КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

"До захисту допущено"
Завідувач кафедри
В. М. Терещенко _____
(підпис)
"__" _____20__р.

Дипломна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Виконала студентка 4 курсу
Северинеко Наталія Олександрівна _____
(підпис)

Науковий керівник:
доцент кафедри математичної інформатики,
кандидат фіз.-мат. наук
Деревянченко Олександр Валерійович _____
(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ - 2021

РЕФЕРАТ

Обсяг роботи 42 сторінки, 18 ілюстрацій, 2 графіки, 7 формул, 3 таблиці, 31 джерело посилань.

ANDROID, JAVA, PYTHON, RESNET, TENSORFLOW, U-NET, ГЕНЕРАТИВНІ ЗМАГАЛЬНІ МЕРЕЖІ, ДОПОВНЕННЯ ДАНИХ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, МАКІЯЖ, МАШИННЕ НАВЧАННЯ, МОБІЛЬНИЙ ДОДАТОК, НЕЙРОННА МЕРЕЖА, ОПТИМІЗАЦІЯ МОДЕЛІ, ПЕРЕНЕСЕННЯ СТИЛЮ.

Об’єктом дослідження цієї дипломної роботи є модель нейронної мережі, яка дозволяє реалістично зняти макіяж з зображення людини. Об’єктом розробки є мобільний додаток з використанням цієї мережі.

Метою дипломної роботи є побудова нейронної мережі, яка дозволить реалістично знімати макіяж з зображення з використанням непарних даних, та створення додатку на платформі Android з використанням цієї нейронної мережі.

В якості інструменту для написання нейронної мережі було використано бібліотеку TensorFlow. В якості інструменту для створення додатку було використано Android Studio.

Результатом виконання є мобільний додаток для платформи Android з використанням власної побудованої та натренованої нейронної мережі для зняття макіяжу. Новизна: додатків на Android, які знімають макіяж поки не існує, тому тема дипломної роботи є актуальною.

Взаємозв’язок з іншими роботами: модель, яка була побудована у цій дипломній роботі, базується на архітектурі CycleGAN; генератор G, який знімає макіяж, побудований за архітектурою U-net, як було використано в роботі Pix2Pix; генератор F наносить макіяж і має ResNet архітектуру, яку використовували в мережі BeautyGAN.

Розроблену модель нейронної мережі можна використати для подальших досліджень в сфері перенесення різних ознак на зображеннях не тільки для

зняття макіяжу. У свою чергу сам імплементований додаток на Android можна застосовувати як в комерційних, так і в індивідуальних цілях для розваги.

Об’єкт дослідження цієї дипломної роботи може бути розглянутий і проаналізований глибше, адже були виявлені області в яких можливе покращення.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта дослідження. Станом на сьогодні існує багато різних досліджень [7, 16, 17] пов’язаних з перенесенням ознак на зображеннях за допомогою нейронних мереж. Для генерації зображень схожих до реальних використовуються генеративні змагальні мережі GAN[8], які складаються з генератора і дискримінатора з використанням CNN[5]. На основі цих архітектур були розроблені нейронні мережі для перенесення стилю. Pix2Pix[16] на відміну від попередніх досліджень використали архітектуру U-net [9] для генератора і PatchGAN [18] для дискримінатора, проте цей метод потребував парних даних для тренування. CycleGAN [17] вирішили проблему відсутності парних вхідних даних за допомогою двох GAN, які дають змогу обчислити додаткову функцію втрат - циклічну. Ці висновки були використані у роботах, які безпосередньо переносили і знімали макіяж [19, 20, 21]. BeautyGAN [19] першими запропонували GAN фреймворк з двома входами і двома виходами. Вони також запропонували свою функцію втрат для макіяжу. BeautyGlow [21] запропонували виокремити компоненту макіяжу і не макіяжу. PairedCycleGAN [20] використав додатковий дискримінатор для того, щоб провести перенесення макіяжу за допомогою псевдо перенесених зображень, які були створені поєднанням вхідного зображення без макіяжу і зображення з макіяжем, на яке посилаються. LADN [22] використали велику кількість локальних дискримінаторів, щоб перенести і зняти більш яскравий макіяж.

Існує одна реалізація додатку для платформи IOS - MakeApp [25], серед основних функцій якого є зняття макіяжу.

Актуальність роботи та підстави для її виконання. Використання нейронних мереж у додатках для комерційних і розважальних цілей набуває все більшої популярності. З розвитком соціальних мереж контент пов’язаний з макіяжем став дуже розповсюдженим оскільки він має попит [24].

Зараз існують сервіси, як сайти так і додатки, які дозволяють класифікувати об’єкти, змінювати та стилізувати зображення, покращувати

фотографії та інше за допомогої нейронних мереж. Проте додатків на Android, які знімають макіяж поки не існує, тому тема дипломної роботи є актуальною.

Мета і завдання роботи. Метою дипломної роботи є побудова нейронної мережі, яка дозволить реалістично знімати макіяж з зображення з використанням непарних даних, та створення додатку на платформі Android з використанням цієї нейронної мережі. Для досягнення цієї мети поставлені наступні завдання:

- Дослідити існуючі наукові статті, які описують архітектури для зняття макіяжу.
- Дослідити підходи і програмні засоби, які використовуються для побудови, реалізації нейронних мереж та розгортання їх на Android.
- Знайти набір даних, який б підходив для тренування мережі для цього завдання.
- Побудувати власну модель нейронної мережі.
- Спроектувати на розробити мобільний додаток.
- Оптимізувати та розгорнути побудовану модель в додатку.
- Продемонструвати результати досліджень і розробки.

Об’єкт, методи й засоби розробки. Об’єктом дослідження цієї дипломної роботи є модель нейронної мережі, яка дозволяє реалістично зняти макіяж з зображення людини. Об’єктом розробки є мобільний додаток з використанням цієї мережі.

Методами дослідження є *теоретичні*: аналіз (для узагальнення наукових досліджень за цією темою), синтез (для формулювання висновків); *практичні*: для створення моделі нейронної мережі, для розробки мобільного додатку; *прикладні*: комп’ютерне програмування.

В якості інструменту для написання нейронної мережі було використано бібліотеку TensorFlow. В якості інструменту для створення додатку було використано Android Studio, яка є офіційним інтегрованим середовищем розробки для Android додатків.

Можливі сфери застосування. Розроблену модель нейронної мережі можна використати для подальших досліджень в сфері перенесення різних ознак на зображеннях не тільки для зняття макіяжу. Натреновану модель можна використовувати як на мобільних пристроях, так і у веб-додатках. У свою чергу сам імплементований додаток на Android можна застосовувати як в комерційних, так і в індивідуальних цілях для розваги.

Взаємозв'язок з іншими роботами. Модель, яка була побудована у цій дипломній роботі, базується на архітектурі CycleGAN [17]. Генератор G, який знімає макіяж, побудований за архітектурою U-net, як було використано в роботі Pix2Pix[16], генератор F наносить макіяж і має ResNet архітектуру, яку використовували в мережі BeautyGAN [19].

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Загальні поняття про мобільну розробку

Програми під Android можна писати на різних мовах, таких як Kotlin, Java та C++. Інструменти SDK для Android компілюють код разом з будь-якими файлами даних та ресурсів у файл APK (формат архівних файлів-додатків для Android з суфіксом .apk). Файл APK містить весь вміст програми і за допомогою нього пристрої на базі Android встановлюють програми. [1]

Для розробки додатків для Android найчастіше використовується Android Studio. Це офіційне інтегроване середовище розробки (IDE), засноване на IntelliJ IDEA. Додатково до потужного редактора коду та інструментів розробника IntelliJ, Android Studio пропонує ще більше функцій, які підвищують продуктивність при створенні програм. [2]

Система Android реалізує принцип найменших привілеїв. Тобто кожна програма за замовчуванням має доступ лише до тих компонентів, які потрібні для її роботи, і не більше. Це створює безпечне середовище, в якому програма не може отримати доступ до частин системи, на які їй не надано дозволу. [1]

1.2. Компоненти Android додатків

Програми для Android складаються з компонентів, кожен з яких є точкою входу, через яку система або користувач може увійти у додаток. Одні компоненти залежать від інших. Існує чотири різні типи компонентів програми: Діяльності (Activities), Сервіс (Services), трансляційні приймачі (Broadcast receivers), Провайдери контенту (Content providers). Кожен тип має певну мету і чіткий життєвий цикл, який визначає спосіб створення та знищення компонента. [1]

Найважливішим компонентом програми для Android є клас Activity. Спосіб у який Activity запускаються і складаються є фундаментальною

частиною моделі програми платформи. На відміну від парадигм програмування, в яких програми запускаються методом main(), система Android ініціює код в екземплярі Activity, викликаючи конкретні методи зворотного виклику, які відповідають певним етапам його життєвого циклу. Activity надає вікно для інтерфейсу. Як правило, одна з дій у програмі вказується як основна, тобто перший екран, що з’являється, коли користувач запускає програму [3].

Сервіс - це універсальна точка входу для роботи програми у фоновому режимі. Це компонент, який працює у фоновому режимі для виконання тривалих операцій або виконання роботи для віддалених процесів. Послуга не має інтерфейсу користувача. Наприклад, служба може відтворювати музику у фоновому режимі, поки користувач знаходиться в іншій програмі, або отримувати дані через мережу, не заважаючи користувачеві взаємодіяти з класом діяльністю. Інший компонент (наприклад, діяльність) може запустити сервіс та змусити його запуститися або з’єднатись з іншим сервісом для взаємодії.

Трансляційні приймачі - це компоненти, що дозволяють системі доставляти події до прикладної програми за межами звичайного потоку користувачів, а також вони дозволяють прикладній програмі реагувати на загальносистемні широкомовні повідомлення. Оскільки трансляційний приймач є ще одним чітко визначеним елементом програми, система може навіть надсилати трансляції програмам, які наразі не працюють.

Постачальник вмісту керує спільним набором даних програми, який можна зберігати у файловій системі, у базі даних SQLite, в Інтернеті або в будь-якому іншому постійному сховищі, до якого програма може отримати доступ. Інші програми можуть запитувати або змінювати дані, якщо постачальник вмісту дозволяє це.

Три з чотирьох типів компонентів активуються асинхронним повідомленням, яке називається наміром. Наміри пов'язують окремі

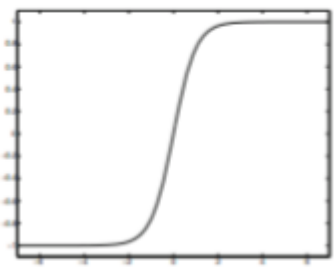
компоненти між собою під час виконання. Намір створюється за допомогою об'єкта Intent, який визначає повідомлення для активації або певного компонента (явний намір), або певного типу компонента (неявний намір)[1].

1.3. Загальні поняття про нейронні мережі

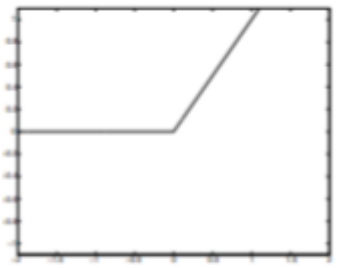
Нейронні мережі, також відомі як штучні нейронні мережі (ANN) або модельовані нейронні мережі (SNN), є підмножиною машинного навчання і лежать в основі алгоритмів глибокого навчання.

Штучні нейронні мережі складаються з шарів вузлів, що містять вхідний шар, один або кілька прихованих шарів та вихідний шар. Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну вагу та порогове значення. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі. Функція активації [12], або передавальна функція штучного нейрона — це залежність вихідного сигналу штучного нейрона від вхідного.

У цій дипломній роботі використовуються наступні функції активації: TanH (графік 1), ReLu (графік 2), LeakyReLu.



Графік 1. TanH



Графік 2. ReLu

Нейронні мережі покладаються на навчальні дані, щоб навчитися та підвищувати їх точність з часом. Вони є потужними інструментами в галузі інформатики та штучного інтелекту, що дозволяє класифікувати та кластеризувати дані з великою швидкістю. Завдання з розпізнавання мови

чи розпізнавання зображень можуть зайняти хвилини проти годин у порівнянні з ідентифікацією вручну, проведеною експертами-людьми.

Для більш практичних випадків використання нейронних мереж, таких як розпізнавання зображень або класифікація, використовується контрольоване навчання (supervised learning). Для задач, в яких відомий опис множини об'єктів (навчальна вибірка), і необхідно виявити внутрішні взаємозв'язки, залежності, закономірності, що існують між об'єктами використовується навчання без вчителя (unsupervised learning).

Під час навчання моделі оцінюється її точність за допомогою функції втрат. Функція втрат — це функція, яка відображує подію, або значення однієї чи декількох величин, на дійсне число, яке інтуїтивно представляє якісь «витрати», пов'язані з цією подією. Цільова функція є або функцією втрат, або протилежною їй. Зрештою, метою є мінімізація функції витрат, щоб забезпечити правильність визначення для будь-якого вибраного спостереження. Оскільки модель коригує свої ваги та упередження, вона використовує функцію витрат та навчання підкріплення, щоб досягти точки збіжності, або місцевого мінімуму. Алгоритм регулює свої ваги за допомогою градієнтного спуску, що дозволяє моделі визначати напрямок руху для мінімізувати функції витрат. З кожним навчальним екземпляром параметри моделі регулюються для того, щоб поступово збігтися до мінімуму. [4]

Нормалізація - метод, який дозволяє підвищити продуктивність та стабільність нейронних мереж. Існує декілька видів: пакетна нормалізація та нормалізація екземплярів.

Нормалізація пакету (batch) [14] - це метод, який нормалізує активацію в мережі через міні-пакет певного розміру. Для кожної функції пакетна нормалізація обчислює середнє значення та дисперсію цієї функції в міні-партії. Потім він віднімає середнє значення і ділить ознаку на його міні-пакетне стандартне відхилення.

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_\beta^{\,2} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_i - \mu_\beta\right)^2$$

Формула 1. Батч нормалізація

Нормалізація екземплярів [14] замість нормалізації вхідних параметрів нормалізує по кожному каналу у кожному навчальному прикладі. На відміну від пакетної нормалізації, нормалізація екземпляра застосовується і під час тестування.

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}},\ \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm},\ \sigma_{ti}^{\,2} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m-1}^{H}\left(x_{tilm} - mu_{ti}\right)^2$$

Формула 2. Нормалізація екземплярів

1.4. Нейронні мережі для обробки зображень

Для обробки зображень використовуються згорткові нейронні мережі (CNN). Архітектури ConvNet висловлюють припущення, що вхідні дані - це зображення, що дозволяє закодовувати певні властивості в архітектурі. Потім вони роблять функцію переадресації більш зручною для реалізації та значно зменшують кількість параметрів у мережі. Зокрема, на відміну від звичайної нейронної мережі, шари ConvNet мають нейрони, розташовані в трьох вимірах: ширина, висота, глибина.

Параметри в згортковому шарі складаються з набору фільтрів, які можна навчити. Кожен фільтр невеликий по ширині та висоті, але простягається на всю глибину вхідного об'єму. Наприклад, типовий фільтр на першому шарі може мати розмір 5x5x3 (тобто 5 пікселів у ширину та висоту та 3 кольорові канали). Під час прямого проходу ми згортаємо кожен фільтр по ширині та висоті вхідного об'єму та обчислюємо скалярні добутки між вхідними даними фільтра та входом у будь-якій позиції. Коли ми просуваємо фільтр по ширині та висоті вхідного об'єму, ми створимо двовимірну карту активації, яка дає відповіді цього фільтра в кожному

просторовому положенні. Інтуїтивно, мережа вивчить фільтри, які активуються, коли вона бачать певний тип візуальних особливостей, таких як пляма якогось кольору на першому шарі, або врешті-решт цілі візерунки на вищих шарах мережі.[5]

Обмеженням вихідної карти об’єктів згорткових шарів є те, що вони записують точне положення об’єктів на вході. Це означає, що невеликі рухи в положенні об’єкта на вхідному зображенні призведуть до іншої карти об’єктів. Це може статися при повторному обрізанні, обертанні, зсуві та інших незначних змінах вхідного зображення. Загальноприйнятим підходом до вирішення цієї проблеми при обробці сигналу називається вибірка вниз (down sampling). Створюється версія вхідного сигналу з нижчою роздільною здатністю, яка все ще містить великі або важливі структурні елементи, без дрібних деталей, які можуть бути не такими корисними для завдання.

Вибірка вниз можна зробити за допомогою згорткових шарів, змінивши крок згортки по всьому зображенню. Але більш надійним і поширеним підходом є використання шару об'єднання (pooling). Пулінг - це новий шар, доданий після згорткового шару. Зокрема, після того, як нелінійна функція активації (наприклад, ReLU) застосована до карт об'єктів, що виводяться згортковим шаром.

В операції об'єднання найчастіше використовуються такі загальні функції:

- Середнє об’єднання: обчислює середнє значення для кожного патча на карті об’єктів.
- Максимальне об’єднання: обчислює максимальне значення для кожного патча на карті об’єктів.

Результатом використання шару об'єднання та створення об'єднаних карт об'єктів є узагальнена версія функцій, виявлених у вхідних даних. [13]

1.5. Нейронні мережі для перенесення стилю

Нейронне перенесення стилю (Neural Style Transfer) відноситься до класу програмних алгоритмів, які маніпулюють цифровими зображеннями або відеозаписами, щоб прийняти зовнішній вигляд або візуальний стиль іншого зображення. Алгоритми NST характеризуються використанням глибоких нейронних мереж для перетворення зображення. [6]

Під час виконання цього завдання подання змісту та стилю в згортковій нейронній мережі можна розділити. Тобто можна маніпулювати обома зображеннями незалежно, щоб створювати нові, сприйнятливі для людського ока зображення. Під час реконструкції вмісту краще використовувати нижні шари мережі. Реконструкція стилю відбувається за рахунок обчислення співвідношення між різними ознаками в різних шарах CNN. [7]

Для перенесення ознак використовуються архітектура GAN (генеративні змагальні мережі). В цій архітектурі одночасно навчаються дві моделі: генеративна модель G, що фіксує розподіл даних, і дискримінаційна модель D, яка оцінює ймовірність того, що вибірка взяла дані тренувань, а не G. Процедура тренування для G полягає в тому, щоб максимізувати ймовірність помилки D. Це фреймворк відповідає мінімаксній грі для двох гравців. Метод зворотного поширення помилки застосовується в обох мережах, так що генератор створює кращі зображення, тоді як дискримінатор стає більш кваліфікованим при визначенні синтезованих зображень. Генератор, як правило, є деконволюційною нейронною мережею, а дискримінатор — згортковою нейронною мережею. [8]

U-net мережа (рис. 1) базується на повністю згортковій мережі, а її архітектура була модифікована та розширена для роботи з меншою кількістю навчальних даних та отримання більш точних сегментацій. Основна ідея полягає в тому, щоб доповнити звичайну мережу послідовними шари, де оператори пулінгу замінюються операторами

апсемплінгу. Отже, ці шари збільшують роздільну здатність вихідних даних. Для того, щоб локалізувати риси високої роздільної здатності послідовні шари поєднуються з результатом збільшення розмірності. Послідовний шар згортки таким чином може навчитися отримувати більш точний результат. [9]

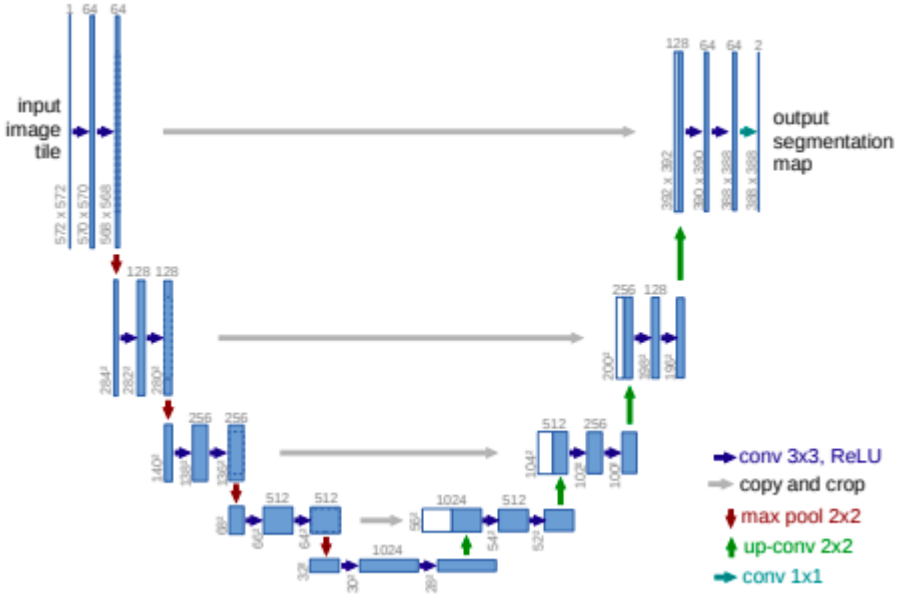


Рисунок 1. U-net архітектура

Залишкова нейронна мережа (ResNet) - це нейронна мережа, архітектура якої складається за блоків в яких з'єднуються вхідні і вихідні шари, як це показано на рисунку 2. Доведено, що залишкові мережі легше оптимізувати і вони отримують точні результати за рахунок збільшення глибини. [10]

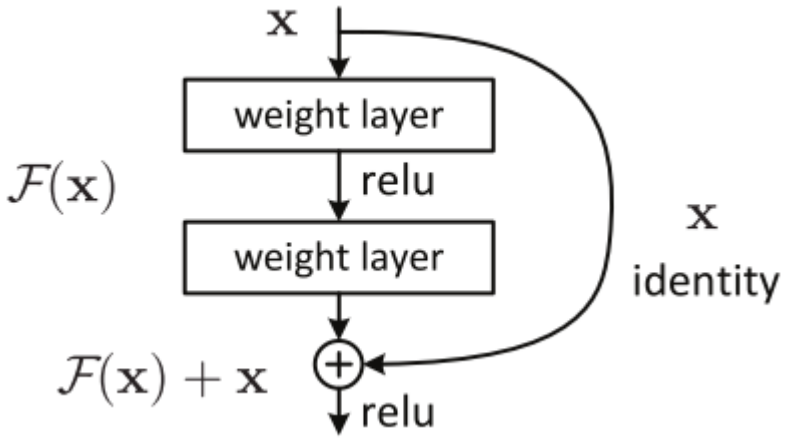


Рисунок 2. Залишковий блок ResNet

ResNet дає можливість тренувати до сотень, а то й тисяч шарів і при цьому досягає переконливих результатів без перенавчання. За допомогою цієї архітектури було покращено продуктивність багатьох додатків комп'ютерного зору, крім класифікації зображень, таких як виявлення об'єктів та розпізнавання облич. [11]

1.6. Доповнення даних

Графічне доповнення даних - це техніка, яка може штучно збільшити розмір набору навчальних даних шляхом створення модифікованої версії зображення в наборі даних. Вивчення моделей нейронних мереж шляхом поглибленого вивчення більшої кількості даних може створити більш досвідчені моделі, а розширена технологія може створити варіанти зображень, які можуть покращити здатність відповідних моделей узагальнювати свої вивчені знання в нових зображеннях.

Такі методи доповнення даних, як обрізання, заповнення та горизонтальне обертання, часто використовуються для навчання великих нейронних мереж. Однак більшість методів, що використовуються у навчанні нейронних мереж, використовують лише основні методи вдосконалення. Незважаючи на те, що архітектура нейронної мережі детально вивчена, мало уваги приділяється стратегіям збільшення даних для виявлення потужних типів розширень даних та фіксування незмінності даних.

Ефективність нейронних мереж глибокого навчання, як правило, покращується за рахунок кількості доступних даних.

Доповнення даних - це техніка штучного створення нових освітніх даних із наявних даних. Це робиться шляхом застосування методу відповідного поля, до якого належать дані (з прикладу в навчальних даних для створення нових різних навчальних даних).

Збільшення графічних даних - це, мабуть, найвідоміший тип додавання даних, і передбачає створення перетвореної версії зображення в наборі

навчальних даних, яка належить до тієї ж категорії, що і вихідне зображення. Перетворення включає ряд операцій обробки зображень, таких як зміщення, обертання, масштабування тощо.

Мета - розширити сферу навчання на нових обґрунтованих прикладах. Це означає, що модель може помітити зміни у наборі навчальних зображень. Наприклад, горизонтальне обертання зображення кота може мати сенс, оскільки фотографію можна зробити ліворуч або праворуч. Враховуючи, що модель навряд чи побачить фотографію кота, що стоїть догори дном, вертикальний поворот фотографії кота безглуздий і може бути недоречним.

Отже, очевидно, що конкретний метод удосконалення даних, що використовується у наборі навчальних даних, повинен бути ретельно відібраний, і вибір повинен здійснюватися в контексті набору навчальних даних та знань в області проблем. Крім того, може бути корисно експериментувати з методами збільшення даних окремо або спільно, щоб перевірити, чи можуть вони покращити продуктивність моделі (можливо, використовуючи лише невелику кількість прототипів даних, моделі та навчання).

Сучасні алгоритми глибокого навчання, такі як згорткові нейронні мережі (CNN), можуть вивчати функції, які не змінюють їх положення на зображенні. Однак масштабування може додатково допомогти цьому інваріантному методу трансформації та може допомогти моделі навчитися розпізнавати функції, які також є інваріантними для трансформації, такі як сортування зліва направо зверху вниз, рівні освітленості на фотографіях тощо.

Графічне збільшення даних зазвичай застосовується лише до наборів навчальних даних, а не до даних перевірки чи тестування. Це відрізняється від додаткових даних (таких як зміна розміру зображення та масштабування пікселів); вони повинні виконуватися послідовно у всіх наборах даних, які взаємодіють із моделлю. [31]

РОЗДІЛ 2. ПОБУДОВА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ

2.1. Огляд використаних технологій

Частина дипломної роботи, яка відповідає за побудову нейронної мережі, була написана на Python з використанням бібліотеки для машинного навчання TensorFlow. Бібліотека TensorFlow дозволяє будувати і тренувати нейронні мережі за допомогою інтегрованого в неї Keras API. Keras Functional API забезпечує можливості для побудови складних моделей у вигляді графу шарів, що і необхідно для даної курсової роботи.

Для побудови набору даних також було використано можливості інтегрованого в TensorFlow Keras API. Він дозволяє автоматично розподіляти набір даних на дані для тренування і тестування, а також розділяти на категорії, що необхідно для тренування.

Для створення самого проекту використовувався блокнот Google Colab. Google Colab - це безкоштовний хмарний сервіс на основі Jupyter Notebook. Основною його перевагою є можливість підключення до віддаленого GPU чи TPU для швидшої обробки і тренування масивних моделей.

Для зображення даних і проміжних результатів була використана бібліотека matplotlib, яка призначена для візуалізації.

2.2. Огляд дата-сету

Дата-сет, який використовується для тренування в цій дипломній роботі, містить загалом 3834 фотографій людей. Серед яких 1115 знімків без макіяжу і 2719 з макіяжем. Це один з найбільших наборів даних, який можна знайти у відкритому доступі на даний момент. Він містить фотографії жінок різних рас з незначними відмінностями у позах і виразах обличчя. Стилі макіяжу теж представлені різноманітні: від легкого до насиченого, лише з нафарбованими очима чи губами. Для кращої роботи деякі фотографії, які мають природній макіяж, тобто фізично він нанесений, але на фото виглядає так, ніби його немає, поміщені у категорію без макіяжу.

Всі фото мають розмір 256×256. Для тестування серед 3834 фотографій випадковим чином вибрано 100 зображень з категорії без макіяжу і 250 зображень з категорії з макіяжем. Всі інші фотографії використовуються для тренування.

Особливістю цього дата-сету є те, що він містить дані без пари, тобто у обох категоріях немає однієї і тієї ж людини. До цього були створені дата-сети, які містили таку пару, наприклад, YMU[27], VMU[28] і MIW[26]. Вони більше підходять для вивчення впливу макіяжу на ідентифікацію особистості, але не можуть бути застосовані для перенесення стилю.

Зразок фото з категорії без макіяжу зображено на рис. 3, а приклад з категорії без макіяжу на рис. 4.

Рисунок 3. Зразок фото без макіяжу

Рисунок 4. Зразок фото з макіяжем

2.3. Формулювання ідеї

Для вирішення проблеми зняття макіяжу використаємо машинне навчання без вчителя з архітектурою CycleGAN, оскільки ця архітектура якраз розроблена для даних без пари.

Нехай X і Y це масиви з зображень без макіяжу і з макіяжем відповідно. X характеризується колекцією фотографій без макіяжу $\{x_i\}_{i=1,\dots,N}$, $x_i \in X$, де N кількість фото без макіяжу. Y множина фотографій з макіяжем:

$\{y_i\}_{i=1,\dots,M}$, $x_i \in Y$, де M - кількість фото з макіяжем. $Y^\beta \subset Y$ стосується підмножини, що містить певний стиль макіяжу β. Якщо $y_i \in Y^\beta$, позначимо як y_i^β.

Ідея полягає в тому, щоб натренувати дві окремі генеративні нейронні мережі G і F, одна для перенесення макіяжу, а інша для зняття. Нейронна мережа $F : X \times Y^\beta \rightarrow Y^\beta$ витягує шар з макіяжем з y^β і застосовує його до x з збереженням особистості. Результат $F(x, y^\beta)$ має належати множині Y^β. Нейронна мережа $G : Y \rightarrow X$ навчається як знімати макіяж з збереженням особистості. G і F є не збалансованими функціями. Вхідні дані для F це фото без макіяжу. Вхідні дані для G - це одне фото з макіяжем. За повністю успішного виконання вихідні дані з функції G можуть бути використані як вхідні дані для F.

2.4. Функції втрати

Нехай x вибрана з X згідно з деяким розподілом P_x і y^β вибрана з Y згідно з деяким розподілом P_y. Спираючись на особливості перенесення макіяжу, виберемо наступні функції втрати.

Для мережі G використаємо втрату змагання (adversarial loss) (формула 3), щоб результати G виглядали схоже на обличчя з множини Y.

$$L_G\left(G, D_Y\right) = E_{y\sim PY}\left[logD_Y(y)\right] + E_{x\sim PX_y\sim PY}\left[log\left(1 - D_Y(G(x,y))\right)\right]$$

(Формула 3. Втрата змагання для G)

Де дискримінатор Dy намагається відрізнити реальний зразок з множини Y від згенерованого $G(x, y^\beta)$ і генератор G намагається згенерувати зображення, яке Dy не взмозі відрізнити від реального.

Для мережі F використаємо втрату змагання (формула 4) таку, щоб генератор F генерував зображення, які неможливо відрізнити від облич без макіяжу з множини X.

$$LF(F, DX) = Ex\sim PX[logDX(x)] + Ey\beta\sim PY[log(1 - DX(F(y\beta)))]$$

(Формула 4. Втрата змагання для F)

Визначимо втрату ідентичності як L_I *(G, F)* (формула 5). Будемо використовувати генератор F для збереження ідентичності в процесі перенесення стилю макіяжу. Використаємо норму L1 для того, щоб знайти різницю між *F (G(x, yβ))* і *x*.

$$L_I(G,F) = E_{x\sim PX, y\beta\sim PY}\left[\left|\left|F\left(G\left(x, y^{\beta}\right)\right) - x\right|\right|_I\right]$$

(Формула 5. Втрата ідентичності)

Щоб база зображення на фото, тобто обличчя людини і фон за нею, залишались незмінними визначимо циклічну втрату (див. формула 6). Тут F і G генератори відповідних генеративних мереж.

$$L\_S(G,\ F)\ =\ E_{x\sim PX,\ y\beta\sim PY}\left[||F(G(x,\ y)\ -\ x||\right]\ +\ E_{x\sim PX,\ y\beta\sim PY}\left[||(G(F(y),\ y)\ -\ y)||\right]$$

(Формула 6. Циклічна втрата)

Загальною ідеєю є те, що ми намагаємось мінімізувати цільову функцію для генераторів і максимізувати для дискримінаторів. Загальна втрата визначається за формулою 7. Де $\lambda_G = \lambda_F = 0.5$.

$$L = \lambda_G L_G + \lambda_F L_F + L_I + L_S$$

(Формула 7. Загальні втрати)

2.5. Реалізація

Спочатку всі фотографії завантажуються в середовище Google Colab, за допомогою вбудованих можливостей Keras API вони розподіляються на дані для тренування і тестування. Після цього кожному зображенню привласнюється категорія «макіяж» чи «без макіяжу».

Для покращення результатів моделі на тестових даних було збільшено їх варіативність за допомогою доповнення даних (data augmentation). А саме, деякі зображення було відображено дзеркально випадковим чином, як показано на рисунку 5.

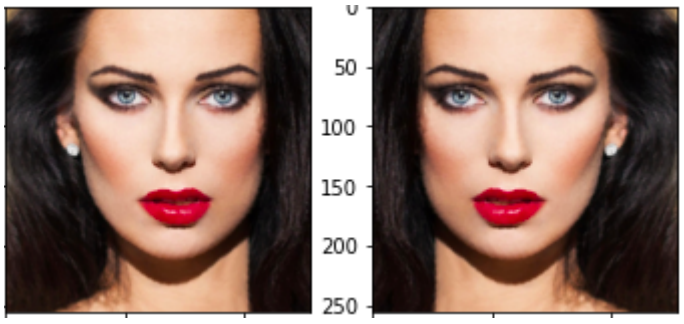


Рисунок 5. Зразок доповнення даних

Наступним кроком є нормалізація з середнім значенням 0 і дисперсією 1, після якої всі дані належать проміжку [-1, 1]. Крім цього до майже всіх шарів моделі було додано нормалізацію екземплярів, оскільки вона покращує точність результатів моделі саме для проблеми перенення стилю [15].

Далі було побудовано моделі для дискримінатора (табл. 1) і генераторів G (табл. 2) та F (табл. 3). В таблицях присутні наступні позначення:

- Conv - згортковий шар,
- Deconv - зворотній згортковий шар,
- F - кількість фільтрів,
- K - розмір ядра згортки,
- S - розмір кроку,
- P - розмір заповнення (padding), де s означає зберегти розмір вихідних даних, z - заповнити нулями, r - рефлексивне заповнення,
- Dr - використати дропаут.
- IN - нормалізація екземплярів

Таблиця 1. Архітектура моделі дискримінаторів

Етап	Налаштування шару
Зменшення розмірності	Conv (F64, K4, S2, Ps), ReLU
	Conv (F128, K4, S2, Ps), IN, ReLU
	Conv (F256, K4, S2, Ps), IN, ReLU
	Conv (F512, K4, S1, Pz), IN, LeakyReLU
Вихід	Conv (F1, K4, S1, Pz)

Таблиця 2. Архітектура моделі генератора G для зняття макіяжу

Етап	Налаштування шару
Зменшення розмірності (downsample)	Conv (F64, K4, S2, Ps), ReLU (1)
	Conv (F128, K4, S2, Ps), IN, ReLU (2)
	Conv (F256, K4, S2, Ps), IN, ReLU (3)
	Conv (F512, K4, S2, Ps), IN, ReLU (4)
	Conv (F512, K4, S2, Ps), IN, ReLU (5)
	Conv (F512, K4, S2, Ps), IN, ReLU (6)
	Conv (F512, K4, S2, Ps), IN, ReLU (7)
	Conv (F512, K4, S2, Ps), IN, ReLU (8)
Збільшення розмірності (upsample) та встановлення з’єднань	Deconv (F512, K4, S2, Ps), IN, ReLU, Dr об'єднати з (7)
	Deconv (F512, K4, S2, Ps), IN, ReLU, Dr об'єднати з (6)
	Deconv (F512, K4, S2, Ps), IN, ReLU, Dr об'єднати з (5)
	Deconv (F512, K4, S2, Ps), IN, ReLU, об'єднати з (4)
	Deconv (F256, K4, S2, Ps), IN, ReLU, об'єднати з (3)
	Deconv (F128, K4, S2, Ps), IN, ReLU, об'єднати з (2)
	Deconv (F64, K4, S2, Ps), IN, ReLU, об'єднати з (1)
Вихідні дані	Deconv (F3, K4, S2, Ps), Tanh

Таблиця 3. Архітектура моделі генератора F для нанесення макіяжу

Етапи	Налаштування шарів
Вхідний шар	Conv (F64, K7, S1, Pr3), IN, ReLU
Зменшення розмірності	Conv (F128, K3, S2, Pz), ReLU
	Conv (F256, K3, S2, Pz), ReLU
12 залишкових блоків	Залишковий блок: Conv (F256, K3, S1, Pr1), IN, ReLU
Збільшення розмірності	Deconv (F64, K3, S2, Ps), IN, ReLU
	Deconv (F32, K3, S2, Ps), IN, ReLU
Вихідний шар	Conv (F3, K7, S1, Pr3), Tanh

```
def train_step_original(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:

        fake_y = generator_g(real_x, training=True)
        cycled_x = generator_f([fake_y, real_x], training=True)

        fake_x = generator_f([real_y, real_x], training=True)
        cycled_y = generator_g(fake_x, training=True)

        # same_x and same_y are used for identity loss.
        same_y = generator_f([real_y, real_x], training=True)
        same_x = generator_g(real_x, training=True)

        disc_real_x = discriminator_x(real_x, training=True)
        disc_real_y = discriminator_y(real_y, training=True)

        disc_fake_x = discriminator_x(fake_x, training=True)
        disc_fake_y = discriminator_y(fake_y, training=True)

        # calculate the loss
        gen_g_loss = generator_loss(disc_fake_y)
        gen_f_loss = generator_loss(disc_fake_x)

        total_cycle_loss = calc_cycle_loss(real_x, cycled_x) + calc_cycle_loss(real_y, cycled_y)

        # Total generator loss = adversarial loss + cycle loss
        total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_x, same_x)
        total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_y, same_y)

        disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
        disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)
```

Рисунок 6. Крок тренування.

Наступним кроком є тренування нейронної мережі з нуля, використовуючи оптимізатор Adam з параметром 0.0002 і розміром пакету 1.

Крок тренування містить в собі 4 нейронні мережі і має вигляд, як на рисунку 6.

Щоб не втратити результати тренувань використовувався менеджер черпоінтів (рис. 7). Кожну другу епоху за допомогою нього зберігались натреновані ваги.

```
[ ]  from google.colab import files

     checkpoint_path = F"/content/gdrive/My Drive/Colab Notebooks/checkpointsTest1/Previous"

     ckpt = tf.train.Checkpoint(generator_g=generator_g,
                                generator_f=generator_f,
                                discriminator_x=discriminator_x,
                                discriminator_y=discriminator_y,
                                generator_g_optimizer=generator_g_optimizer,
                                generator_f_optimizer=generator_f_optimizer,
                                discriminator_x_optimizer=discriminator_x_optimizer,
                                discriminator_y_optimizer=discriminator_y_optimizer)

     ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=3)

     # if a checkpoint exists, restore the latest checkpoint.
     if ckpt_manager.latest_checkpoint:
       ckpt.restore(ckpt_manager.latest_checkpoint)
       print ('Latest checkpoint restored!')

     Latest checkpoint restored!
```

Рисунок 7. Збереження даних тренування.

2.6. Результати тренування

Тренування відбувалось в Google Colab з підключенням віддаленого середовища виконання і ресурсів. Модель була натренована впродовж 60 епох, кожна епоха займала в середньому 14 хвилин. Отже, все тренування зайняло приблизно 14 годин.

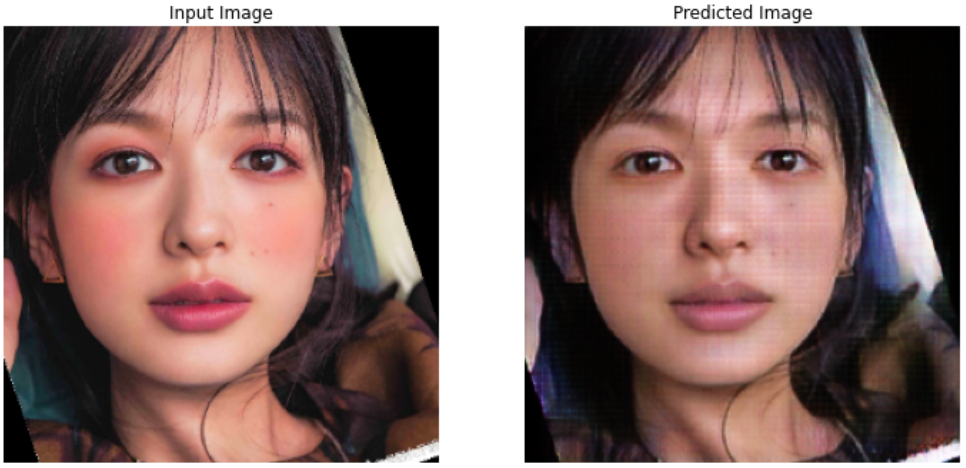


Рисунок 8. Результат зняття макіяжу на тренувальних даних

Результати зняття макіяжу на тренувальних даних показані на рисунку 8. Можна побачити, що успішно було знято тіні і помаду, також змінився тон шкіри, він виглядає більш природньо без нанесеного тону і яскравих рум’ян. Риси обличчя після обробки не змінились, проте з’явились шуми на фото.

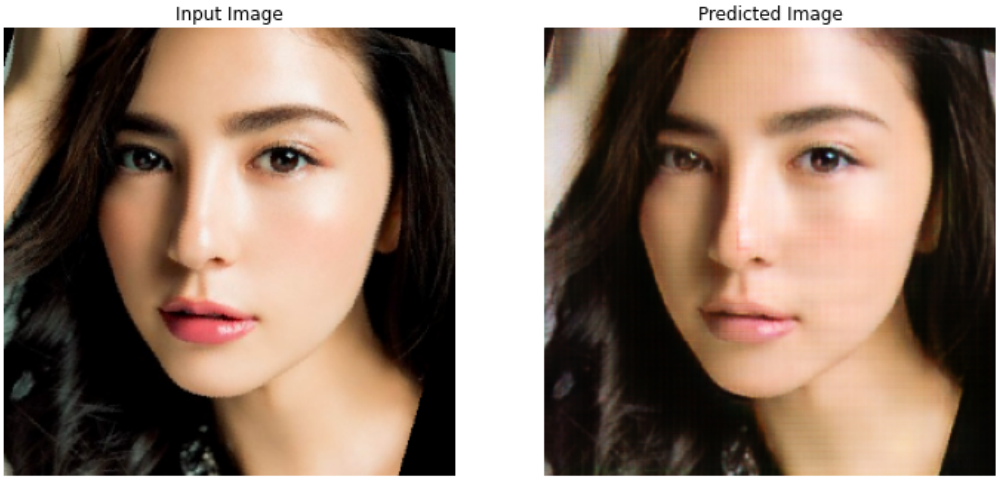


Рисунок 9. Результат зняття макіяжу на тестових даних

На тестових даних (рис. 9) зняття макіяжу виглядає однаково добре, як і на тренувальних. Видно, що колір губ і тон шкіри змінились в сторону більш натурального.

Дуальною задачею до зняття макіяжу було його нанесення. Результати на тренувальних даних зображені на рисунку 10. На фото видно, що перенесений макіяж яскравий, перенеслись тіні, колір помади, колір брів, а також змінився тон обличчя. Але саме зображення стало менш якісним та з

додатковими шумами. Також через особливість побудованої нейронної мережі згенерований макіяж є єдиним, тобто переноситься загальний стиль без референса.

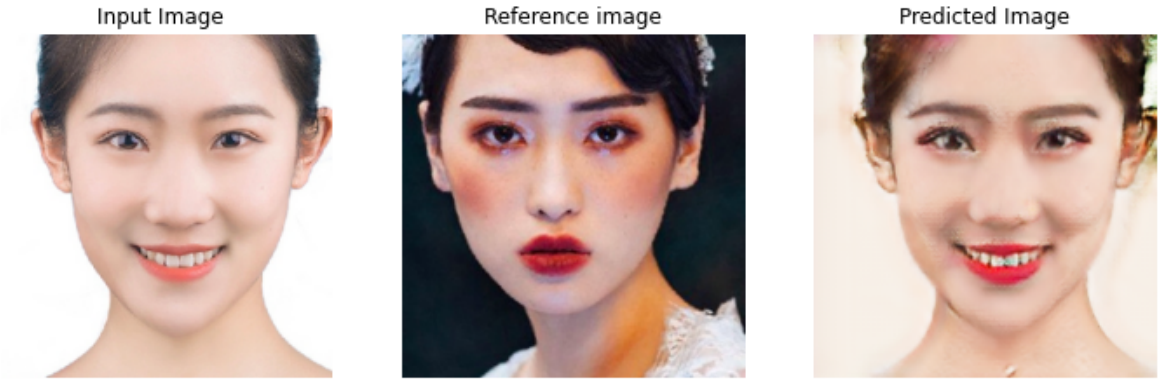


Рисунок 10. Результат нанесення макіяжу на тренувальних даних

Результат нанесення макіяжу на тестових даних продемонстровано на рисунку 11. Тут видно наскільки добре перенеслась губна помада, також можна помітити тіні на очах. Проте також цей приклад демонструє сильну зміну кольору зображення, при тому яка не відповідає референсу на якому вона навчалась.

Рисунок 11. Результат нанесення макіяжу на тестових даних

Окрім відносно хороших результатів також були невдалі. На рисунку 12 показаний такий приклад. Можна побачити, що шумів дуже багато, і хоча колір шкіри змінений, але макіяж з очей зовсім не знявся. Така поведінка можлива через поворот голови людини на зображенні.

Рисунок 12. Приклад невдалого зняття макіяжу

РОЗДІЛ 3. СТВОРЕННЯ ДОДАТКУ НА ANDROID

3.1. Побудова архітектури додатку

Для написання додатку було обрано мову програмування Java і середовище розробки Android Studio. Тестування усіх функцій проводилось на телефоні Xiaomi 9 SE, на якому встановлена 9 версія Android.

Спочатку було розроблено макет додатку, який містив інформацію про основні сторінки, їхній дизайн та переміщення між ними. Для зручності було використано вбудовану компоненту навігації і побудовано два графи навігації.

Перший граф (рис. 13) створюється в основному класі діяльності і складається з двох фрагментів і двох класів діяльностей.

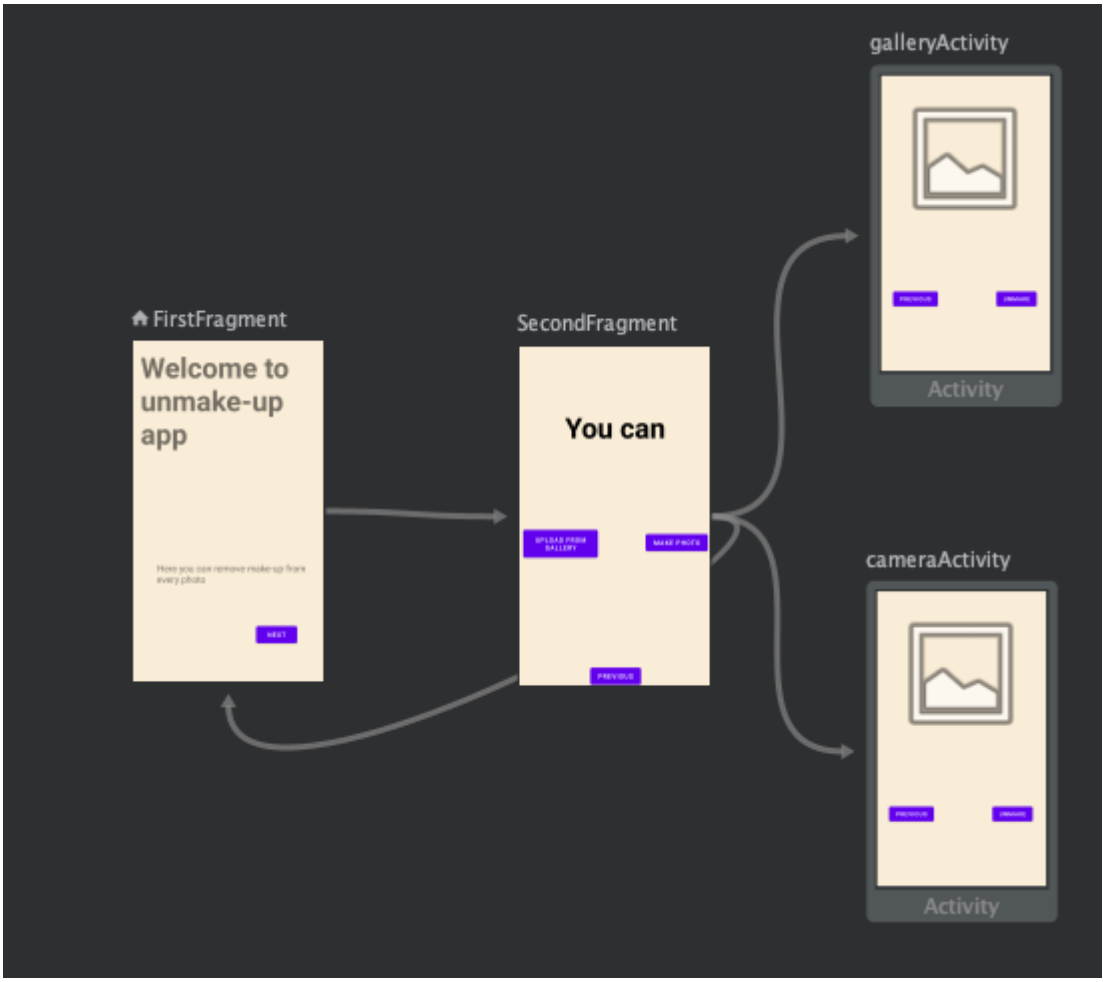


Рисунок 13. Перший навігаційний граф

Другий граф (рис. 14) складається з двох фрагментів і посилання на попередній граф. Він викликається з класів діяльності попереднього графа, а саме з galleryActivity або cameraActivity.

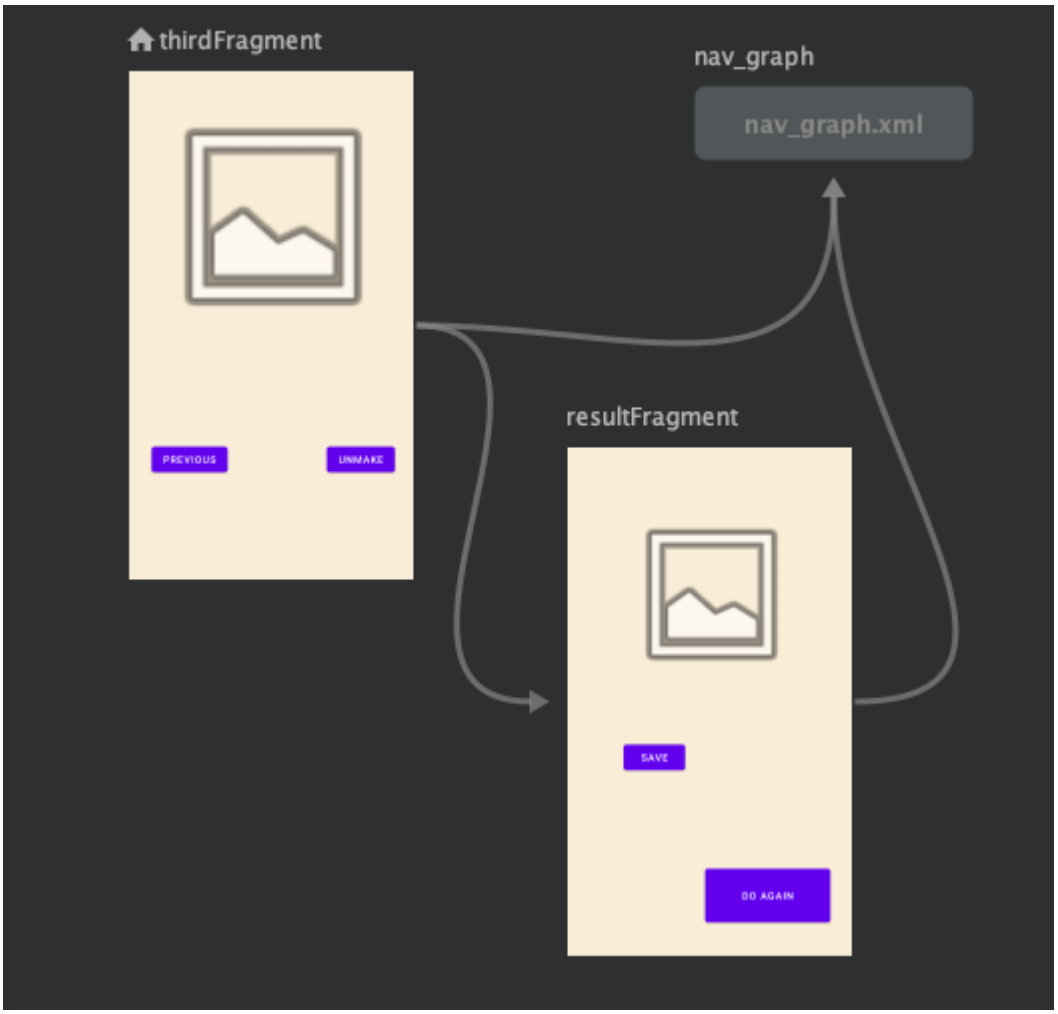


Рисунок 14. Другий навігаційних граф

3.2. Реалізація основних компонентів

На першій сторінці додатку (рис. 15 зліва) розміщений заголовок і підзаголовок, який розказує користувачу призначення додатку. Після натискання кнопки “next” з’являється друга сторінка.

На другій сторінці (рис. 15 справа) користувач може обрати спосіб в який завантажуватиме зображення. Він може завантажити його з галереї або зробити фото. У другому випадку відкриється нативна програма камери.

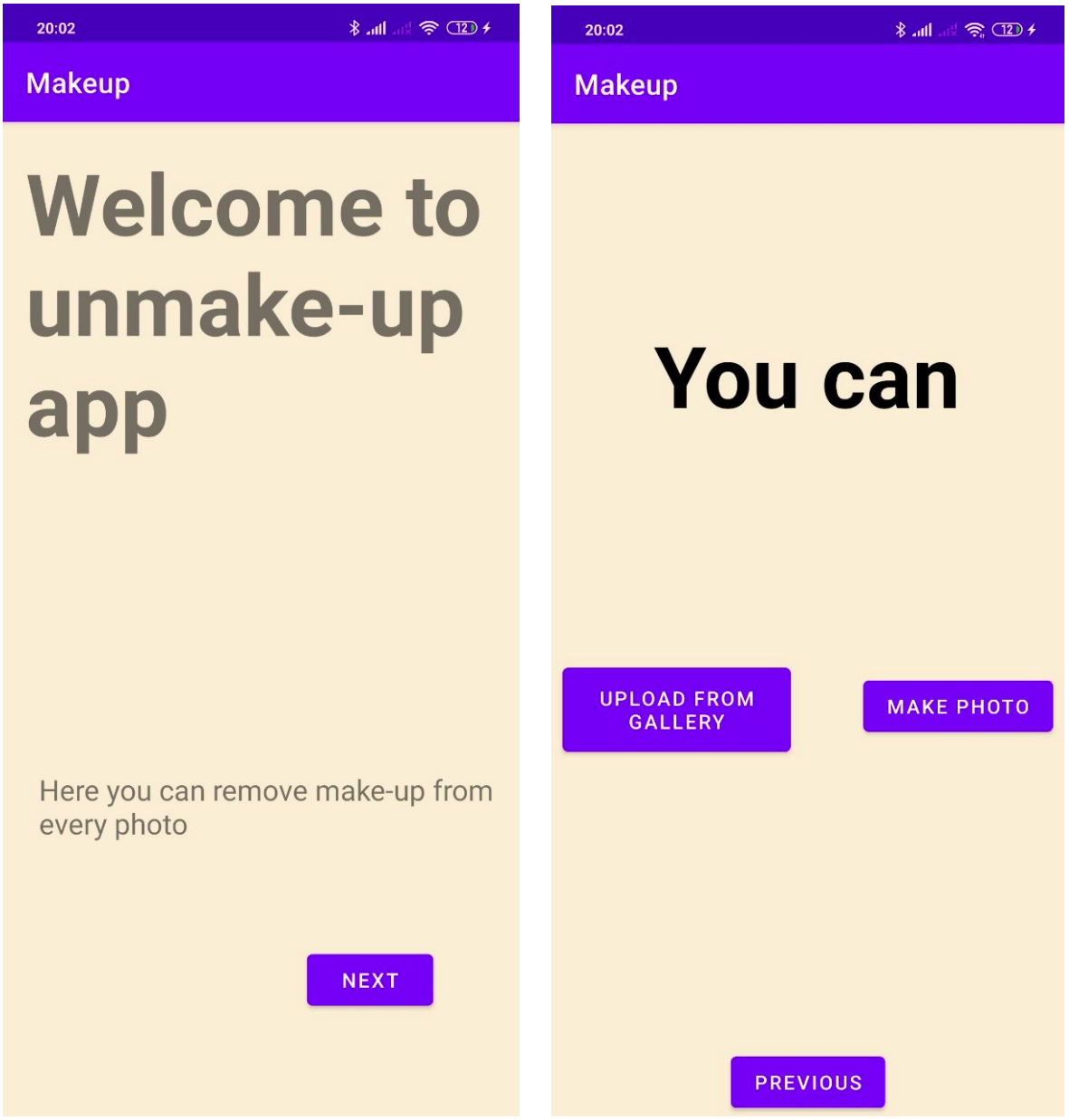


Рисунок 15. Зліва перша сторінка додатку, а справа - друга

Після цього відбувається перевірка фото на наявність обличчя, щоб уникнути обробки зображення на якому його немає. Для того, щоб виконати цю перевірку, до додатку було підключено сервіс ML Kit.

Існує два способи за допомогою яких можна інтегрувати розпізнавання обличчя в додатку. Перший - це вбудована модель, яка стає частиною додатка, та віддалена модель, яка залежить від служб Google Play. Обидва варіанти мають свої переваги та недоліки. В першому випадку модель займає більше пам’яті в додатку, а в другому необхідне підключення

до мережі. В іншому ці дві моделі однакові. В цій дипломній роботі був обраний перший варіант.

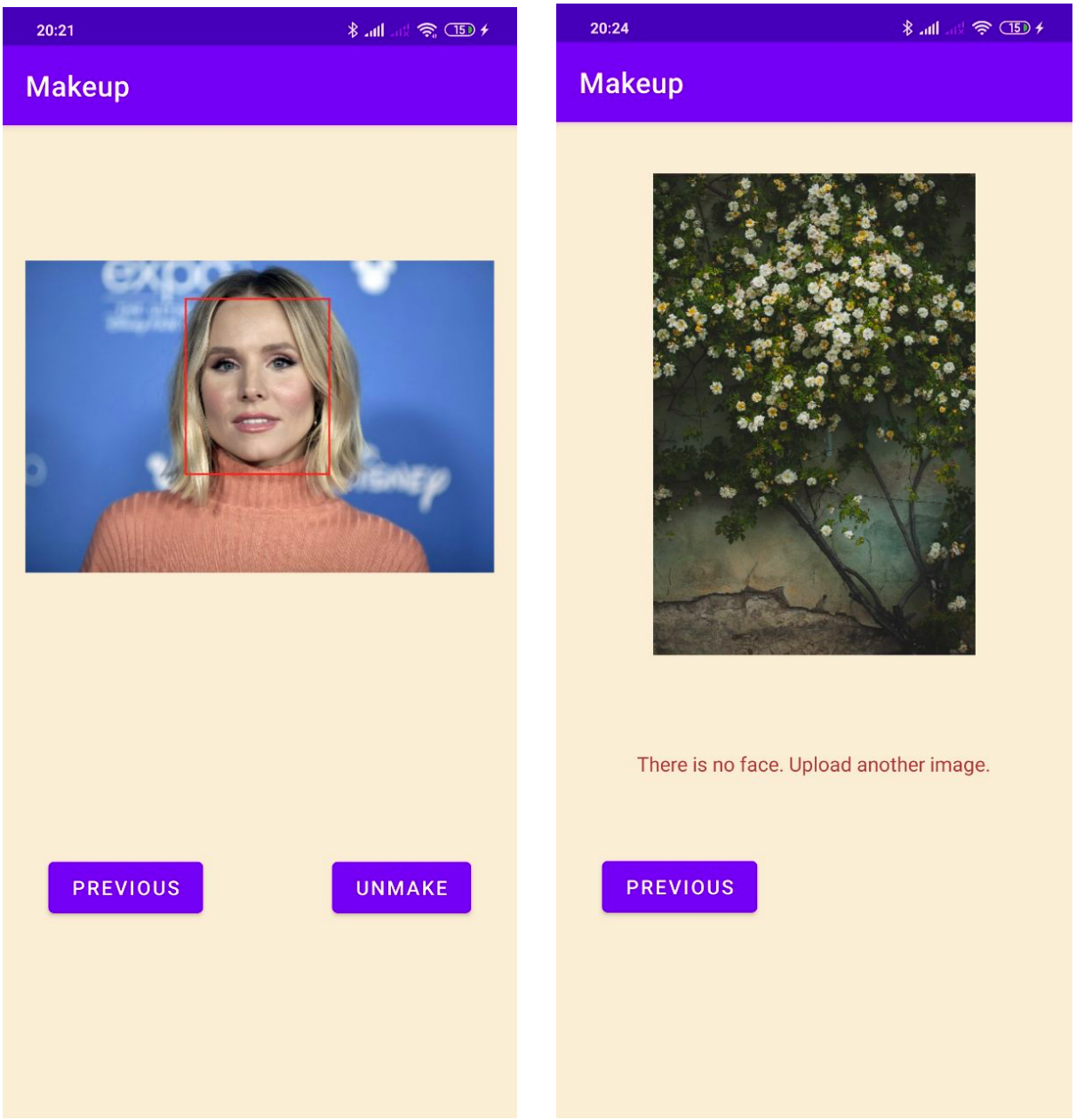


Рисунок 16. Зліва третя сторінка без помилки, а справа з попередженням

У випадку, якщо обличчя було знайдено, навколо нього малюється червоний прямокутник, як це видно на рисунку 16 зліва. Проте, якщо обличчя немає, то виводиться повідомлення про те, що користувач має завантажити інше зображення, це показано на рисунку 16 справа. Для того, щоб користувач не міг обробити таке фото, була прибрана кнопка для зняття макіяжу.

3.3. Оптимізація моделі під мобільний додаток

Оскільки модель була написана за допомогою бібліотеки TensorFlow, було вирішено використати TensorFlow Lite для інтеграції моделі в додаток, оскільки він має підтримку Java. TensorFlow Lite надає набір інструментів, що підтримують машинне навчання на пристроях. Таким чином допомагаючи розробникам легко запускати моделі на мобільних пристроях.

Першим кроком було перетворити збережену натреновану модель, яка зберігалась на Google Drive в форматі SaveModel, в модель TensorFlow Lite. Щоб це зробити треба було використати TFLiteConverter на Python в Google Colab. Перетворена модель мала розмір 217 мб.

Перед тим як інтегрувати натреновану нейронну мережу в мобільний додаток було вирішено її оптимізувати, оскільки мобільні пристрої часто мають обмежену пам’ять та обчислювальні можливості. До моделей можуть застосовуватися різні оптимізації, щоб вони могли працювати в рамках цих обмежень. Проте оптимізація потенційно може призвести до зміни в точності моделі. Ці зміни залежать від індивідуальної моделі.

Зараз TensorFlow Lite підтримує оптимізацію за допомогою квантування, обрізання та кластеризації. Для моделі, спроектованої в дипломній роботі, найкраще підходить оптимізація за допомогою квантування динамічного діапазону. Вона не має умов на вхідні дані, а також виконується безпосередньо після тренування. [30]

Отже, після того, як модель була оптимізована за допомогою квантування динамічного діапазону її розмір зменшився і досяг 157 мб.

Останнім кроком було інтегрувати її в додаток. Для цього файл моделі з розширенням tflite був завантажений в проект. І потім за допомогою вбудованої бібліотеки TensorFlow Lite модель була підключена.

3.4. Попередня обробка зображення

Після того, як було визначено чи присутнє обличчя на фото, відбувається його обробка. На вхід до нейронної мережі має подаватись Tensor розмірністю (1, 256, 256, 3). Щоб обрізати фотографію навколо обличчя використовується детектор облич, який вже згадувався до цього. На виході ми отримуємо фото квадратної форми, але ще великої розмірності. Тому перед тим, як передавати його в мережу, розширення фото було зменшено.

У TensorFlow Lite на Java ще немає зручного інтерфейсу, якщо модель не має метаданих, тому фотографія подається у форматі ByteBuffer. Для того, щоб перетворити BitMap у ByteBuffer, було використано функцію зображену на рисунку 17. Важливим нюансом є те, що дані зображення мають формат Float32 і при конвертації це має бути враховано, а саме, під час виділення пам'яті під байтовий буфер ми множимо значення на 4.

```
private static ByteBuffer convertBitmapToByteBuffer(Bitmap bitmap) {
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * BATCH_SIZE * inputSize * inputSize * PIXEL_SIZE);
    byteBuffer.order(ByteOrder.nativeOrder());
    int[] intValues = new int[inputSize * inputSize];
    bitmap.getPixels(intValues, offset: 0, bitmap.getWidth(), x: 0, y: 0, bitmap.getWidth(), bitmap.getHeight());
    int pixel = 0;
    for (int i = 0; i < inputSize; ++i) {
        for (int j = 0; j < inputSize; ++j) {
            final int val = intValues[pixel++];
            byteBuffer.putFloat((((val >> 16) & 0xFF)-IMAGE_MEAN)/IMAGE_STD);
            byteBuffer.putFloat((((val >> 8) & 0xFF)-IMAGE_MEAN)/IMAGE_STD);
            byteBuffer.putFloat((((val) & 0xFF)-IMAGE_MEAN)/IMAGE_STD);
        }
    }
    return byteBuffer;
}
```

Рисунок 17. Функція для конвертації BitMap в ByteBuffer

І вже після цього ми можемо обробити фотографію і отримати вихідне зображення також у форматі ByteBuffer, який потрібно знову конвертувати в формат BitMap.

3.5. Результати роботи

Результати роботи додатку можна побачити на рисунку 18 справа, зліва видно, яка фотографія завантажувалась в додаток. Порівнюючи з результатами на рисунку 8, можна сказати, що оптимізація моделі дійсно завдала певних втрат якості виконання. На зображенні присутні шуми, а також фон за обличчям змінив колір. Окрім цього через попередню обробку зображення, описану в пункті 3.4, фото стало менш якісним.

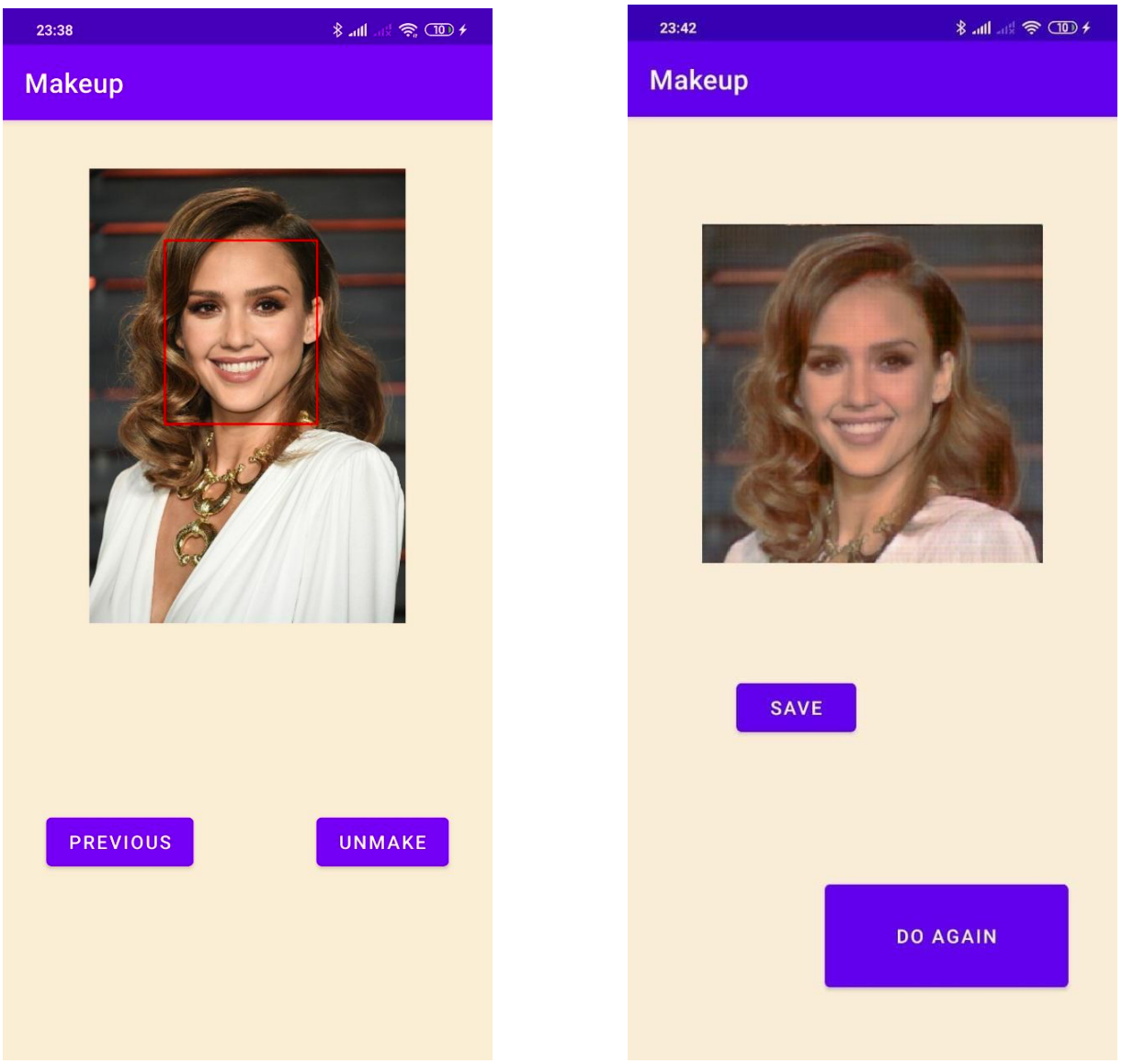


Рисунок 18. Результати роботи додатку

Код додатку можна побачити на Github за посиланням: https://github.com/NataNorth/graduate-work.

ВИСНОВОК

Результатом виконання цієї дипломної роботи є мобільний додаток для платформи Android з використанням власної побудованої та натренованої нейронної мережі для зняття макіяжу.

Згідно з метою роботи, було проведено всі заплановані етапи дослідження та розробки:

1. Було систематизовано існуючі наукові роботи на тему перенесення стилю з використанням даних без пари і з парою, зняття та накладання макіяжу. А також було визначено особливості кожної проаналізованої роботи.
2. Було досліджено програмні засоби для написання нейронних мереж з можливістю подальшого інтегрування в мобільний додаток на Android. І вибрано бібліотеку TensorFlow на Python, яка мала можливість конвертації в TensorFlow Lite на Java.
3. Знайдено необхідний набір даних, які не містили пари, що дозволило застосувати архітектуру CycleGAN, а також який мав достатню кількість екземплярів, для якісного дослідження і тренування нейронної мережі.
4. Була спроектована модель нейронної мережі, яка складалась з двох GAN. Для цього були спочатку були описані функції втрати і ідеї до реалізації, а потім було реалізовано і натреновано саму модель.
5. Було спроектовано макет і реалізовано додаток, який містив 3 класи діяльності, 4 фрагменти і 2 допоміжні класи для детекції обличчя на фото і зняття макіяжу.
6. Було оптимізовано під мобільний пристрій та інтегровано побудовану нейронну модель в додаток з використанням вбудованого набору інструментів TensorFlow Lite

7. Було спочатку продемонстровано результати роботи спроектованої моделі на тестових даних в Google Colab, а потім показано роботу оптимізованої нейронної мережі на реальних даних в додатку.
8. Показано результати роботи додатку, його функціонал та навігаційний граф переходу між фрагметами та діяльностями.

Отже, об’єкт дослідження цієї дипломної роботи може бути розглянутий і проаналізований глибше, адже були виявлені області в яких можливе покращення, а саме:

1. Було б доцільним додати сегментацію обличчя, як попередню обробку зображення. Це покращить не тільки якість результатів виконання, а й швидкість оскільки не потрібно буде обробляти все зображення.
2. Виділяти стиль макіяжу з сегментованого обличчя за допомогою спеціально розробленої нейронної мережі.
3. Збільшити варіантивність даних за допомогою доповнення даних, створеного спеціально для доповнення даних, які містять обличчя.
4. Оптимізувати модель не тільки після тренування, а й підчас, що дозволить отримати результати без значної втрати якості.

Також може бути покращений об’єкт розробки цієї дипломної роботи.

1. Може бути розширений його функціонал, наприклад:
    a. можливість реєстрації,
    b. накладання макіяжу,
    c. завантаження фотографій в базу даних;
2. Зменшений його розмір за рахунок:
    a. краще оптимізованої і меншої моделі,
    b. використання хмарних технологій для інтеграції нейронної мережі,
    c. використання Google Play для вивантаження моделі для детекції облич;

3. Покращений дизайн.

Результати проведеного дослідження можуть бути використані для подальшого аналізу в області нейронних мереж, розроблених для перенесення стилю, а конкретно нанесення і зняття макіяжу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Android Developers. Documentation. Guides. Application Fundamentals [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://developer.android.com/guide/components/fundamentals (дата звернення 11.05.2021) – Назва з екрана.
2. Android Developers. Android Studio. Guide. Overview. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://developer.android.com/studio/intro (дата звернення 11.05.2021) – Назва з екрана.
3. Android Developers. Documentation. Guides. Introduction to Activities [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://developer.android.com/guide/components/activities/intro-activities (дата звернення 11.05.2021) – Назва з екрана.
4. IBM Cloud Learn Hub. Neural Networks [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.ibm.com/cloud/learn/neural-networks (дата звернення 11.05.2021) – Назва з екрана.
5. CS231n Convolutional Neural Networks for Visual Recognition [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://cs231n.github.io/convolutional-networks/ (дата звернення 11.05.2021) – Назва з екрана.
6. Neural Style Transfer. Вікіпедія [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://en.wikipedia.org/wiki/Neural_Style_Transfer (дата звернення 11.05.2021) – Назва з екрана.
7. A Neural Algorithm of Artistic Style/ Gatys, Leon A.; Ecker, Alexander S.; Bethge, Matthias – 26 серпня 2015.

8. Generative Adversarial Networks. Proceedings of the International Conference on Neural Information Processing Systems/ Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua – 2014 – (NIPS 2014). pp. 2672–2680.
9. U-Net: Convolutional Networks for Biomedical Image Segmentation/ Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas – 2015.
10. Deep Residual Learning for Image Recognition/ He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian – 2015.
11. An Overview of ResNet and its Variants. Towards Data Science. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035 (дата звернення 11.05.2021) – Назва з екрана.
12. Ke-Lin Du, Swamy M. N. S., Neural Networks and Statistical Learning, Springer-Verlag London, 2014.
13. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (дата звернення 11.05.2021) – Назва з екрана.
14. Ioffe S., Szegedy C. — Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2016.
15. Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky - Instance Normalization: The Missing Ingredient for Fast Stylization, 2016.
16. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5967–5976, July 2017.

17. Jun-Yan Zhu, Taesung Park, Phillip Isola Alexei A. Efros - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, in BAIR, 2017.
18. Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In 2016 European Conference on Computer Vision (ECCV), 2016.
19. Tingting Li, Ruihe Qian, Chao Dong, Si Liu, Qiong Yan, Wenwu Zhu, and Liang Lin. 2018. BeautyGAN: Instance-level Facial Makeup Transfer with Deep Generative Adversarial Network
20. Huiwen Chang, Jingwan Lu, Fisher Yu, and Adam Finkelstein. «PairedCycleGAN: Asymmetric Style Transfer for Applying and Removing Makeup.»CVPR 2018, June 2018.
21. Hung-Jen Chen, Ka-Ming Hui, Sishui Wang, Li-Wu Tsao, Hong-Han Shuai, Wen-Huang Cheng, and National Chiao Tung. Beautyglow : On-demand makeup transfer framework with reversible generative network. In *CVPR*, 2019
22. Si Liu, Xinyu Ou, Ruihe Qian, Wei Wang, and Xiaochun Cao. 2016. Makeup like a superstar: deep localized makeup transfer network. In the Association for the Advance of Artificial Intelligence. AAAI Press, 2568–2575.
23. Qiao Gu, Guanzhi Wang, Mang Tik Chiu, Yu-Wing Tai, and Chi-Keung Tang. Ladn: Local adversarial disentangling net- work for facial makeup and de-makeup. In *ICCV*, 2019
24. Makeup transformation video watch time data [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.thinkwithgoogle.com/data/makeup-transformation-video-watch-time-data/ (дата звернення 9.05.2021) – Дата публікації : 06.2018. – Назва з екрана.
25. App Store Preview [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу:

https://apps.apple.com/us/app/makeapp-ai-based-editor/id1236717369 (дата звернення 9.05.2021) – Назва з екрана.

26. Cunjian Chen, Antitza Dantcheva, and Arun Ross. 2013. Automatic facial makeup detection with application in face recognition. InBiometrics (ICB), 2013 Interna-tional Conference on. IEEE, 1–8
27. Cunjian Chen, Antitza Dantcheva, and Arun Ross. 2016. An ensemble of patch-based subspaces for makeup-robust face recognition.Information fusion32 (2016), 80–92
28. Antitza Dantcheva, Cunjian Chen, and Arun Ross. 2012. Can facial cosmetics affect the matching accuracy of face recognition systems?. InBiometrics: Theory, Applications and Systems (BTAS), 2012 IEEE Fifth International Conference on. IEEE, 391–398
29. Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky Instance Normalization: The Missing Ingredient for Fast Stylization, 2017
30. TensorFlow. Guide. Post-training dynamic range quantization [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.tensorflow.org/lite/performance/post_training_quant (дата звернення 12.05.2021) – Назва з екрана.
31. How to Configure Image Data Augmentation in Keras. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ (дата звернення 12.05.2021) – Назва з екрана.