = np.concatenate([input_labels, np.array([-1])], axis=0)[
         None, :
     ].astype(np.float32)
     onnx_coord = self.apply_coords(
         onnx_coord, self.input_size, self.target_size
     ).astype(np.float32)
     onnx_coord = np.concatenate(
         [
             onnx_coord,
             np.ones((1, onnx_coord.shape[1], 1), dtype=np.float32),
         ],
         axis=2,
     )
     onnx_coord = np.matmul(onnx_coord, transform_matrix.T)
     onnx_coord = onnx_coord[:, :, :2].astype(np.float32)

     # Create an empty mask input and an indicator for no mask.
     onnx_mask_input = np.zeros((1, 1, 256, 256), dtype=np.float32)
     onnx_has_mask_input = np.zeros(1, dtype=np.float32)

     decoder_inputs = {
         "image_embeddings": image_embedding,
```

```
            "point_coords": onnx_coord,
            "point_labels": onnx_label,
            "mask_input": onnx_mask_input,
            "has_mask_input": onnx_has_mask_input,
            "orig_im_size": np.array(self.input_size, dtype=np.float32),
        }
        masks, _, _ = self.decoder_session.run(None, decoder_inputs)

        # Transform the masks back to the original image size.
        inv_transform_matrix = np.linalg.inv(transform_matrix)
        transformed_masks = self.transform_masks(
            masks, original_size, inv_transform_matrix
        )

        return transformed_masks

    def transform_masks(self, masks, riginal_size, transform_matrix):
        """Transform masks
        Transform the masks back to the original image size.
        """
        output_masks = []
        for batch in range(masks.shape[0]):
            batch_masks = []
            for mask_id in range(masks.shape[1]):
                mask = masks[batch, mask_id]
                mask = cv2.warpAffine(
                    mask,
                    transform_matrix[:2],
                    (original_size[1], original_size[0]),
                    flags=cv2.INTER_LINEAR,
                )
                batch_masks.append(mask)
            output_masks.append(batch_masks)
        return np.array(output_masks)


def encode(self, cv_image):
    """
    Calculate embedding and metadata for a single image.
    """
    original_size = cv_image.shape[:2]

    # Calculate a transformation matrix to convert to self.input_size
    scale_x = self.input_size[1] / cv_image.shape[1]
    scale_y = self.input_size[0] / cv_image.shape[0]
    scale = min(scale_x, scale_y)
    transform_matrix = np.array(
        [
            [scale, 0, 0],
            [0, scale, 0],
            [0, 0, 1],
        ]
    )
    cv_image = cv2.warpAffine(
        cv_image,
        transform_matrix[:2],
        (self.input_size[1], self.input_size[0]),
        flags=cv2.INTER_LINEAR,
    )

    encoder_inputs = {
```

```
            self.encoder_input_name: cv_image.astype(np.float32),
        }
        image_embedding = self.run_encoder(encoder_inputs)
        return {
            "image_embedding": image_embedding,
            "original_size": original_size,
            "transform_matrix": transform_matrix,
        }

    def predict_masks(self, embedding, prompt):
        """
        Predict masks for a single image.
        """
        masks = self.run_decoder(
            embedding["image_embedding"],
            embedding["original_size"],
            embedding["transform_matrix"],
            prompt,
        )

        return masks
```

Додаток Г

Код із файлу utils.py для допоміжних функцій.

```
from PIL import Image, ImageDraw
import random
import tensorflow as tf
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2,
preprocess_input, decode_predictions
from PIL import Image, ImageDraw
import numpy as np

def draw_segments(image, masks):
    image = image.convert("RGBA")
    combined = image
    for mask in masks:
        mask = mask.convert("L")
        overlay = Image.new('RGBA', image.size, (0, 0, 0, 0))

        mask_np = np.array(mask)
        unique_segments = np.unique(mask_np)
        segment_colors = {segment: (random.randint(0, 255), random.randint(0,
255), random.randint(0, 255), 100)
                          for segment in unique_segments}

        for segment, color in segment_colors.items():
            overlay_np = np.zeros((mask_np.shape[0], mask_np.shape[1], 4),
dtype=np.uint8)
            overlay_np[mask_np == segment] = color
            overlay = Image.alpha_composite(overlay,
Image.fromarray(overlay_np))
            combined = Image.alpha_composite(image, overlay)

    return combined

def classify_box(model, image_np, box):

    ymin, xmin, ymax, xmax = box
    crop_img = image_np[int(ymin):int(ymax), int(xmin):int(xmax), :]
    crop_img = Image.fromarray(crop_img).resize((224, 224))
    crop_img = np.expand_dims(crop_img, axis=0)
    crop_img = preprocess_input(np.array(crop_img))
    preds = model.predict(crop_img)
    return decode_predictions(preds, top=3)[0][0]

def draw_boxes_and_labels(image, boxes, labels):

    draw = ImageDraw.Draw(image)
    for box, label in zip(boxes, labels):
        ymin, xmin, ymax, xmax = box
        left, top, right, bottom = xmin, ymin, xmax, ymax
        draw.rectangle([left, top, right, bottom], outline="green", width=3)
        draw.text((left, top), f"{label[1]}: {label[2]:.2f}", fill="white")

def classify_and_draw_boxes(model, image, boxes):
    labels = [classify_box(model, image, box) for box in boxes]
    draw_boxes_and_labels(image, boxes, labels)
    return image


def iou(box1, box2):
```

```
    ymin1, xmin1, ymax1, xmax1 = box1
    ymin2, xmin2, ymax2, xmax2 = box2

    inter_xmin = max(xmin1, xmin2)
    inter_ymin = max(ymin1, ymin2)
    inter_xmax = min(xmax1, xmax2)
    inter_ymax = min(ymax1, ymax2)

    inter_area = max(0, inter_xmax - inter_xmin) * max(0, inter_ymax -
inter_ymin)
    box1_area = (xmax1 - xmin1) * (ymax1 - ymin1)
    box2_area = (xmax2 - xmin2) * (ymax2 - ymin2)

    iou = inter_area / float(box1_area + box2_area - inter_area)
    return iou

def non_max_suppression(boxes, labels=None, iou_threshold=0.5):
    if len(boxes) == 0:
        return [], []
    labels =  np.ones(len(boxes))
    indices = list(range(len(boxes)))
    indices.sort(key=lambda i: labels[i][2], reverse=True)

    selected_boxes = []
    selected_labels = []

    while indices:
        current = indices.pop(0)
        selected_boxes.append(boxes[current])
        selected_labels.append(labels[current])

        indices = [i for i in indices if iou(boxes[current], boxes[i]) <
iou_threshold]

    return selected_boxes, selected_labels
```