Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ РАННЬОГО ВИЯВЛЕННЯ ДЕРМАТОЛОГІЧНИХ ЗАХВОРЮВАНЬ

Текстова частина до курсової роботи

за спеціальністю „Комп’ютерні науки ” 122

Керівник курсової роботи

с.в. _Франків О.О.__

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2024 р.

Виконала студентка _

3-го року навчання_

___Кучина Є.В._____

(прізвище та ініціали)

“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

Календарний план виконання курсової роботи

Тема: Розробка мобільного застосунку для раннього виявлення дерматологічних захворювань

Календарний план виконання курсової роботи:

№ п/п	Назва етапу роботи	Термін виконання етапу	Примітка
1	Отримання завдання на курсову роботу	01.10.2023	
2	Аналіз матеріалів за темою	01.10.2023 -14.10.2023	
3	Розробка та програмування програми	25.12.2023 – 10.03.2024	
4	Написання текстової частини до курсової роботи	25.03.2024 – 02.05.2024	
5	Коригування виконаної роботи	07.05.2024	
6	Створення слайдів для доповіді та написання доповіді.	10.05.2024 – 12.05.2024	
7	Остаточне оформлення роботи та слайдів	17.05.2024	
8	Захист курсової роботи	21.05.2024	

Кучина Є. В. _____

Франків О. О. _____

“_____”_____

Зміст

ВСТУП

Останнім часом все більше й більше людей стикаються зі шкірними хворобами. Однією з найпоширеніших є так зване акне, або ж іншими словами – вугрові висипання. Найчастіше воно турбує людей на обличчі, проте висипання бувають також на спині та грудній клітині. Проте дана хвороба є не лише неприємною, а й дійсно небезпечною, бо може призводити до зниження самооцінки та навіть клінічної депресії у пацієнтів. Дана хвороба може ментально змінити людину й додати безліч комплексів.[9] Тому люди витрачають великі кошти аби увійти в ремісію й покращити стан своєї шкіри. Як результат, є реакція на запити суспільства у вигляді великої кількості інформації в мережі Інтернет про методи професійного та домашнього лікування хвороби шкіри. Більше того, з розвитком сфери машинного навчання та комп'ютерного зору, набувають популярності різного виду додатки, які можуть надавати людям рекомендації по догляду за шкірою та лікуванню, визначати типи шкіри та наявні хвороби на ній.

Як і з усіма хворобами, у ефективному лікуванні дуже допомагає раннє виявлення акне. Чим швидше ви розумієте з чим маєте справу – тим легше буде впоратися з висипами.

Об'єктом дослідження курсової роботи є процес раннього виявлення акне на шкірі обличчя за допомогою IOS додатку, використовуючи згорткові нейронні мережі (CNN).

Предметом дослідження є інструменти для створення та імплементації згорткової нейронної мережі у IOS додаток та методи забезпечення конфіденційності даних користувачів.

Метою дослідження курсової роботи є створення програмного додатку, який буде допомагати людині виявити шкірну хворобу на ранніх стадіях й

відслідковувати її розвиток та бути проінформованим, якщо акне прогресує, або ж, навпаки, ситуація стає ліпшою.

Для реалізації мети курсової роботи потрібно виконати наступні завдання:

- зробити аналіз програмних додатків з класифікацією зображень хвороб шкіри
- побудувати модель нейронної мережі по розпізнаванню ступеню складності Акне
- вирішити проблему конфіденційності користувача й безпеки його даних
- забезпечити фонову роботу додатку
- розробити процес виявлення тенденції розвитку хвороби

Результатом дослідження буде програмний додаток, у складі якого буде модель, реалізована на базі згорткової нейронної мережі, який буде надавати змогу користувачам бути в курсі будь яких змін у стані їхньої шкіри, не вимагаючи від них ніяких зусиль й при цьому буде забезпечувати конфіденційність особистих чутливих даних.

РОЗДІЛ 1. Аналіз предметної області та підходу до реалізації програмного продукту

1.1. Огляд проблеми

Створення додатку для раннього виявлення дерматологічних захворювань потребує уваги щодо деяких аспектів, без котрих його функціонування в реальному світі є неможливим. Ці проблеми повинні бути вирішені, щоб забезпечити ефективність додатку та визнання користувачами.

По-перше, питання конфіденційності користувача є першочерговим. Враховуючи чутливий характер особистих медичних даних користувача, забезпечення того, щоб вся інформація про власника додатка залишалася приватною та безпечною, є надзвичайно важливим. Це передбачає використання таких підходів до реалізації, які дозволять захистити особисту інформацію від несанкціонованого доступу або ж забезпечити невитікання інформації взагалі.

Іншим важливим аспектом є забезпечення ефективної роботи у фоні. Додаток повинен коректно працювати у фоновому режимі, щоб забезпечити безперебійну роботу по розпізнаванню хвороби шкіри. Це означає, що програма повинна вміти непомітно аналізувати дані, не порушуючи робочий процес користувача і при цьому не виснажуючи ресурси телефону. Потрібно забезпечити сканування та аналізування зображень не спричиняючи надмірного навантаження на продуктивність пристрою.

Третім аспектом є визначення тенденції в прогресуванні дерматологічних захворювань, а саме - акне. Це передбачає виявлення закономірностей, таких як поява постійних висипань або погіршення симптомів з плином часу. Розпізнаючи ці тенденції, додаток може надавати користувачам своєчасні попередження та рекомендації щодо звернення за

медичною допомогою, що потенційно може призвести до більш ранньої діагностики та лікування шкірних захворювань.

1.2. Аналіз існуючих додатків для розпізнавання хвороби шкіри за зображенням

Протягом останнього часу з’явилося багато нових шляхів діагностики та лікування шкіри. Технологічні рішення також не стоять на місці. Мобільні додатки, зокрема, стали потужним інструментом, що дозволяє людям активно контролювати своє здоров'я. У сфері дерматології, де раннє виявлення та своєчасне втручання є критично важливими, дуже ефективним рішенням є розробка мобільних додатків для розпізнавання шкірних захворювань за допомогою аналізу зображень.

Шкірні захворювання вражають людей різного віку та демографічної приналежності. Спектр дерматологічних розладів широкий і різноманітний - від звичайних захворювань, таких як акне та екзема, до більш серйозних недуг, таких як меланома і псоріаз. Однак доступ до своєчасної і точної діагностики та лікування цих захворювань може бути складним, особливо в громадах з недостатнім рівнем обслуговування або в регіонах з обмеженим доступом до ресурсів охорони здоров'я.

На цьому тлі мобільні додатки пропонують багатообіцяюче рішення, адже маючи телефон людина вже може проаналізувати стан своєї шкіри. Такого плану додатки дозволяють користувачам робити знімки своїх шкірних уражень або станів за допомогою камер смартфонів і отримувати автоматизовані аналізи або рекомендації щодо подальших дій. Надаючи користувачам негайний зворотний зв'язок і практичні поради, ці додатки можуть сприяти

ранньому виявленню захворювань, поліпшенню результатів лікування та загальному зміцненню здоров'я шкіри.

Деякі приклади вже існуючих рішень в даній сфері.

1.2.1 Cureto

"Cureto" – додаток для визначення акне по фотографії й надання рекомендацій. Його реалізовано з використанням гібридного підходу, тобто з використанням глибокої згорткової нейронної мережі (CNN) та обробки природної мови (NLP). Додаток розроблений для класифікації щільності вугрів, чутливості шкіри та ідентифікації специфічних підтипів вугрів, а саме білих вугрів, чорних вугрів, папул, пустул, вузликів та кіст. Cureto не тільки класифікує помітні вугри, але й рекомендує відповідні методи лікування, засновані на їх класифікації, тяжкості та інших демографічних факторах, таких як вік, стать тощо. Отримані результати показують, що для класифікації типу акне точність коливається в межах 90%-95%, а для чутливості шкіри та щільності акне точність коливається в межах 93%-96%.

Додаток працює таким чином: приймає вхідне зображення користувача або опис симптомів користувача, а іноді і те, і інше, і видає на виході інформацію про щільність акне, чутливість шкіри, підтип акне та відповідну рекомендацію щодо лікування, рис. 1.1.

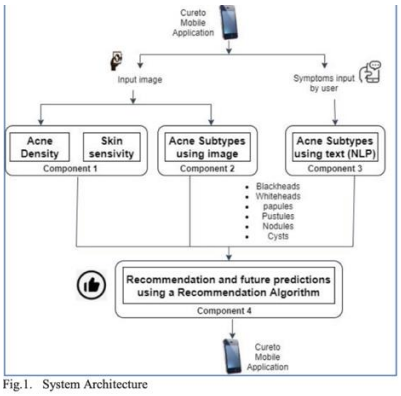

Fig.1. System Architecture

Рис. 1.1

Таким чином, основна реалізація поділена на два етапи: розпізнавання акне за зображенням та за симптомами. Щодо реалізації за зображенням, дослідження робиться лише на видимих змінах шкіри, оскільки отримується діагноз за допомогою зображень. Результати будуть мати 3 рівні: легкий, помірний і важкий, що базується на відсотках результатів, отриманих двома моделями щільності акне і чутливості шкіри.

Класифікація ґрунтується на CNN, оскільки в алгоритмах машинного навчання не відбувається трансферного навчання, тоді як в глибокому навчанні - відбувається. Тому для класифікації використано модель CNN з переносним навчанням на 100 епох з функціями Resnext.

Також були використані методи доповнення даних для покращення функціональності набору даних.

Важливими конфігураціями навчання та параметрами моделі були розмір партії=64, частота падіння=0.5, епохи=100, відсоток перевірки=20%, відсоток тестування=80% та швидкість навчання=0.001.[4]

1.2.2 Detection and Classification of Acne Lesions in Acne Patients: A Mobile Application

Даний додаток також фокусується на виявленні акне по зображенню, робота відбувається за чітким шляхом: підвантаження фото, а потім система визначає наявність акне. Ніяких збережень даних не відбувається, лише визначення в реальному часі.

Автори при розробці дослідили декілька варіантів реалізації: результати продемонстрували, що серед аналізу текстури, кластеризації k-середніх, методи сегментації моделі HSV та дворівнева кластеризація k-середніх, останні перевершили з точністю близько 70%.

Спочатку для сегментації зображення використовується кластеризація K-середніх, яка ділить зображення на регіони на основі схожості інтенсивності пікселів. Цей метод ефективно відокремлює фон (чисту шкіру) від акне, що дозволяє точно ідентифікувати уражену шкіру. Потім застосовується аналіз для характеристики областей на основі їхніх текстурних особливостей, що дозволяє відрізнити гладкі фонові ділянки від текстурованих уражень акне. Крім того, сегментація на основі кольору з використанням моделі HSV ще більше підвищує точність виявлення акне, використовуючи релевантні для сприйняття компоненти кольору. Додаток також реалізує алгоритми машинного навчання для завдань класифікації, включаючи диференціацію між шкірою з акне і без нього за допомогою методу нечітких c-середніх (FCM - метод кластеризації, який дозволяє віднести групу даних до двох або більше кластерів) і розрізнення між рубцями від акне і запальними акне за допомогою машини опорних векторів (SVM) з лінійним ядром.

Загалом, у цьому додатку використовано два методи класифікації на основі машинного навчання. Для SVM, контрольованого методу, використано 10-кратну перехресну перевірку. Середня точність класифікації для розрізнення рубців від запальних акне для методів FCM і лінійного SVM склала 80% і 66,6% відповідно. При класифікації нормальної шкіри від ураженою акне за допомогою методу FCM точність виконання становить 100%.[5]

РОЗДІЛ 2. Використання комп’ютерного зору для розпізнавання ознак акне

2.1. Огляд методів розпізнавання хвороб шкіри за зображенням

Кінець минулого століття можна вважати початком активних досліджень у сфері комп’ютерного зору. Робота Девіда Марра, яка датується 1982 роком, ‘Vision: A Computational Investigation into the Human Representation and Processing of Visual Information’ заклала основу обчислювальної нейронауки та комп’ютерного зору. Одна з головних теоретичних речей, які Марр заклав для подальшого розвитку комп’ютерного зору є розділення задачі нейронауки на три рівні аналізу: теорія обчислень (чітке визначення вхідних і вихідних даних процесу та обмежень, які дозволять розв’язати проблему), алгоритми (більш чітко описує, як перейти від вхідних даних до вихідних) та реалізація (зокрема, як апаратні обмеження впливають на вибір алгоритму, і, навпаки, як алгоритми можуть бути адаптовані відповідно до можливостей апаратного забезпечення).[1]

Таким чином, теорія Марра наголошує на інтеграції теоретичних уявлень про завдання та обмеження, а також на практичних міркуваннях щодо ефективності алгоритму та реалізації апаратного забезпечення. Поєднуючи наукові та статистичні підходи з інженерними принципами, філософія Марра забезпечує міцну основу для формулювання та вирішення проблем у сфері комп’ютерного зору, залишаючись актуальною та цінною протягом багатьох років.

Метод класифікації Байєса

Даний метод - це спосіб визначення ймовірностей того, що об'єкт належить до певного класу, на основі відомих даних. Він базується на теоремі Байєса, яка дозволяє оцінити ймовірності настання подій, враховуючи інформацію про інші події. Перевагою даного методу є те, що він ефективно

керує чіткими та постійними даними, ігноруючи невідповідні характеристики як для бінарних, так і для багатокласових класифікацій. Проте мінусом даної класифікації в рамках розпізнавання вад шкіри по зображенню є те, що вона часто є недостатньою для ймовірнісного моделювання, оскільки вона не підходить для класифікації даних без наглядачів і не має незалежності серед предикторів.[7]

Метод k-найближчих сусідів

Метод k-найближчих сусідів - це спосіб класифікації об'єктів на основі того, наскільки схожі вони на об'єкти з відомими класами. Для даного методу важливо мати набір даних, де для кожного об'єкта відомий його клас. Коли подається новий об'єкт, аналізуються його найближчі сусіди й класи, яким ці сусіди належать. Клас, який найчастіше зустрічається серед сусідів, і стає класом для нового об'єкта. Кількість сусідів для визначення класу визначається параметром k. Мінусом цього методу є те, що точність результатів безпосередньо залежить від стану даних. Крім того, час прогнозування може бути великим для більшого розміру вибірки.[7]

Метод опорних векторів

SVM добре працює, навіть якщо у даних багато функцій або параметрів. Це ефективно, тому що не вимагає багато пам’яті для обробки великих наборів даних із багатьма вимірами. Проте SVM може бути не найкращим вибором для роботи з зашумленими зображеннями - випадковими змінами яскравості або кольорової інформації на зображеннях. Це призводить до помилкових класифікацій або зниження точності виявлення уражень акне. Іншим аспектом, є те, що SVM має гіперпараметри, які необхідно налаштовувати для оптимальної продуктивності. Знайти правильні параметри чи функції для використання під час роботи із зображеннями може бути складно. SVM краще працює зі структурованими даними, а не з неструктурованими, такими як

зображення, де може бути багато нерелевантної інформації й немає чіткого розділення між класами в просторі ознак.[7]

Кластеризація k-середніх

Кластеризація K-середніх - це алгоритм машинного навчання без нагляду, який використовується для розбиття набору даних на заздалегідь визначену кількість кластерів. Даний підхід спрямований на пошук кластерів, в яких точки даних всередині кожного кластера схожі між собою і відрізняються від точок в інших кластерах.[3] Такий метод не є ефективним для класифікації зображень ураженої шкіри з декількох причин. По перше, кластеризація к-середніх працює шляхом поділу даних на кластери на основі схожості. У випадку зі зображеннями акне зважаючи на те, що існує багато типів хвороби й різні ступені ураження, складно порівняти два фото, які зовні дуже відрізняються, проте на обидвох зображено акне. По друге, кластеризація к-середніх є методом без наглядача й не використовує марковані дані для керування процесом кластеризації. У випадку зі зображеннями хвороб шкіри, де лікарі визначають наявність проблеми, не є раціональним нехтувати такими даними, бо результати кластеризації будуть найбільш ймовірно неточними. По третє, у контексті виявлення акне на зображеннях є проблема у визначенні відповідних початкових центроїдів, які ефективно фіксують варіабельність зовнішнього вигляду та розподілу акне на зображеннях. Висипання можуть відрізнятися за розміром, кольором і текстурою, що ускладнює вибір центроїдів, які будуть представляти основні кластери.

CNN - Convolutional Neural Networks

Твердження, що використання CNN є хорошим підходом для вирішення задачі класифікації зображень, є загально-відомим фактом. На це є ряд причин. Одним з головних недоліків підходів, описаних вище була нездатність

розрізняти, що різні варіації акне – це одна й та ж хвороба, адже висипання можуть проявляти різноманітні характеристики, такі як варіації кольору, нерівності форми та відмінності текстури. Також, кількість даних на зображенні неймовірно велика й тренування моделей було б дуже довгим. Всі ці проблеми можуть вирішити згорткові нейронні мережі.[7]

2.2 Вирішення завдань комп'ютерного зору за допомогою згорткових нейронних мереж (CNN)

Для розуміння, яким чином працюють згорткові нейронні мережі, потрібно мати уявлення про архітектуру глибоких нейронних мереж загалом. Загальна архітектура мережі – нейрони, згруповані у шари. Нейроном можна вважати функцію, яка бере значення з нейронів попереднього шару й обчислює нове значення, яке буде в діапазоні 0..1. Першим шаром будуть сирі дані – пікселі зображення. Вони будуть пропускатися прямим поширенням крізь приховані шари. Останній шар буде містити ту кількість нейронів, скільки класів об’єктів мережа розпізнає. Кожному зв’язку між нейронами буде надаватися вага (чим більша вага – тим більш яскравий піксель). Функція, яка буде обчислювати наступне значення нейрону, буде перемножувати матрицю пікселів на вектор ваг. Щоб нормалізувати, буде також застосовуватись функція зведення значення до діапазону 0..1. В розрахунок також буде внесено значення “bias”, яке буде позначати, наскільки великою має бути сума ваг, щоб нейрон ставав активним. Суть навчання буде полягати в тому, щоб комп’ютер знайшов правильні ваги та biases.[3]

На противагу стандартним моделям глибоких нейронних мереж, згорткові були спеціально розроблені для обробки структурованих сіткоподібних даних, таких як зображення. Є декілька головних відмінностей, якими вони різняться. Перш за все, як зазначено вище, у звичайних моделях

нейронних мереж зазвичай кожен нейрон з’єднаний з кожним нейроном наступного шару. У згорткових же мережах існують згорткові шари та агрегувальні шари, а далі буде повноз'єднаний шар , який буде застосовуватись для класифікації.

Згортковий шар складається з набору фільтрів, які будуть застосовуватись до вхідного зображення, представленого як N-вимірна матриця, для отримання вихідної карти ознак. Фільтр, або ж ядро, представляє собою матрицю значень, кожне з яких є вагою ядра. Операція згортки буде відбуватись таким чином, що береться початкова матриця ознак й менша матриця - рандомно заповнений фільтр. Ядро буде перетинати матрицю зображення, обчислюючи скалярний добуток між собою та поділами зображення тої самої розмірності. Далі добутки підсумовуються й утворюють скалярні значення, що представляють вихідну карту ознак. Таким чином формуються фільтри на всі ознаки зображення (наприклад очі, ніс, лапи, тулуб, голова тварини). Ця логіка базується на природі людського мозку – людина ідентифікує, що зображено, розбиваючи картинку на окремі компоненти.[8]

Головне завдання агрегувального шару (шару об’єднання) - зменшення карт ознак об’єктів великого розміру, щоб відповідно створити менші карти об’єктів, проте зберегти всю важливу інформацію. Береться мапа ознак й застосовується фільтр певної розмірності й кроку. Є різні методи, за допомогою яких можна фільтрувати. Найпоширеніші - мінімальне об’єднання, максимальне об’єднання та глобальне середнє об’єднання (GAP). Агрегувальний шар зменшує перенавчання (overfitting), оскільки зменшує кількість параметрів та робить модель толерантною до варіацій.[8]

Важливим аспектом є застосування функції активації до вихідної карти ознак. Вона допомагає зробити модель нелінійною. Функція активації

визначає, чи слід активувати нейрон, шляхом обчислення зваженої суми та додавання до неї зміщення. У CNN найчастіше використовується ReLu функція. Суть полягає у заміні всіх від’ємних значень у матриці на нулі.[8]

Повноз'єднаний шар використовується для класифікації вхідного зображення на класи (лейби).

Також особливість згорткових мереж, про яку не можна не згадати – це наскрізне навчання. Раніше класифікатори зображення працювали наступним чином: спочатку визначаються конкретні частини фото з метою сконцентруватися на ділянках зображення, які найбільше відповідають меті класифікації. Після цього вручну виділяються ознаки для кожної ідентифікованої точки інтересу, використовуючи предметну область. Потім використовується алгоритм відбору ознак, щоб залишити лише найвпливовіші ознаки.[10] Проте з появою CNN з’явився новий шлях тренування моделі, а саме - наскрізне навчання. Основною метою даного навчання є автоматизувати процес вилучення ознак, дозволяючи нейронній мережі вивчати ієрархічні представлення ознак безпосередньо з необроблених вхідних даних. Отже, CNN приймає необроблені зображення як вхідні дані і вчиться виокремлювати ознаки без будь-якого ручного етапу виокремлення ознак. CNN навчається виявляти візерунки, форми, текстури та інші ознаки безпосередньо з пікселів вхідних зображень за допомогою серії шарів згортки та об'єднання.

Отже, переваги використання CNN для класифікації зображень:

1) Згорткові мережі вміють вивчати складні шаблони з даних, що дозволяє їм вловлювати тонкі варіації уражень від вугрів. Це вирішить проблему визначення різного вигляду акне й дозволить правильно виявляти акне при різному освітленні на фотографії.
2) Фільтри дозволяють використовуючи той самий набір вагових коефіцієнтів для різних позицій у вхідних даних, ефективно вивчати та розпізнавати

ознаки, незалежно від їх розташування у вхідних даних. Тож CNN можуть ефективно виявляти ураження вугрів незалежно від їх положення чи масштабу на зображенні.

3) Згорткові мережі застосовують розподіл ваг для зменшених параметрів, що реалізовано за допомогою згорткових шарів. Усі шари CNN мають кілька згорткових фільтрів, які працюють і сканують повну матрицю ознак і здійснюють зменшення розмірності. Це критично важливо в контексті роботи з фотографіями акне, адже фото – це дані зі складною структурою й великою кількістю інформації, тому змога зменшити кількість даних, не втрачаючи важливі аспекти є ключовим моментом, вартим уваги.
4) Наскрізне навчання дозволяє CNN автоматично виявляти конкретні характеристики уражень шкіри, присутніх у навчальних даних. У контексті вугрів, це означає, що CNN може навчитися визначати важливі ознаки, які вказують на наявність вугрів на зображеннях, і використовувати ці ознаки для точної класифікації зображень, навіть якщо вугри мають різні розміри, форми або текстури. Такий підхід є більш ефективним, аніж вилучення ознак вручну, тому що таким чином краще виявити складні патерни на зображеннях.

Виходячи з вище сказаного, згорткові мережі чудово підходять для створення великомасштабної мережі.

Тож, для вирішення завдання комп’ютерного зору по розпізнаванню акне різних рівнів складності варто використовувати згорткові нейронні мережі.

2.3 Особливості роботи у фоновому режимі

Для хорошого функціоналу додатку часто потрібно враховувати які завдання на якому етапі життєвого циклу найбільш оптимально виконувати.

Загалом, існує п’ять різних стадій циклу:

- Unattached
- Foreground Inactive
- Foreground Active
- Background
- Suspended

Unattached є станом програми, коли вона не запущена й ніякі події додаток не отримує. Foreground Inactive є етапом підготовки програми до роботи, відображається екран запуску й відбувається готування структур даних. Foreground Active є станом працюючого додатку, відбувається взаємодія з користувачем. Background є частиною життєвого циклу, коли взаємодія з користувачем не відбувається, проте якщо додаток підписаний на певні події у роботі у фоні, при отриманні відповідних команд, програма вийде з режиму сну, виконає подію й повернеться у сплячий режим. Додаток захищений від несанкціонованого доступу, бо існує механізм під назвою сторожовий пес (watch dog), який припинить роботу програми, якщо є загроза несанкціонованого доступу. Останнім етапом циклу є Suspended. Єдина різниця між ним та Unattached є те, що в Suspended програма не працює, проте код самої програми і ресурси завантажені в оперативну пам’ять і запуск додатку може бути швидшим.

Одна з ключових проблем реалізації додатку по розпізнаванню хвороби шкіри за зображенням, яка потребує вирішення – забезпечення приватності користувача. Фотографії є особистими та чутливими даними, й для користувача важливо, щоб матеріали були захищені й не було ніякого витоку

інформації. Одним зі способів реалізації додатку для розпізнавання акне з фотографії таким чином, щоб дані були захищені – робити перевірку зображення у реальному часі, тобто користувач заходить у додаток, робить фото й бачить результат. За таким принципом реалізовано програму Cureto – збереження даних немає, лише безпосередня перевірка. Інший приклад існуючого додатку по виявленню шкірних хвороб по зображенню є Mobile-Based Skin Disease Diagnosis System. З перспективи безпеки, він працює таким же чином, що й Cureto, проте дозволяє користувачу не лише робити фото в реальному часі, а й завантажити його з галереї. Очевидно, підходи до реалізації приватності у такого типу додатків однакові у двох наведених прикладах й вони, безперечно, працюють.

З іншого боку, дуже важливо мати змогу виявляти шкірну хворобу на ранніх її проявах й слідкувати чи буде вона прогресувати. На основі цього розуміння, звертатися або не звертатися до лікаря. Даний шлях реалізації не дозволить користувачу відслідковувати свій стан шкіри ефективно й вимагатиме витрати часу на щоденні дії у додатку, що не завжди влаштовує людину.

Таким чином, логічною є ідея відслідковувати стан шкіри користувача й при виявленні покращень/погіршень ситуації – надсилати сповіщення про зміну кондиції. Було би добре не вимагати від людини ніяких особливих дій, лише повідомлення, аби користувач звернув увагу на стан своєї шкіри. У такому випадку, яким чином бути з приватністю й безпекою даних, адже для виявлення появи й тенденції розвитку хвороби потрібно якимось чином зберігати фото користувача. Тут дійсно у нагоді може стати фоновий режим, у якому може перебувати додаток. Програма зможе виконувати такі завдання, як обробка зображень, аналіз даних та аналіз періодичних оновлень. Додаток може час від часу зчитувати всі фото обличчя користувача або лише певну

частину й при виявленні певних змін – надсилати сповіщення власнику додатка. Таким чином практично ніяка взаємодія з людиною відбуватися не буде. Єдине, що потрібно – скачавши додаток, зробити своє селфі у програмі для того, щоб система мала вигляд власника й за даним фото здійснювала пошук його обличчя в галереї. Після цього, вийти з додатку. Таким чином, він перейде у стан фонового режиму й зможе виконувати певні події, а саме регулярну перевірку тенденції розвитку хвороби.

Є декілька шляхів реалізації даного механізму для ОС IOS.

Реалізація фонового режиму в додатках для iOS передбачає дотримання рекомендацій Apple та використання спеціальних API, створених для виконання подій у фоновому режимі. Залежно від характеру завдань і особливостей бажаної поведінки, можна використовувати різні фонові режими, що надаються iOS, такі як:

Фонова вибірка (Background Fetch): Цей режим дозволяє програмі періодично отримувати невеликі обсяги даних у фоновому режимі, забезпечуючи оновлення інформації без взаємодії з користувачем. Система буде планувати ці операції вибірки ситуативно, беручи до уваги такі фактори, як стан пристрою, щоб оптимізувати використання акумулятора. Щоб використовувати фонову вибірку, програма повинна спочатку зареєструватися для цієї можливості у класі делегата додатку. Під час першого запуску система може спитати дозвіл у користувача на обробку даних у фоні. Коли програма визначає, що доцільно виконати фонову вибірку, вона ненадовго пробуджує систему фоновому режимі. Потім програмі надається обмежений проміжок часу (близько 30 секунд), щоб виконати операцію вибірки та отримати будь-які відповідні дані або оновлення.

Фонове завдання (Background Task): iOS дозволяє програмам виконувати певні завдання у фоновому режимі, такі як,

завантаження/вивантаження файлів, оброблення даних із зовнішніх джерел або виконання довготривалих обчислень. Існує два типи фонових задач: BGAppRefreshTask і BGProcessingTask. BGAppRefreshTask зазвичай використовується програмами для виконання періодичних операцій оновлення у фоновому режимі. Наприклад, програма новин може використовувати BGAppRefreshTask для отримання останніх статей або заголовків через регулярні проміжки часу, гарантуючи, що користувачі завжди матимуть доступ до актуальної інформації, коли вони відкривають програму. Програми можуть планувати виконання BGAppRefreshTasks через певні проміжки часу за допомогою структури BackgroundTasks. Потім система періодично виводить програму з режиму сну, щоб вона могла виконати операцію оновлення, навіть якщо програма не запущена активно або знаходиться у стані foreground active. BGAppRefreshTasks розроблено для ефективного використання заряду батареї, що забезпечує мінімальне споживання ресурсів під час виконання. BGProcessingTask — це інший тип фонових завдань, що дозволяє програмам виконувати довгострокові або ресурсозатратні завдання у фоновому режимі. Як і BGAppRefreshTask, BGProcessingTask є частиною структури BackgroundTasks і пропонує аналогічні механізми планування та виконання. BGProcessingTask зазвичай використовується програмами для виконання завдань, які вимагають значної потужності процесора або тривалого часу виконання. Наприклад, програма може використовувати BGProcessingTask для виконання завдань обробки зображень у фоновому режимі.

Аналізуючи наявні механізми роботи у фоновому режимі, найбільш ефективним є використання BGProcessingTask для запланованого сканування й перевірки фотографій, щоб перевірка точно відбулася. Background Fetch можна використовувати для ситуативного сканування.

2.4 Прогнозування погіршення ситуації з акне

Загальною ідеєю додатку MiSkin є виявлення ранніх стадій хвороби акне й регулярний аналіз шкіри користувача й автоматичне сповіщення при виявленні змін стану. Користувач заходить в додаток й робить селфі, погоджується на доступ до фотографій й виходить з додатку. Програма одразу сканує фото у фоні вперше, при цьому відбуваються наступні процеси: визначення чи на фото зображено лице користувача і якщо так – фото відправляється на аналіз до згорткових мереж. Після аналізу зображень буде відбуватися аналіз результатів й визначення тенденції: чи з’явилося у користувача акне, чи змінилася інтенсивність висипань. Також якщо виявлені зміни, чи сповіщати вже людину, чи ще недостатньо підстав для хвилювання.

Аналіз відбуватиметься наступним чином: дані будуть ділитися на два масиви й порівнюватись кожні два фото послідовно. Для кожної пари фотографій:

Якщо на попередній фотографії не було акне, а на поточній є, вважаємо, що акне з'явилося і буде виводитись відповідне повідомлення.

Якщо на обох фотографіях є акне, порівнюється їх важкість. Якщо важкість акне зросла, виводиться повідомлення про збільшення важкості акне. Якщо важкість стабільна та немає змін, вважається, що стан акне став краще.

Для виявлення стабільності наявності акне протягом певного часу, буде вираховуватись, чи присутнє акне на мінімум десяти фотографіях. Якщо так, вважається, що акне є стабільним протягом цього періоду і буде створюватись відповідне сповіщення.

Якщо немає змін у тенденції акне, ніяких сповіщень користувач не отримуватиме.

РОЗДІЛ 3. Реалізація додатку для розпізнавання акне

3.1 Зібрання та підготовка датасету

Одним з найважливіших етапів розробки моделі нейронної мережі є зібрання валідного набору даних для тренування. Якщо підготувати датасет не вдаючись в деталі, то якість роботи натренованої моделі істотно зменшиться. У даному підрозділі представлено опис процесу збору та підготовки набору для тренування згорткової нейронної мережі (CNN). Головним аспектом є створення репрезентативного та різноманітного датасету, який дозволяє ефективно навчити та оцінити модель глибокого навчання.

Для створення тренувального набору даних для визначення ступені складності акне було використано датасети, взяті з ресурсу Kaggle. Датасет був взятий для визначення рівня складності акне. З причини недостатньої кількості зображень, було вручну додано фото з іншого набору даних. Для визначення стану шкіри ‘не акне’ використано датасет FaceData. Таким чином, створено навчальний набір (train dataset), набір підкріплення (validation dataset) та тестовий (test dataset). Дані були анотовані з метою позначення кожного елементу датасету згідно з його категорією: не акне, акне нульвої стадії, першої та другої (non_acne, level0, level1, level2). Навчальний набір є основним джерелом даних, який використовується для оновлення вагових коефіцієнтів та зсувів у всіх шарах CNN моделі під час процесу навчання. Модель намагається зробити прогнози для кожного прикладу в тренувальному датасеті, порівнюючи їх із відомими анотованими відповідями, і коригуючи параметри таким чином, щоб мінімізувати помилку прогнозування. Набір підкріплення використовується для виявлення перенавчання моделі (overfitting). Якщо модель показує високі показники точності на навчальному датасеті, але низькі на підкріпленні, це може свідчити про перенавчання.

Тестовий набір використовується для того, щоб додатково перевірити правильність роботи кінцевої натренованої моделі. Він дозволяє отримати об'єктивні оцінки продуктивності моделі, оскільки вона ще не бачила ці дані під час тренування або валідації. Це допомагає визначити, наскільки добре модель узагальнює свої знання на нові зображення.

3.2 Розроблення, навчання та інтегрування моделі CNN

Фокальною точкою додатку MiSkin, на якому і будується основна й більша частина функціоналу є натренована згорткова нейронна мережа. Одним зі шляхів її розробки є використання фреймворку TensorFlow - відкритої платформи глибокого навчання розробленою компанією Google.

TensorFlow забезпечує базовий рівень функцій та низькорівневий доступ до операцій глибокого навчання, тоді як Keras надає зручний інтерфейс для швидкої розробки та навчання моделей. Дана комбінація робить їх ефективними інструментами у галузі глибокого навчання.

Створення моделі нейронної мережі відображено у додатку 3.2.3.

Створюється модель Sequential, яка дозволить нам створювати нейронні мережі шляхом послідовного додавання шарів. Далі буде створений перший згортковий шар із 32 фільтрами розміром 3x3 й функцією активації ReLU та з вхідною формою зображень розміром 224x224x3 (ширина x висота x канали). Далі створено шар максимального зведення з фільтрами розміром 2x2, який зменшує розмірність зображення у 2 рази. Також буде додано ще три пари згорткових і максимального зведення шарів з різними кількостями фільтрів (64, 128, 128) та активацією ReLU.

Важливим є шар розгладжування, який перетворює вихід з попередніх шарів у вектор перед подачею у повнозв'язні шари. Далі - повнозв'язний шар з 512 нейронами та активацією ReLU.

Кінцевим шаром буде повнозв'язний шар з 4 виходами (для класифікації на 4 рівні акне) та активацією softmax, яка повертає ймовірності належності кожного зображення до кожного класу.

Підготовка даних для тренування та валідації за допомогою аугментації та генераторів даних відображено на лістингу 3.2.4. Анотуємо дані чотирма класами.

Після проведення тренування моделі, максимальна точність, якої вдалося досягнути – на 23 епосі. Результати відображені на додатку 3.2.1. Не найвища можлива точність пояснюється недостатньою кількість даних у датасеті.

Для інтеграції даної моделі у IOS проєкт потрібно декілька кроків. Firebase є платформою, яка використовується для розробки мобільних та веб додатків. За допомогою фреймворку ML Kit, який міститься у даній платформі, можна інтегрувати власну TensorflowLite модель у IOS додаток. Для цього потрібно додати бібліотеку ML Kit – Firebase/MLModelInterpreter у pod файл проєкту.

TensorFlowLite - це частина платформи TensorFlow, оптимізована для роботи на мобільних пристроях з обмеженими ресурсами. Основна мета TensorFlowLite - це забезпечити високу ефективність та швидкість виконання моделей машинного навчання на таких пристроях. Для того, щоб сконвертувати модель TensorFlow у TensorFlowLite, потрібно виконати наступні дії: [2]

```
keras_model = tf.keras.models.load_model(tflite_model_path)
converter = tf.lite.TFLiteConverter.from_keras_model(keras_model)
```

tflite_model = converter.convert()

Проте є недолік даного підходу – частково втрачається точність моделі. З метою виправлення проблеми було розроблено допоміжну бінарну модель нейронної мережі, яка визначатиме чи є у людини, зображеної на фото, акне. Перший згортковий шар буде мати 32 фільтри, розміром ядра (3, 3) та функцією активації ReLU. Після кожного згорткового шару наявний шар пулінгу з розміром пулінгу (2, 2). Другий згортковий шар з 64 фільтрами, розміром ядра (3, 3) та так само функцією активації ReLU.

Реалізація даної нейронної мережі й її тренування відображена на лістингу 3.2.5

При тренуванні бінарної моделі вдалося досягнути практично максимальної точності – 0.99 на 25 епосі. Результати відображені на додатку 3.2.2. За рахунок цього, при переведенні її у формат моделі TensorFlowLite точність практично не порушується.

3.3 Інтегрування фонового процесу для регулярного аналізу фотографій шкіри

Для досягнення якісної безпеки користувача одним з ключових аспектів, які потрібно реалізувати є інтегрування фонового процесу. Доцільним є створення двох варіантів роботи: перевірка, чи наявні нові фото кожні конкретні відведені періоди часу, для того, щоб бути певним, що аналіз фото відбудеться. Також створити перевірку, яка спрацьовуватиме тоді, коли система це вирішує.

Перший варіант реалізується за допомогою Background Task (фонового завдання). Буде використовуватися BGProcessingTask, бо процес аналізу фотографій не є швидким процесом. Дане завдання буде зареєстроване за допомогою BGTaskScheduler. У handleAppRefresh() визначена логіка

виконання завдання, яке включає перевірку наявності нових фотографій. Перевірка буде робитися кожні 20 днів, як зображено на рисунку 3.3.1. Для кожного з можливих результатів (.newPhotos, .noNewPhotos, .failed) ми будемо вказувати, чи успішно спрацював task, тобто в перших двох випадках - task.setTaskCompleted(success: true). Відповідно, якщо нові фото знайдено – буде викликатися метод, який починатиме сканування - scheduleAppRefresh().

Другий варіант реалізується за допомогою Background Fetch (фонового оновлення). Логіка обробки фонового оновлення визначена у відповідному методі application(_:performFetchWithCompletionHandler:). В даному методі виконується перевірка наявності нових фотографій через об'єкт observer, як зображено на рисунку 3.3.2. Якщо нові фото знайдені – викликаємо початок сканування.

Також, є ще один спосіб задання процесу, який відбуватиметься у фоні. У точці входу програми потрібно визначити реагування на зміни фази життєвого циклу додатка, тобто зміну на background. Це робиться за допомогою модифікатора .onChange, де для кейсу .background буде вказано виклик scheduleAppRefresh(). Даний підхід не є важливим для закладеної ідеї роботи MiSkin, проте він забезпечить миттєву перевірку фото користувача, коли він вийде з додатку, що є особливо корисним при першому виході користувача з додатку, адже користувач одразу зафіксує стан своєї шкіри. Даний підхід зображено на рисунку 3.3.3.

3.4 Опис розробки мобільного застосунку та принципи роботи

Мобільний застосунок MiSkin призначений для аналізу шкіри обличчя користувача з метою раннього виявлення шкірної хвороби Акне та вчасного сповіщення власника додатку про помічені зміни. При першому відкритті

додатку користувачу пропонується зробити селфі. Для роботи додатку користувачу варто погодитися на доступ до фотографій. MiSkin працює в фоновому режимі, скануючи фотографії користувача, щоб визначити наявність Акне й ступінь складності, а також виявляти тенденції у змінах стану хвороби.

MiSkin працює за наступними принципами:

1) користувач відкриває додаток та дозволяє доступ до камери та фотографій
2) користувач робить селфі за допомогою камери телефону
3) після погодження на доступ до фотографій, додаток починає працювати в фоновому режимі. Він періодично сканує нові фотографії, які додані у галерею, визначаючи, на яких фото присутнє обличчя користувача
4) якщо на знайдених фотографіях є обличчя користувача, додаток використовує згорткову нейронну мережу для визначення наявності акне. Він аналізує кожне фото, визначаючи, чи є на ньому ознаки акне й якщо так, якої ступені складності
5) якщо додаток виявляє будь-які зміни в стані шкіри користувача (наприклад, поява нових висипань або зміна ступеня акне), він надсилає сповіщення користувачеві.

Як зазначено вище, на кожному фото шукається обличчя користувача. Проте, важливим аспектом є процес препроцесингу – виявлення скільки облич й де саме вони розташовані на фото й обрізання – щоб надати моделі чітко фото обличь. Для цього створено метод detectFaces(photo:in:with:withBinar:), який буде виконувати обробку фотографій з метою виявлення обличчя на них для подальшого аналізу. За допомогою Vision Framework здійснюється

виявлення кожного обличчя, наявного на фото. Після цього знову таки, використовуючи Vision Framework, відбувається обрізання фотографії так, щоб залишити лише обличчя. Це дозволить зменшити розмір фото та зосередитися на області, яка містить інформацію про обличчя.

Наступним кроком буде перевірка схожості обличь за допомогою моделі Faces, яка є у відкритому доступі. Якщо обличчя користувача співпадає з обличчям на фото, відбувається аналіз за допомогою згорткових мереж. Дана реалізація відображена на рис. 3.4.1.

ВИСНОВКИ

Результатом виконання курсової роботи є створений додаток “MiSkin” для раннього виявлення акне, який розпізнає дану хворобу по зображенню користувача за допомогою згорткової нейронної мережі. Спочатку відбувається визначення чи наявна хвороба на обличчі користувача за допомогою бінарної моделі, й якщо акне виявлено, то відбувається визначення рівня складності акне. Також, MiSkin допомагає користувачу розуміти його ситуацію зі шкірою обличчя, адже додаток періодично відслідковує й аналізує стан шкіри й сповіщає користувача про зміни.

З дослідницької точки зору було проаналізовано декілька додатків по розпізнаванню вад шкіри по зображенню й виявлено, що згорткові нейронні мережі є найефективнішим методом для такого роду завдань. З основних причин, які надали CNN перевагу були наступні: дані мережі можуть розпізнавати патерни незалежно від їхнього положення на зображенні, об'єднання шарів допомагає зменшити просторові розміри карт об'єктів, зберігаючи при цьому найважливішу інформацію, а також CNN не чутливі до різних країв, текстур та форм на зображеннях.

В результаті виконання завдання курсової роботи було вирішено наступні проблеми:

- створення базової згорткової нейронної мережі, яка буде визначати складність акне
- питання конфіденційності користувача й безпеки його чутливих даних
- впровадження фонового процесу для періодичного аналізу фотографій користувача
- визначення способу впровадження створення тенденції розвитку шкірної хвороби користувача

У майбутньому додаток має багато шляхів для розвитку, аби надати користувачам більший спектр інформації, яку можна отримати. Наприклад, можна додати рекомендації по догляду та лікуванню шкіри у сповіщеннях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Marr D. VISION / David Marr. – San Francisco: W. H. Freeman and Company, 1982. – 393 с.

2 Bhalley R. Deep Learning with Swift for TensorFlow / Rahul Bhalley. – Ludhiana: Apress Media LLC, 2021. – 283 с.

3 Szeliski R. Computer Vision: Algorithms and Applications 2nd Edition / Richard Szeliski., 2021. – 861 с.

4 CURETO: Skin Diseases Detection Using Image Processing And CNN [Електронний ресурс] / [R. Karunanayake, M. Dananjaya, M. Peiris та ін.] // Sri Lanka Institute of Information Technology Press. – 2020. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/9299041.

5 Alamdari N. Detection and Classification of Acne Lesions in Acne Patients: A Mobile Application [Електронний ресурс] / N. Alamdari, K. Tavakolian, M. Alhashim // IEEE. – 2016. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/7535331.

6 Ahammed M. A machine learning approach for skin disease detection and classification using image segmentation [Електронний ресурс] / M. Ahammed, M. Al Mamun, M. Shorif Uddin // Elsevier Inc. – 2022. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S2772442522000624.

7 Maduranga P. Mobile-Based Skin Disease Diagnosis System Using Convolutional Neural Networks (CNN) [Електронний ресурс] / P. Maduranga, D. Nandasena // MECS. – 2022. – Режим доступу до ресурсу: https://www.mecs-press.org/ijigsp/ijigsp-v14-n3/IJIGSP-V14-N3-5.pdf.

8 Alzubaidi L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions [Електронний ресурс] / L. Alzubaidi, J.

Zhang, A. Humaidi // J Big Data. – 2021. – Режим доступу до ресурсу: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8.

9 Tan J. A global perspective on the epidemiology of acne [Електронний ресурс] / J. Tan, K. Bhate // British Journal of Dermatology. – 2015. – Режим доступу до ресурсу: https://academic.oup.com/bjd/article/172/S1/3/6615714?login=true.

10 Panagiotis A. What Is End-to-End Deep Learning? [Електронний ресурс] / Antoniadis Panagiotis // Baeldung. – 2023. – Режим доступу до ресурсу: https://www.baeldung.com/cs/end-to-end-deep-learning.

ДОДАТКИ

```
Found 2466 images belonging to 4 classes.
Found 2208 images belonging to 4 classes.
Epoch 1/25
124/124 [==============================] - 179s 1s/step - loss: 0.9825 -
accuracy: 0.5600 - val_loss: 0.7184 - val_accuracy: 0.6467
Epoch 2/25
124/124 [==============================] - 162s 1s/step - loss: 0.7389 -
accuracy: 0.6500 - val_loss: 0.7004 - val_accuracy: 0.6635
Epoch 3/25
124/124 [==============================] - 166s 1s/step - loss: 0.6862 -
accuracy: 0.6732 - val_loss: 0.6890 - val_accuracy: 0.6368
Epoch 4/25
124/124 [==============================] - 176s 1s/step - loss: 0.6646 -
accuracy: 0.6886 - val_loss: 0.6673 - val_accuracy: 0.6753
Epoch 5/25
124/124 [==============================] - 188s 2s/step - loss: 0.6517 -
accuracy: 0.6902 - val_loss: 0.5962 - val_accuracy: 0.6952
Epoch 6/25
124/124 [==============================] - 181s 1s/step - loss: 0.6377 -
accuracy: 0.6886 - val_loss: 0.5952 - val_accuracy: 0.6988
Epoch 7/25
124/124 [==============================] - 194s 2s/step - loss: 0.6169 -
accuracy: 0.7048 - val_loss: 0.5921 - val_accuracy: 0.7124
Epoch 8/25
124/124 [==============================] - 180s 1s/step - loss: 0.6119 -
accuracy: 0.7003 - val_loss: 0.5572 - val_accuracy: 0.7455
Epoch 9/25
124/124 [==============================] - 198s 2s/step - loss: 0.6160 -
accuracy: 0.7068 - val_loss: 0.5668 - val_accuracy: 0.7423
Epoch 10/25
124/124 [==============================] - 201s 2s/step - loss: 0.5948 -
accuracy: 0.7129 - val_loss: 0.5421 - val_accuracy: 0.7559
Epoch 11/25
124/124 [==============================] - 179s 1s/step - loss: 0.5827 -
accuracy: 0.7210 - val_loss: 0.5236 - val_accuracy: 0.7405
Epoch 12/25
124/124 [==============================] - 151s 1s/step - loss: 0.5671 -
accuracy: 0.7186 - val_loss: 0.5261 - val_accuracy: 0.7219
```

```
Epoch 13/25
124/124 [==============================] - 137s 1s/step - loss: 0.5611 -
accuracy: 0.7344 - val_loss: 0.5560 - val_accuracy: 0.7477
Epoch 14/25
124/124 [==============================] - 137s 1s/step - loss: 0.5629 -
accuracy: 0.7307 - val_loss: 0.5164 - val_accuracy: 0.7631
Epoch 15/25
124/124 [==============================] - 157s 1s/step - loss: 0.5622 -
accuracy: 0.7332 - val_loss: 0.5029 - val_accuracy: 0.7582
Epoch 16/25
124/124 [==============================] - 148s 1s/step - loss: 0.5507 -
accuracy: 0.7332 - val_loss: 0.5042 - val_accuracy: 0.7749
Epoch 17/25
124/124 [==============================] - 155s 1s/step - loss: 0.5556 -
accuracy: 0.7356 - val_loss: 0.5442 - val_accuracy: 0.7631
Epoch 18/25
124/124 [==============================] - 153s 1s/step - loss: 0.5462 -
accuracy: 0.7360 - val_loss: 0.4916 - val_accuracy: 0.7582
Epoch 19/25
124/124 [==============================] - 151s 1s/step - loss: 0.5427 -
accuracy: 0.7360 - val_loss: 0.5130 - val_accuracy: 0.7695
Epoch 20/25
124/124 [==============================] - 146s 1s/step - loss: 0.5359 -
accuracy: 0.7490 - val_loss: 0.4904 - val_accuracy: 0.7659
Epoch 21/25
124/124 [==============================] - 157s 1s/step - loss: 0.5401 -
accuracy: 0.7364 - val_loss: 0.4997 - val_accuracy: 0.7387
Epoch 22/25
124/124 [==============================] - 162s 1s/step - loss: 0.5232 -
accuracy: 0.7474 - val_loss: 0.4846 - val_accuracy: 0.7899
Epoch 23/25
124/124 [==============================] - 160s 1s/step - loss: 0.5444 -
accuracy: 0.7409 - val_loss: 0.4745 - val_accuracy: 0.7803
Epoch 24/25
124/124 [==============================] - 159s 1s/step - loss: 0.5194 -
accuracy: 0.7498 - val_loss: 0.4507 - val_accuracy: 0.7889
Epoch 25/25
124/124 [==============================] - 145s 1s/step - loss: 0.5135 -
accuracy: 0.7559 - val_loss: 0.5071 - val_accuracy: 0.7613
```

Лістинг 3.2.1

```
Epoch 1/25
120/120 [==============================] - 86s 706ms/step - loss:
0.2180 - accuracy: 0.9121 - val_loss: 0.1609 - val_accuracy: 0.9494

Epoch 2/25
120/120 [==============================] - 86s 721ms/step - loss:
0.0773 - accuracy: 0.9728 - val_loss: 0.0658 - val_accuracy: 0.9781

Epoch 3/25
120/120 [==============================] - 95s 792ms/step - loss:
0.0503 - accuracy: 0.9849 - val_loss: 0.0493 - val_accuracy: 0.9825

Epoch 4/25
120/120 [==============================] - 93s 779ms/step - loss:
0.0512 - accuracy: 0.9858 - val_loss: 0.0705 - val_accuracy: 0.9766

Epoch 5/25
120/120 [==============================] - 95s 794ms/step - loss:
0.0498 - accuracy: 0.9833 - val_loss: 0.0962 - val_accuracy: 0.9701

Epoch 6/25
120/120 [==============================] - 91s 762ms/step - loss:
0.0398 - accuracy: 0.9874 - val_loss: 0.0503 - val_accuracy: 0.9814

Epoch 7/25
120/120 [==============================] - 108s 903ms/step - loss:
0.0371 - accuracy: 0.9866 - val_loss: 0.0656 - val_accuracy: 0.9760

Epoch 8/25
120/120 [==============================] - ETA: 0s - loss: 0.0289 -
accuracy: 0.9908
        Cancelling training
120/120 [==============================] - 99s 825ms/step - loss:
0.0289 - accuracy: 0.9908 - val_loss: 0.0517 - val_accuracy: 0.9805

Epoch 9/25
120/120 [==============================] - 94s 784ms/step - loss:
0.0352 - accuracy: 0.9891 - val_loss: 0.0997 - val_accuracy: 0.9698

Epoch 10/25
120/120 [==============================] - 101s 845ms/step - loss:
0.0276 - accuracy: 0.9900 - val_loss: 0.0455 - val_accuracy: 0.9837
```

Epoch 11/25
120/120 [==============================] - ETA: 0s - loss: 0.0266 - accuracy: 0.9929
Cancelling training
120/120 [==============================] - 114s 955ms/step - loss: 0.0266 - accuracy: 0.9929 - val_loss: 0.1055 - val_accuracy: 0.9678

Epoch 12/25
120/120 [==============================] - ETA: 0s - loss: 0.0240 - accuracy: 0.9921

Cancelling training
120/120 [==============================] - 112s 933ms/step - loss: 0.0240 - accuracy: 0.9921 - val_loss: 0.0569 - val_accuracy: 0.9811

Epoch 13/25
120/120 [==============================] - ETA: 0s - loss: 0.0311 - accuracy: 0.9912
Cancelling training
120/120 [==============================] - 98s 820ms/step - loss: 0.0311 - accuracy: 0.9912 - val_loss: 0.2994 - val_accuracy: 0.9006

Epoch 14/25
120/120 [==============================] - 96s 803ms/step - loss: 0.0300 - accuracy: 0.9887 - val_loss: 0.0662 - val_accuracy: 0.9757

Epoch 15/25
120/120 [==============================] - ETA: 0s - loss: 0.0249 - accuracy: 0.9904
Cancelling training
120/120 [==============================] - 91s 759ms/step - loss: 0.0249 - accuracy: 0.9904 - val_loss: 0.0470 - val_accuracy: 0.9837

Epoch 16/25
120/120 [==============================] - ETA: 0s - loss: 0.0193 - accuracy: 0.9929
Cancelling training
120/120 [==============================] - 89s 748ms/step - loss: 0.0193 - accuracy: 0.9929 - val_loss: 0.0659 - val_accuracy: 0.9802Epoch 17/25
120/120 [==============================] - ETA: 0s - loss: 0.017 - accuracy: 0.9937
Cancelling training

120/120 [==============================] - 108s 905ms/step - loss: 0.0172 - accuracy: 0.9937 - val_loss: 0.0387 - val_accuracy: 0.9888

Epoch 18/25
120/120 [==============================] - 107s 895ms/step - loss: 0.0235 - accuracy: 0.9895 - val_loss: 0.1356 - val_accuracy: 0.9654

Epoch 19/25
120/120 [==============================] - 107s 894ms/step - loss: 0.0244 - accuracy: 0.9895 - val_loss: 0.0963 - val_accuracy: 0.9734

Epoch 20/25
120/120 [==============================] - ETA: 0s - loss: 0.0186 - accuracy: 0.9941
Cancelling training
120/120 [==============================] - 114s 948ms/step - loss: 0.0186 - accuracy: 0.9941 - val_loss: 0.0748 - val_accuracy: 0.9799

Epoch 21/25
120/120 [==============================] - ETA: 0s - loss: 0.0231 - accuracy: 0.9904
Cancelling training
120/120 [==============================] - 103s 864ms/step - loss: 0.0231 - accuracy: 0.9904 - val_loss: 0.0610 - val_accuracy: 0.9817

Epoch 22/25
120/120 [==============================] - ETA: 0s - loss: 0.0190 - accuracy: 0.9941
Cancelling training
120/120 [==============================] - 104s 867ms/step - loss: 0.0190 - accuracy: 0.9941 - val_loss: 0.0893 - val_accuracy: 0.9757

Epoch 23/25
120/120 [==============================] - ETA: 0s - loss: 0.0107 - accuracy: 0.9962
Cancelling training
120/120 [==============================] - 120s 1s/step - loss: 0.0107 - accuracy: 0.9962 - val_loss: 0.1244 - val_accuracy: 0.9722

Epoch 24/25
120/120 [==============================] - ETA: 0s - loss: 0.0191 - accuracy: 0.9933
Cancelling training

```
        120/120 [==============================] - 90s 755ms/step - loss:
0.0191 - accuracy: 0.9933 - val_loss: 0.0790 - val_accuracy: 0.9805

Epoch 25/25
120/120 [==============================] - ETA: 0s - loss: 0.0180 -
accuracy: 0.9941
        Cancelling training
        120/120 [==============================] - 89s 744ms/step - loss:
0.0180 - accuracy: 0.9941 - val_loss: 0.0777 - val_accuracy: 0.9790
```

Лістинг 3.2.2

```
model = models.Sequential()

# Додаємо перший згортковий шар
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# Додаємо другий згортковий шар
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Додаємо третій згортковий шар
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Додаємо четвертий згортковий шар
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Згладжуємо вхід перед додаванням повнозв'язних шарів
model.add(layers.Flatten())

# Додаємо повнозв'язний шар
model.add(layers.Dense(512, activation='relu'))

# Вихідний повнозв'язний шар з 4 виходами (для 4 рівнів складності акне)
model.add(layers.Dense(4, activation='softmax'))

# Компілюємо модель
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Лістинг 3.2.3

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    classes=['non_acne', 'level0', 'level1', 'level2']
)

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    classes=['non_acne', 'level0', 'level1', 'level2']
)
```

Запрограмовуємо тренування моделі:

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=25,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)
```

Лістинг 3.2.4

```
train_data_dir = '/Users/lizzikuchyna/Desktop/dataset/train'
model = Sequential()

# Додаємо згорткові шари
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # One unit for binary classification
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy',
metrics=['accuracy'])

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
)

model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=25
)
```

Лістинг 3.2.5

```swift
func handleAppRefresh(task: BGProcessingTask) {  // для регулярної перевірки
    // Schedule the next background task
    scheduleNextAppRefresh()

    // Perform task
    observer.checkForNewPhotos { result in
        switch result {
        case .newPhotos:
            task.setTaskCompleted(success: true) // якщо є нові фото -> відбувається перевірка
            MLWork().scheduleAppRefresh()
        case .noNewPhotos:
            task.setTaskCompleted(success: true)
        case .failed:
            task.setTaskCompleted(success: false)
        }
    }
}
```

```swift
func scheduleNextAppRefresh() {
    let request = BGAppRefreshTaskRequest(identifier: taskId)
    request.earliestBeginDate = Date(timeIntervalSinceNow: 20 * 24 * 60 * 60) // 20 days
    do {
        try BGTaskScheduler.shared.submit(request)
    } catch {
        print("Unable to submit task: \(error.localizedDescription)")
    }
}
```

Рис. 3.3.1

```swift
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler: @escaping
    (UIBackgroundFetchResult) -> Void) {

        observer.checkForNewPhotos { result in
            switch result {
            case .newPhotos:
                completionHandler(.newData)
                MLWork().scheduleAppRefresh()
            case .noNewPhotos:
                completionHandler(.noData)
            case .failed:
                completionHandler(.failed)
            }
        }
    }
```

Рис. 3.3.2

```
@main
struct detectAcneOneMoreTimeApp: App {
    @Environment(\.scenePhase) private var phase
    var ml = MLWork()

    @UIApplicationDelegateAdaptor(AppDelegate.self) var delegate
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
        .onChange(of: phase) { newPhase in
            switch newPhase {
            case .background: ml.scheduleAppRefresh()
            default: break
            }
        }

    }


}
```

Рис. 3.3.3

```
private func detectFaces(photo image: UIImage, in cgImage: CGImage, with interpreter: Interpreter, withBinar
    interpreterBinar: Interpreter) async {


    let requestHandler = VNImageRequestHandler(cgImage: cgImage)
    let faces = VNDetectFaceRectanglesRequest { [self] request, error in

        guard error == nil else {
            print("Failed to detect faces:", error!)
            return
        }

        if let results = request.results as? [VNFaceObservation], !results.isEmpty {

            results.forEach { faceObservation in

                let boundingBox = faceObservation.boundingBox
                let x = boundingBox.origin.x * image.size.width
                let y = (1 - boundingBox.origin.y - boundingBox.height) * image.size.height
                let width = boundingBox.width * image.size.width
                let height = boundingBox.height * image.size.height

                let expandedBoundingBox = CGRect(x: max(0, x - width * 0.1),
                                                 y: max(0, y - height * 0.2),
                                                 width: min(image.size.width - x, width * 1.2),
                                                 height: min(image.size.height - y, height * 1.5))

                let faceRect = expandedBoundingBox

                guard let croppedCGImage = image.cgImage?.cropping(to: faceRect) else { return }
                let resizedImage = self.resized(image: croppedCGImage, to: CGSize(width: 224, height: 224))

                let faceUIImage = UIImage(cgImage: croppedCGImage)
                guard let imageData = faceUIImage.jpegData(compressionQuality: 1.0) else { return }

                guard let documentsDirectory = FileManager.default.urls(for: .documentDirectory, in:
                    .userDomainMask).first else {
                    print("Unable to locate the documents directory.")
                    return
                }

                let fileName = "face.jpg"
                let fileURL = documentsDirectory.appendingPathComponent(fileName)
```

```
            do {
                try imageData.write(to: fileURL)
                print("Face image saved to documents directory: \(fileURL.path)")
            } catch {
                print("Error saving face image:", error)
            }

            // беремо фото з камери
            guard let imageFromCamera = capturedImageData else { return }

            if let userImage = UIImage(data: imageFromCamera) {


                let resizedImage1 = userImage.resized(to: CGSize(width: 96, height: 96))! // userImage
                let resizedImage2 = image.resized(to: CGSize(width: 96, height: 96))!


                let faceComparator = SFaceCompare(firstFace: resizedImage1, secondFace: resizedImage2)
                faceComparator.compareFaces { result in   // compare user face and detected
                    switch result {
                    case .success(let areSimilar):
                        if areSimilar {

                            self.analyzeImage(resizedImage!, with: interpreter, withBinar: interpreterBinar, amount:
                                results.count) // go to ml models
                            print("Faces are similar.")
                        } else {
                            print("Faces are not similar.")
                        }
                    case .failure(let error):
                        print("Error occurred: \(error)")
                    }
                }
            }

        }
    } else {
        print("No faces detected.")
    }
}
```

```
    #if targetEnvironment(simulator)

    faces.usesCPUOnly = true

    #endif


    do {

        try requestHandler.perform([faces])

    } catch {

        print("Error analyzing image: \(error)")

        return

    }

}
```

Рис. 3.4.1