

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: **«Автоматизоване видалення фону растрового зображення,
отриманого за допомогою смартфона»**

Виконала: студентка 3-го року
навчання,

Спеціальності
122 «Комп'ютерні науки»

Яцків Катерини Богданівни

Керівник Афонін А.О.,
канд. фіз-мат. наук, доцент,
«16» травня
2021 р.

Зміст

Анотація	1
Вступ	2
Розділ 1. Дослідження та аналіз готових рішень в даній області	3
1.1 Remove BG.....	3
1.2 Canva PRO.....	4
1.3 Clipping magic.....	6
1.4 Slazzer.....	8
1.5 Adobe Photoshop – Інструмент «Виділення об’єкта»	10
Розділ 2. Дослідження різних методів та підходів до вирішення даної проблеми	12
2.1 Загальні підходи до вирішення проблеми.....	12
2.2 Аналіз різних моделей розпізнавання об’єктів.....	14
2.2.1 Ознаки Хаара.....	14
2.2.2 Гістограми орієнтованих градієнтів	15
2.2.3 Оператор Соболя	17
2.2.4 R-CNN, Mask R-CNN та інші CNN моделі	18
2.2.5 Silent Object Detection.....	19
2.3 Бібліотека OpenCV	20
Розділ 3. Практичне застосування моделей машинного навчання для видалення фону	21
3.1 Видалення фону методами бібліотеки OpenCV	21
3.1.1 Теоретичні відомості.....	21
3.1.2 Використання алгоритму GrabCut на Java	22
3.2 Видалення фону використовуючи модель U ² -Net	24

3.2.1 Архітектура додатку для видалення фону	25
3.2.2 Приклади застосування.....	36
Висновки	39
Список використаних джерел.....	40
Додаток А (обов'язковий).....	
Зображення обрані для аналізу веб-застосунків для видалення фону	44
Додаток Б (ознайомчий)	
Код програми для видалення фону методами бібліотеки OpenCV	46

Анотація

У роботі розглядаються різні методи та моделі визначення об'єкту на фотографії методами машинного навчання та комп'ютерного бачення, а також проводиться аналіз використання деяких алгоритмів на практиці. Результатом проведеного дослідження є застосунок та серверна частина, які у режимі реального часу видаляють фон зі зробленої фотографії, та зберігають результат у сховище даних смартфона.

Вступ

Проблема автоматизованого відокремлення заднього плану від переднього є проблемою розпізнавання та відокремлення об'єктів, що в свою чергою є однією з фундаментальних задач машинного навчання та комп'ютерного зору. Ця проблема також включає в себе багато менших підзадач, таких як класифікація об'єктів, локалізація об'єктів, сегментація зображення та ін.

Задача розпізнавання об'єктів бере початок ще в минулому столітті і широко використовується у сучасному світі. Ця проблема має багато сфер застосування.

У своїй роботі я розглядаю проблему розпізнавання об'єктів в контексті задачі видалення фону з растрового зображення в режимі реального часу. Проводиться аналіз вже існуючих рішень, з метою виявлення їх переваг і недоліків. Також проводиться огляд різноманітних методів і підходів до вирішення цієї задачі, які були популярними у різні часи. Після цього наводяться практичні приклади вирішення поставленої проблеми.

Результатом проведеного дослідження стане середовище, що буде складатися з мобільного додатку та серверної частини, функціонал якого дозволить зробити фотографію будь-якого предмету і отримати зображення без фону.

Розділ 1. Дослідження та аналіз готових рішень в даній області

Проблема швидкого і якісного видалення фону з зображення є дуже актуальною у багатьох сферах. Через це на сучасному ринку представлено багато варіантів її вирішення. Я провела аналіз існуючих рішень для оцінки їх переваг та недоліків. Для об'єктивності аналізу, мною було вибрано декілька однакових фотографій, з яких я видалятиму задній фон. Це фото людини (рис А.1), фото з чітко вираженим заднім планом (рис. А.2 - А.3), і фото з відсутніми заднім та переднім планами (рис А.4), для перевірки стійкості сервісів до некоректних даних.

Було обрано наступні популярні сервіси:

- Remove BG [1]
- Canva Pro [2]
- Clipping magic [3]
- Slazzer [4]

Також до аналізу я вирішила додати Adobe Photoshop [5] – інструмент «Виділення об'єкта» - що не є веб-застосунком, але є одним із провідних інструментів для проведення подібних маніпуляцій з зображеннями.

1.1 Remove BG

На рисунках 1.1 – 1.3 показано результат виконання застосунку. Як бачимо об'єкти на зображеннях чітко окреслений і майже не потребує подальшого редагування.

На рисунку 1.4 проілюстровано наскільки програма чутлива до «некоректних» даних. Як бачимо, Remove BG розпізнало, що на рис. А.4 не має переднього плану.

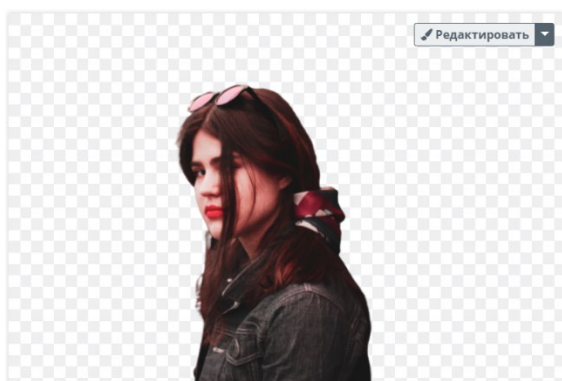


Рисунок 1.1 – Приклад видалення фону за допомогою Remove Bg

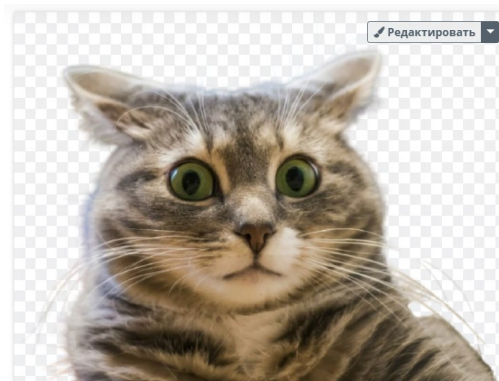


Рисунок 1.2 – Приклад видалення фону за допомогою Remove Bg



Рисунок 1.3 – Приклад видалення фону за допомогою Remove Bg

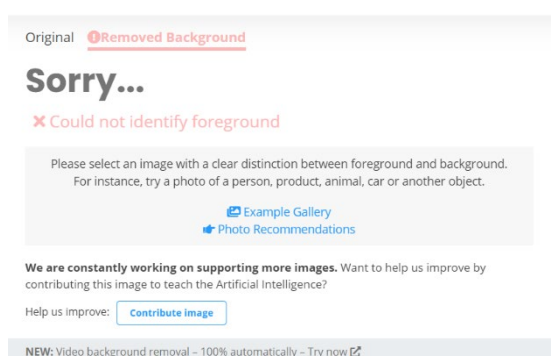


Рисунок 1.4 – Приклад видалення фону за допомогою Remove Bg

Серед переваг даного проєкту, окрім високої точності виконання поставленого завдання, також можна зазначити можливість інтеграції веб-сервісу в Adobe Photoshop та Figma [6]. Також Remove BG надає можливість підключатися до сервісу через API.

Із недоліків: відсутність функціональних інструментів редагування результату – присутнє лише стандартне видалення/додавання за допомогою пензлика.

Слід зазначити, що наразі Remove BG є одним з найпопулярніших веб-сервісів для видалення фону з зображень.

1.2 Canva PRO

Canva PRO – онлайн редактор, який має вбудовану функції видалення фону. Результати виконання задачі даним веб-застосунку зображені на рисунках

1.5 – 1.8. Canva PRO так само дуже добре справляється з поставленою задачею і якісно видаляє фон.



Рисунок 1.5 – Приклад видалення фону за допомогою Canva Pro



Рисунок 1.6 – Приклад видалення фону за допомогою Canva Pro



Рисунок 1.7 – Приклад видалення фону за допомогою Canva Pro



Рисунок 1.8 – Приклад видалення фону за допомогою Canva Pro

Результат виконання рисунку А.4 зображений на рисунку 1.8. Як бачимо, програма не намагалася розпізнати об'єкти там, де їх немає, але і не видала жодної помилки - просто залишила зображення без змін.

До переваг даного сервісу можна зарахувати високу якість виконання поставленої задачі (не зважаючи на те, що дана функція не є основним напрямком проекту), а також можливість подальшої роботи з зображенням безпосередньо в редакторі.

Недоліками є відсутність інтеграції з будь-якими іншими сервісами, а також наявність лише примітивного редагування пензликом (йде мова про редагування саме результату видалення фону).

1.3 Clipping magic

Усі результати видалення фону з використанням Clipping magic (з редагуванням та без) зображені на рисунках 1.9 – 1.14.

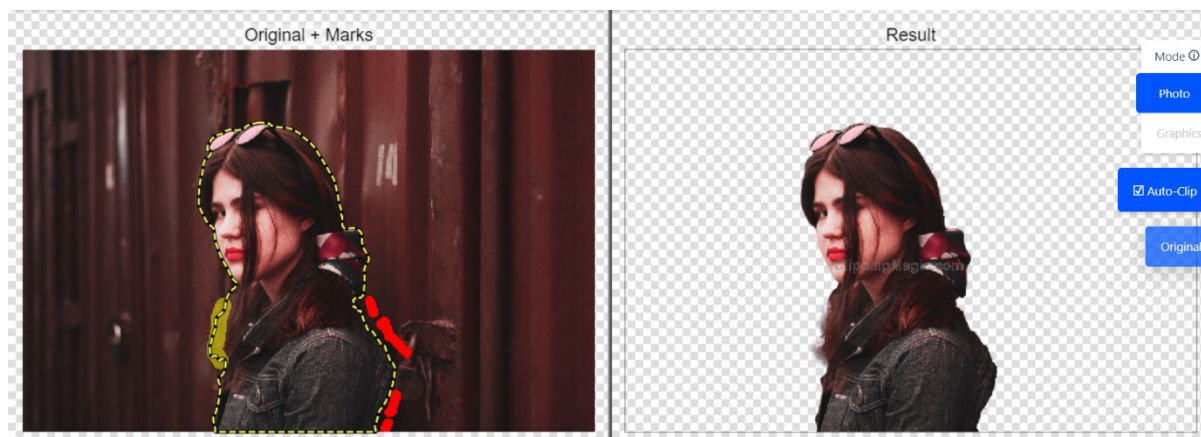


Рисунок 1.9 – Приклад видалення фону за допомогою Clipping magic

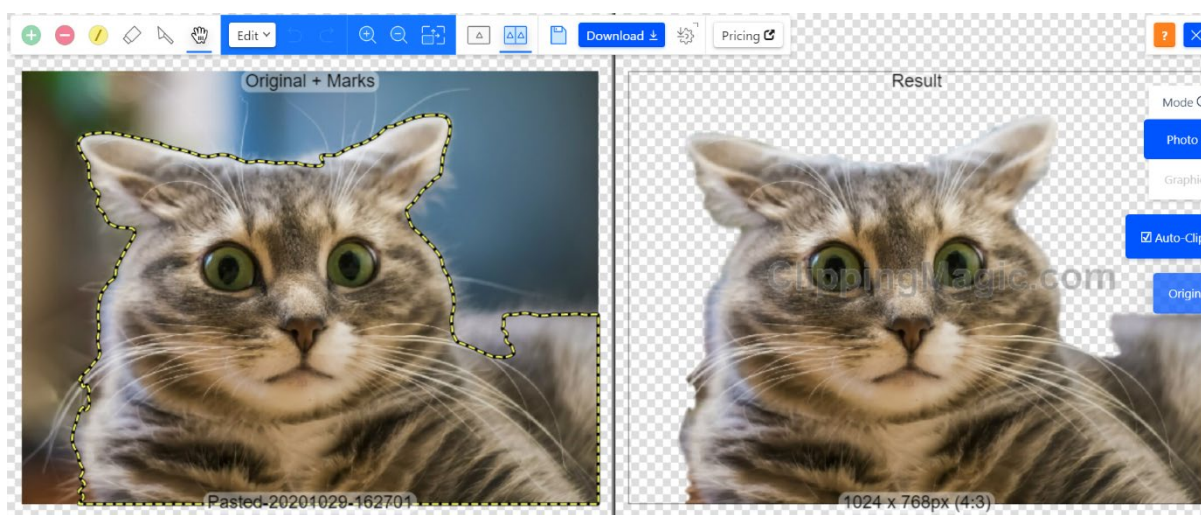


Рисунок 1.10 – Приклад видалення фону за допомогою Clipping magic

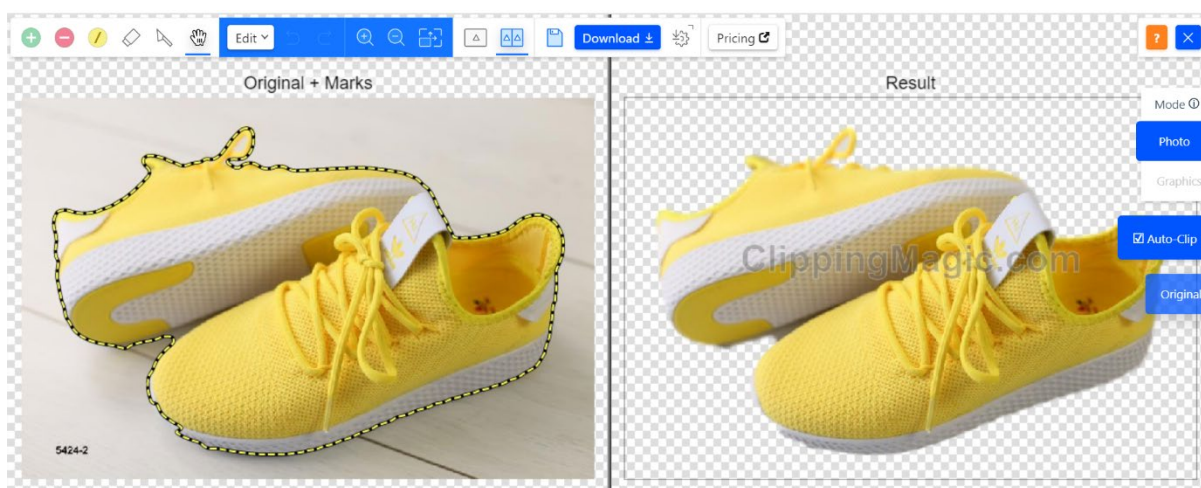


Рисунок 1.11 – Приклад видалення фону за допомогою Clipping magic

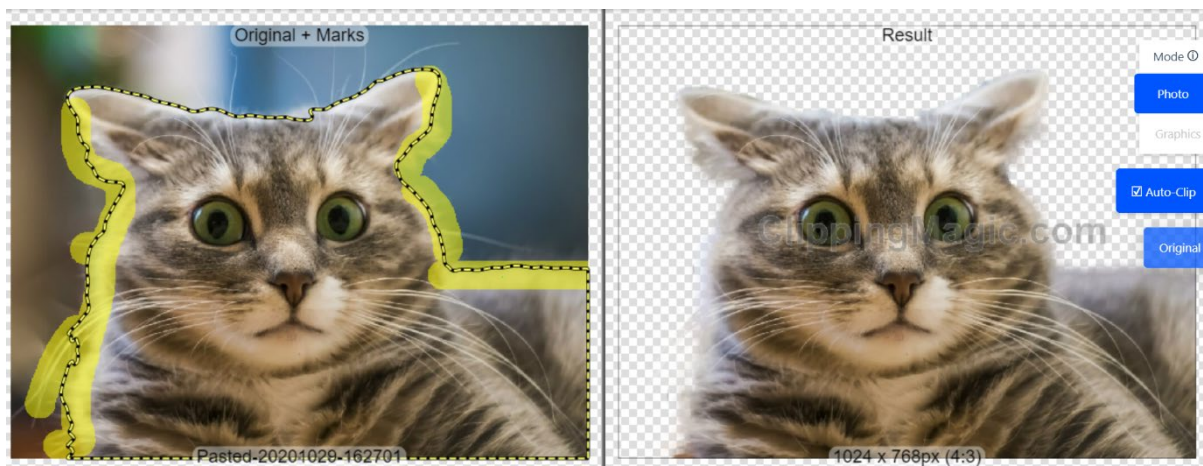


Рисунок 1.12 – Приклад видалення фону за допомогою Clipping magic



Рисунок 1.13 – Приклад видалення фону за допомогою Clipping magic

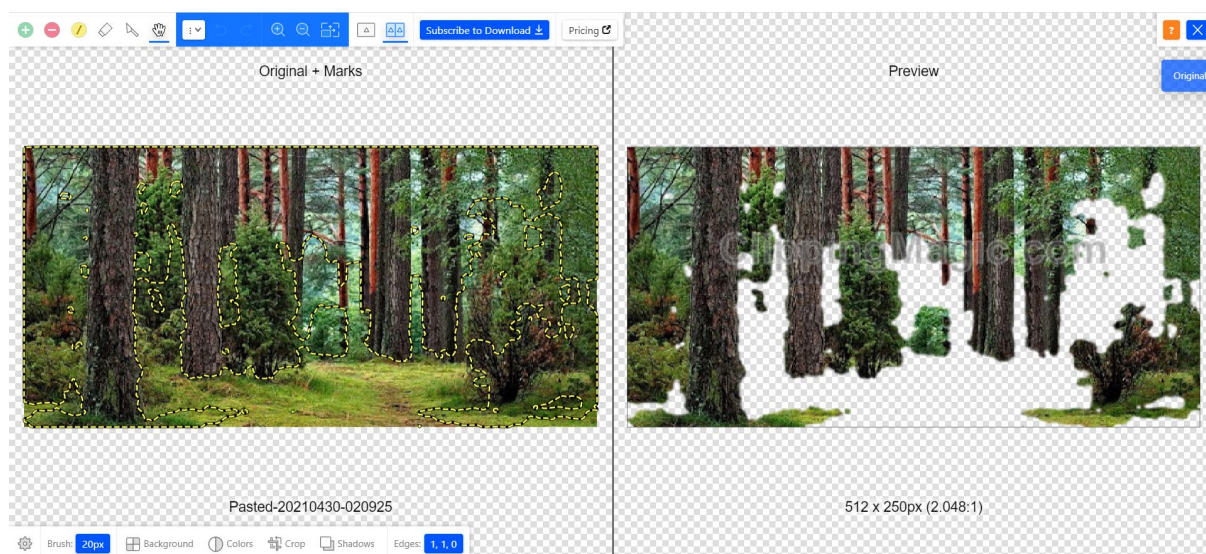


Рисунок 1.14 – Приклад видалення фону за допомогою Clipping magic

Результати даного сервісу не можна назвати успішними, оскільки якісно видалити фон без втручання людини у веб-застосунок не вийшло ні на жодному з зображень. Найкращий результат сервіс показав на рисунку А.2 (результат на рис. 1.10), де задній план розмитий і об'єкт чітко відокремлюється.

Окрему увагу хотілося б приділити рисунку 1.14, оскільки Clipping magic на відміну від попередніх застосунків, не зміг відрізнити некоректне фото і спробував розпізнати хоча б щось на зображенні.

Тим не менш, хорошою перевагою даного сервісу є зручний функціонал для редагування результатів видалення фону. Можна промаркувати зони, які стовідсотково будуть заднім планом і ті, які стовідсотково будуть переднім планом. Також окремо можна відмітити зони, на яких присутнє волосся (або інші елементи схожого матеріалу, що потребують більш детальнішого опрацювання). Після такого редагування, результати виконання стають набагато точнішими (рис. 1.9, 1.12, 1.13).

Тож, перевагою Clipping magic є хороший функціонал для редагування результатів.

Серед недоліків сервісу погані результати виконання програми без сторонньої допомоги (попереднього редагування), а також відсутність інтеграції з іншими сервісами.

1.4 Slazzer

На рисунках 1.15 – 1.18 зображені результати видалення фону сервісом Slazzer. Можна відмітити доволі високу точність даного веб-застосунку, проте не найкращу і не найшвидшу. Зокрема, програма не приділяє окремої уваги шерсті та волоссю, а просто згладжує усі краї.

Також, в Slazzer так само як і в Clipping magic не передбачена перевірка на достовірність вхідних даних. Як ми можемо побачити на рисунку 1.18, сервіс спробував видалити фон з зображення, де це зробити коректно не можливо.



Рисунок 1.15 – Приклад видалення фону за допомогою



Рисунок 1.17 – Приклад видалення фону за допомогою Slazzer



Рисунок 1.16 – Приклад видалення фону за



Рисунок 1.18 – Приклад видалення фону за допомогою Slazzer

Окремий момент, на який хочеться звернути увагу, це те що даний сервіс зміг повністю розпізнати тіло кота на рисунку 1.16, тоді як інші веб-застосунки частково, або повністю видаляли спину.

Отож, перевагами Slazzer є порівняно висока якість видалення фону.

З недоліків: відсутність валідування вхідних даних та довший період обробки зображення. Перевірити можливість редагування не вдалося, оскільки це потребує платної версії програми.

1.5 Adobe Photoshop – Інструмент «Виділення об'єкта»

Adobe Photoshop – потужний інструмент для редагування фотографій, який вже зарекомендував себе. Програма має декілька інструментів, які можуть виконати поставлену задачу, проте для цього аналізу я користувалася «Виділенням об'єкту» - відносно новим інструментом, який, як зазначено в інструкції, знаходить і автоматично виділяє об'єкт у зазначеній області.

Розглянемо результати роботи даного інструменту, зображені на рисунках 1.19 – 1.22.

Примітка: сам інструмент лише виділяє кордони об'єкту, на рисунках зображені результати отримані після накладання маски на виділену область.

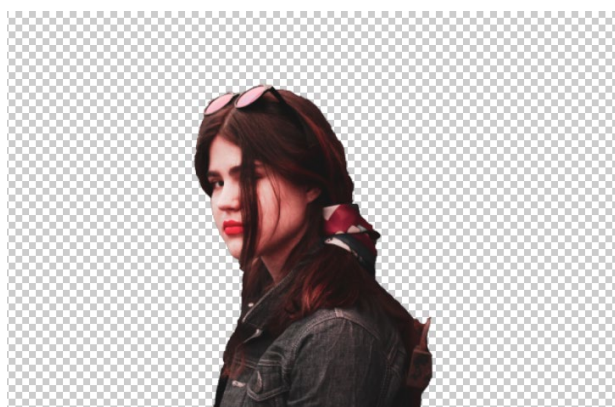


Рисунок 1.19 – Приклад видалення фону за допомогою Adobe Photoshop

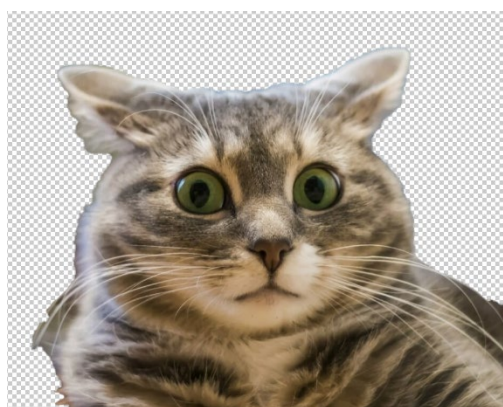


Рисунок 1.20 – Приклад видалення фону за допомогою Adobe Photoshop



Рисунок 1.21 – Приклад видалення фону за допомогою Adobe Photoshop

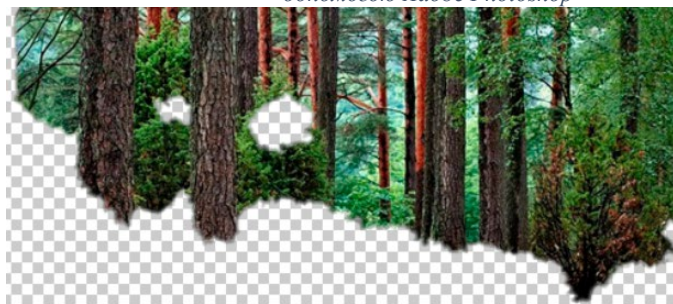


Рисунок 1.22 – Приклад видалення фону за допомогою Adobe Photoshop

Як бачимо, результати є далеко не найкращими. Проте видалення фону, як я згадувала раніше, є далеко не основною функцією Adobe Photoshop, крім того є багато інших інструментів, які виконують цю задачу (мною було

обраний лише той, який максимально схожий за іункціоналом з усіма іншими згаданими у цій роботі веб-сервісами). Вагомою перевагою даної програми є можливість подальшого редагування і роботи з отриманим зображенням.

Отже можна зробити висновок, що більше ніж половина з проаналізованих мною сервісів, не можуть з першого разу якісно видалити фон і результати потребують подальшого редагування.

Розділ 2. Дослідження різних методів та підходів до вирішення даної проблеми

2.1 Загальні підходи до вирішення проблеми

Розпізнавання об'єктів – одна з важливих задач машинного навчання та, зокрема, комп'ютерного бачення. Ця фундаментальна проблема включає в себе багато інших підзадач, таких як класифікація зображення, локалізація об'єктів, сегментація зображень і тп [7]. Розглянемо детальніше ці пвдзадачі.

Модель класифікації зображень на вході отримує фото (або відео), а на виході повідомляє до якого класу відноситься фрагмент.

Модель локалізації об'єктів розпізнає межі одиничного об'єкта на фрагменті. На виході ми отримуємо положення обмежувальної коробки, що окреслює об'єкт (корординати по осі x та осі y, довжину та ширину коробки).

На рисунку 2.1 наглядно пояснюється різниця між класифікацією, локалізацією та розпізнаванням об'єктів. В сучасному світі ці проблеми частіше вирішуються за допомогою глибокого навчання та згорткових

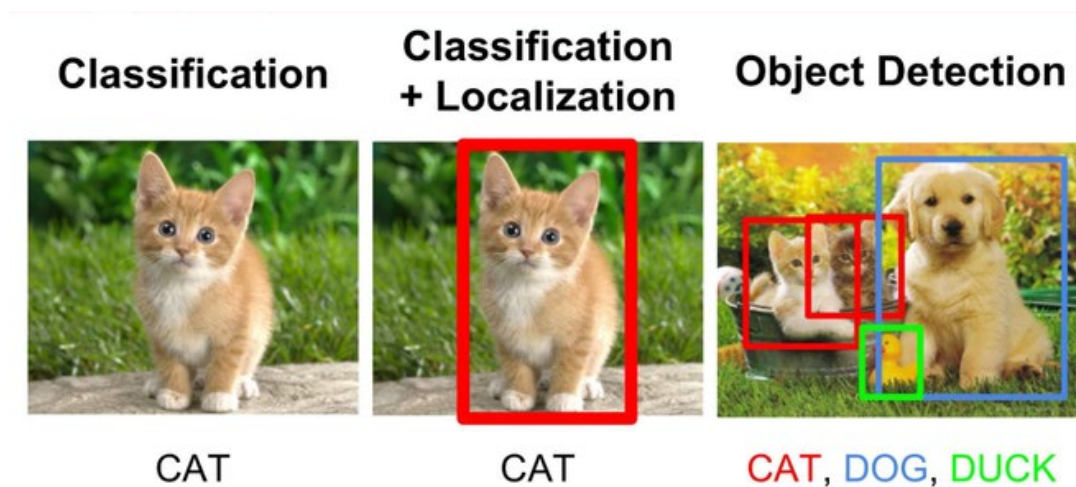


Рисунок 2.1. – Різниця між класифікацією зображень, класифікацією зображень та локалізацією об'єктів, виявленням об'єктів [30]

нейронних мереж (CNN). Такі моделі потребують багато часу для тренування і є доволі затратними по пам'яті та потужності. Ще одним недоліком класифікаторів є те, що модель може працювати лише в рамках описаних класів. Якщо надати зображення, дотепер невідоме нейронній мережі, результат буде абсолютно непередбачуваним.

Логічним продовженням даних підзадач, є задача сегментації об'єктів, яка включає в себе розпізнавання предмету та окреслення його кордонів. Це передбачає семантична сегментація.

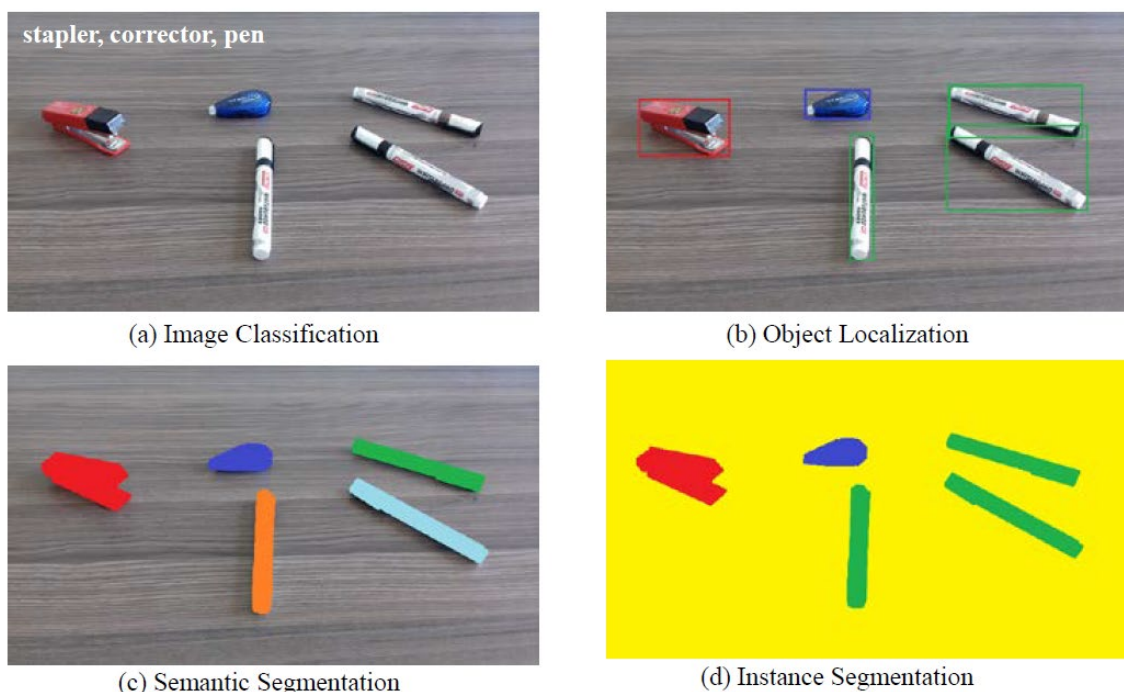


Рисунок 2.2 – Різниця між класифікацією зображень, локалізацією об'єктів, семантичною сегментацією та сегментацією зображень [4, fig.1]

Іншим поширеним варіантом є сегментація зображення, що розділяє кожен піксель вхідного зображення на певні класи і окреслює їх кордони (сегменти) [8]. Наглядну різницю між цими підходами можна побачити на рисунку 2.2. Сегментація не передбачає класифікацію, але вони цілком можуть застосовуватися разом в одній моделі.

Все це є теоретичними підходами до задачі розпізнавання і відокремлення об'єктів, які можуть комбінуватися, або ж виконуватися окремо для певних

потреб. Наразі існує багато різних алгоритмів та моделей машинного навчання, що практично виконують ці підзадачі.

2.2 Аналіз різних моделей розпізнавання об'єктів

Проблема розпізнавання об'єктів з'явилася більш ніж 20 років тому і за цей час було створено багато моделей та їх реалізацій для її вирішення [9]. У цьому розділі розглядатимуться різні історичні підходи, які були мною проаналізовані, а також будуть згадані потенційні моделі для подальшої практичної розробки автоматизованого застосунку для видалення фону.

2.2.1 Ознаки Хаара

Цей підхід з'явився за часів, коли ще не було нейронних мереж. В ті часи більшість алгоритмів створювалися на основі штучно створених ознак.

Ознака Хаара [10] складається з прямокутника розділеного на 2 частини (або більше). Цей прямокутник розміщується на зображенні, після чого сумується інтенсивність пікселів в кожній частині і знаходиться різниця між цими сумами. Це і є значенням даної ознаки на обраному розміщені.

Ознаки Хаара підходять для вирішення задачі розпізнавання образів (візерунків), аніж класичної задачі розпізнавання об'єктів. Для даного методу потрібно знати певні закономірності на шуканих об'єктах і на базі цього створювати маски з відповідними ознаками. Наприклад, якщо говорити про розпізнавання обличчя, то регіон в зоні очей буде темнішим, за регіон в зоні щік. Так само регіони очей, темніші за регіон носа. Таким чином, ці закономірності можна використати як ознаки Хаара (рис. 2.3).

Для розпізнавання, вікно заданого розміру рухається по зображенню (метод Віюли – Джонса [11]) і вираховує ознаки Хаара для кожної області. Якщо значення перевищило навчальний ліміт, то вважається, що шуканий

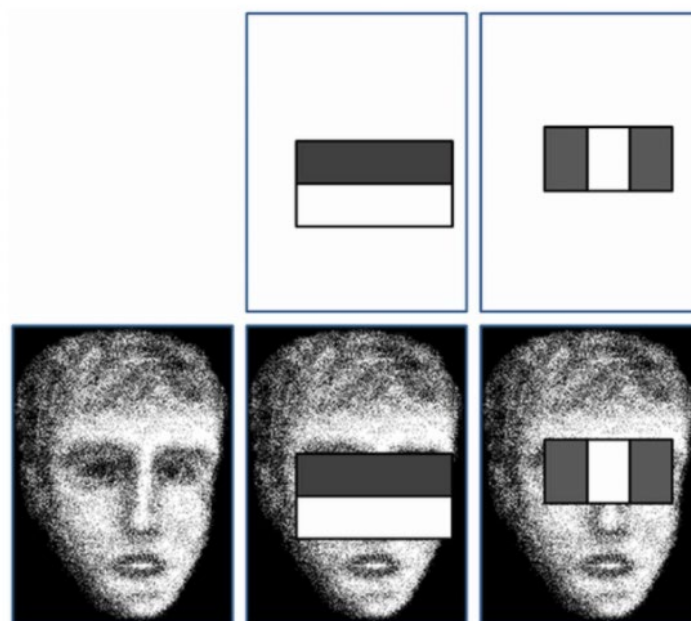


Рисунок 2.3 – Принцип роботи ознак Хаара [12, фіг.2]

предмет наявний у цьому вікні. Але одиночні ознаки дають результати не сильно вищі ніж у випадково розподіленої нормальної величини, тому для збільшення точності організовується каскадний класифікатор (набір декількох різних масок, які в свою чергу є набором декількох різних ознак Хаара).

Тим не менш, хоч цей метод здався мені доволі цікавим, - такий підхід, по-перше, є застарілим, і, по-друге, не сильно підходить для реалізації моєї задачі. Через ознаки Хаара можна було б спробувати відділяти кордони об'єктів, але такий підхід потребує дуже велику кількість слоїв каскадного класифікатору, оскільки всі об'єкти є дуже різної форми і не існує універсальної маски.

2.2.2 Гістограми орієнтованих градієнтів

Гістограми орієнтованих градієнтів [13] – це дескриптор ознак, який використовується для виявлення об'єктів. Метод вираховує усі випадки орієнтації градієнтів в локалізованих частинах зображення. Основна ідея методу полягає в тому, що місцезнаходження і форму об'єкта на фото

можна охарактеризувати розподілом локальних градієнтів інтенсивності або напрямків кордонів, навіть не знаючи точно положення відповідного градієнта або кордону. На практиці, зображення розділяється на невеликі комірки і для кожної вираховується гістограма градієнтного напрямку або орієнтації кордонів пікселів комірки. Після цього отриманий набір ознак можна використати в будь-якій існуючій моделі машинного навчання для класифікації [14].

Наглядний приклад роботи даного дескриптора можна побачити на рисунку 2.4.

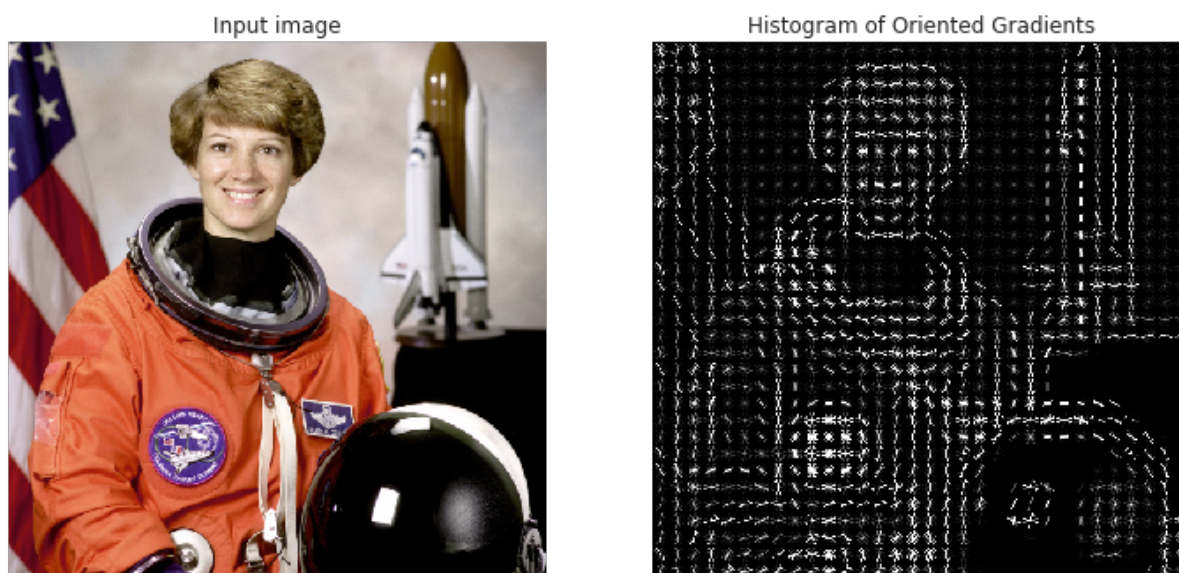


Рисунок 2.4 – Принцип роботи гістограм орієнтованих градієнтів [15]

Найпоширеніше використання гістограм орієнтованих градієнтів – розпізнавання конкретних класів. На цих ознаках можна натренувати будь-який класифікатор. Проте для вирішення проблеми видалення фону гістограми не підходять, оскільки не сильно справляються з проблемою сегментації, бо окреслюють усі наявні на зображенні кордони, а не лише кордони об'єктів. Як варіант, ці дескриптори можуть бути частини більшої моделі, яка спочатку класифікуватиме зображення, а потім робитиме сегментацію. Проте я не зустрічала подібних реалізацій серед уже

існуючих, що дає змогу припустити, що такий метод не даватиме хорошої точності, і з цим питанням набагато краще справляються інші моделі.

2.2.3 Оператор Соболя

Оператор Соболя [16] використовується в алгоритмах виявлення країв об'єктів. В результаті використання дескриптора можна отримати зображення, що чітко окреслює краї об'єктів. Основна ідея оператора полягає в згортанні зображення за допомогою невеликого, відокремлюваного і цілочисельного фільтра в горизонтальному та вертикальному напрямках.

Не заглиблюючись в теорію, результати роботи цього дескриптора можна побачити на рисунках 2.5 – 2.6.



Рисунок 2.5 – Оригінальне зображення [16]

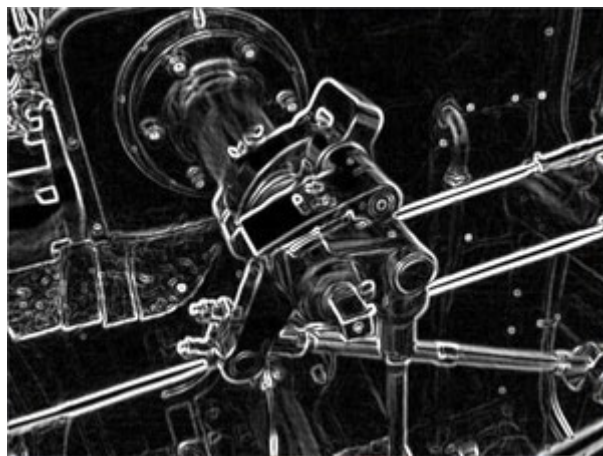


Рисунок 2.6 – Застосування оператора Соболя [16]

Я досліджувала існуючу модель для видалення фону [17], яка використовує саме оператор Соболя: проробивши додаткові операції над зображенням - наприклад, збільшення контрастності та видалення шумів – можна досягти кращих результатів окреслення кордонів; після цього можна створити маску, щоб все що за межами кордонів було заднім планом, а все, що всередині – переднім. Проте сам автор зазначає, що в цього підходу є певні недоліки: якщо видаляти фон у об'єктів з «дірками» всередині (наприклад, бублик), то внутрішня область не буде обрізана; часто тіні також

потраплятимуть в фінальний результат, оскільки їх кордони з поверхнею є дуже чіткими і оператор Соболя на них реагуватиме. Від себе додаю, що якщо фон не є відносно однотонним, а має в собі багато деталей (як на рисунку 2.5), то ці деталі також будуть розпізнані оператором, що робить таку модель не дієздатною саме в контексті відокремлення заднього плану.

Оператор Соболя є непоганим якісним дескриптором для визначення заднього та переднього плану, але в нього також є чимало недоліків. На момент аналізу даного підходу, я вирішила не відкидати його остаточно, але продовжила шукати більш точні моделі.

2.2.4 R-CNN, Mask R-CNN та інші CNN моделі

З появою нейронних мереж та глибоких нейронних мереж точність розпізнавання об'єктів підвищилась в середньому на 30% [9, фіг. 3].

R-CNN – модель для локалізації об'єктів - була запропонована, щоб обійти проблему вибору величезної кількості регіонів [18]. Система складається з 3-х модулів: перший генерує незалежні від категорій регіональні пропозиції (всього система має 2000 варіантів таких регіонів), другий – згорткова нейронна мережа, яка виділяє вектор ознак з кожного регіону, третій – набір SVM класифікаторів [19].

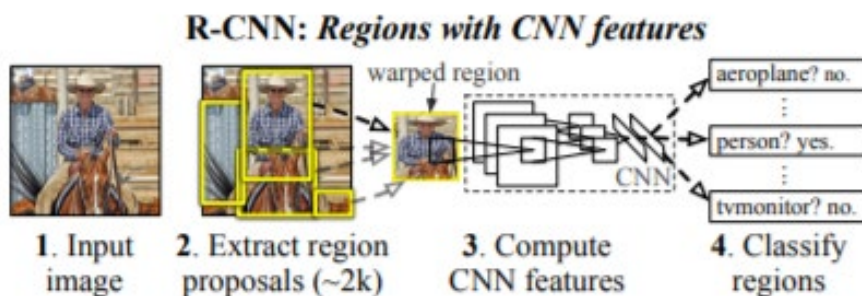


Рисунок 2.7 – Архітектура R-CNN [19, фіг.1]

Пізніше було розроблені моделі які пришвидшували локалізацію об'єктів та покращували точність, такі як Fast R-CNN, Faster R-CNN, YOLO.

Mask R-CNN розвиває архітектуру Faster R-CNN додаючи ще один шар, який вирішує проблему сегментації об'єкту. Сама маска це бінарна прямокутна матриця, для якої 1 означає належність пікселю до об'єкту, а 0 – неналежність [20].

Візуально виконання Mask R-CNN можна побачити на рисунку 2.8.

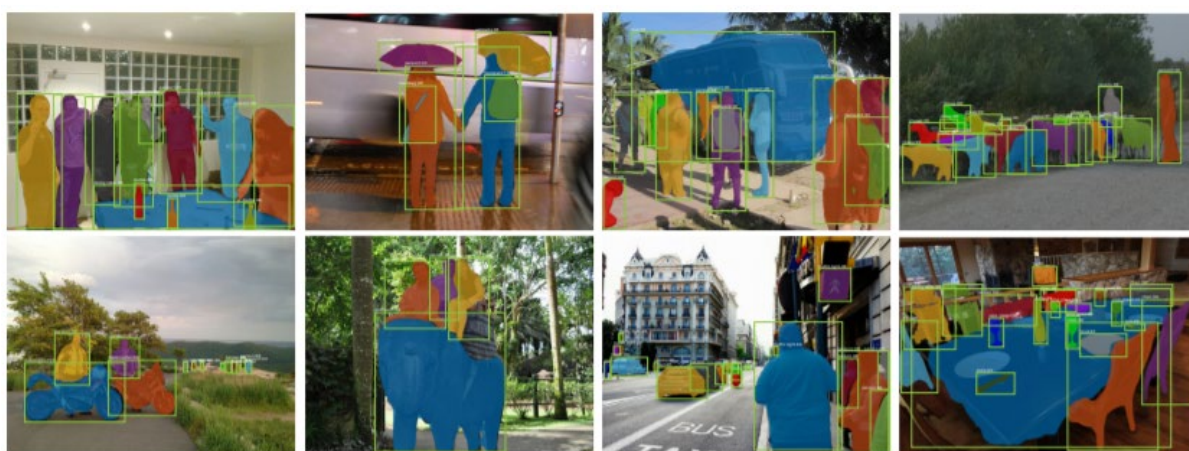


Рисунок 2.8 - Приклад результату виконання Mask R-CNN [20, фіг.2]

Ця модель дуже добре зарекомендувала себе в вирішенні проблеми розпізнавання об'єктів і є доволі популярною в наш час.

2.2.5 Silent Object Detection

Візуальна система людини здатна дуже точно сфокусуватися і вибірково обробляти лише найважливіші об'єкти. Silent Object Detection моделі має на меті визначити об'єкти, які є найбільш візуально привабливими [21], таким чином симулюючи вміння людей розпізнавати об'єкти.

Такі моделі розроблюються з використанням глибоких нейронних мереж. Існує багато різних архітектур для вирішення цієї проблеми, і наразі цей напрямок розпізнавання об'єктів дуже активно розвивається і особливо добре показують себе при роботі з відео рядом [22].

Ця модель сподобалася мені найбільше, оскільки вона є досить таки сучасною і постійно розроблюються нові архітектури, які роблять ще точніші передбачення, тому для практичної реалізації програми я вирішила використати якусь з Silent Object Detection моделей.

2.3 Бібліотека OpenCV

OpenCV [23] – бібліотека функцій та алгоритмів для вирішення проблем комп’ютерного бачення з відкритим кодом. Вона також включає в собі функції обробки зображень. Методами цієї бібліотеки дозволяє аналізувати вміст зображень і вирішувати такі проблеми, як розпізнавання об’єктів, відстеження руху об’єктів, застосування різноманітних перетворень та дескрипторів, застосування методів машинного навчання та ін., за допомогою більш ніж 2500 оптимізованих алгоритмів (як класичних, так і більш сучасних) [24].

OpenCV підтримує C++, Java та Python інтерфейси. Проте, найпоширенішим є все-таки Python, і більшість усіх наявних інструкцій доступні саме цією мовою. Щодо C++ та Java – всі методи описані в відповідних документаціях і є повністю дієздатними [25, 26].

Початково я спробувала реалізувати автоматизоване видалення фону користуючись методами цієї бібліотеки (див. Розділ 3.1), але пізніше зробила висновки, що OpenCV краще використовувати скоріше як допоміжний інструмент для виконання певних функцій, аніж повноцінну базу для проведення операцій видалення фону.

Розділ 3. Практичне застосування моделей машинного навчання для видалення фону

3.1 Видалення фону методами бібліотеки OpenCV

3.1.1 Теоретичні відомості

Під час дослідження практичних алгоритмів та методів видалення фону з зображення, я вирішила скористатися одним з методів бібліотеки Open CV – GrabCut.

Цей алгоритм був розроблений з ціллю задовільнити проблему відокремлення заднього плану від переднього з мінімальним втручанням користувача [27].

На вході алгоритм отримує зображення, а також обмежувальну коробку об'єкту. Таким чином, все, що знаходиться за межами коробки, за замовчуванням вважається заднім планом, а все, що всередині, - невідоме. Залежно від вхідних даних, комп'ютер робить початкове маркування пікселів. Після цього застосовується GMM (Модель Гаусівського змішування), що знову перемарковує пікселі, розділяючи їх на категорії: задній план, передній план, ймовірно задній план, ймовірно передній план. На основі цього розподілення будується граф (рис.3.1).

Вершини графу – пікселі, також додається дві вершини – задній план та об'єкт. Ребра мають свою вагу, яка залежить від схожості пікселів – якщо пікселі мають дуже різні кольори, то вага ребра між ними буде мінімальною. Також вага ребер, що зв'язують пікселі з вершинами заднього плану та об'єкту, виражається за вірогідність того, що цей піксель належить до заднього плану чи об'єкту. Після цього граф розділяється алгоритмом Mincut, таким чином, щоб сума розрізу була мінімальною [28].

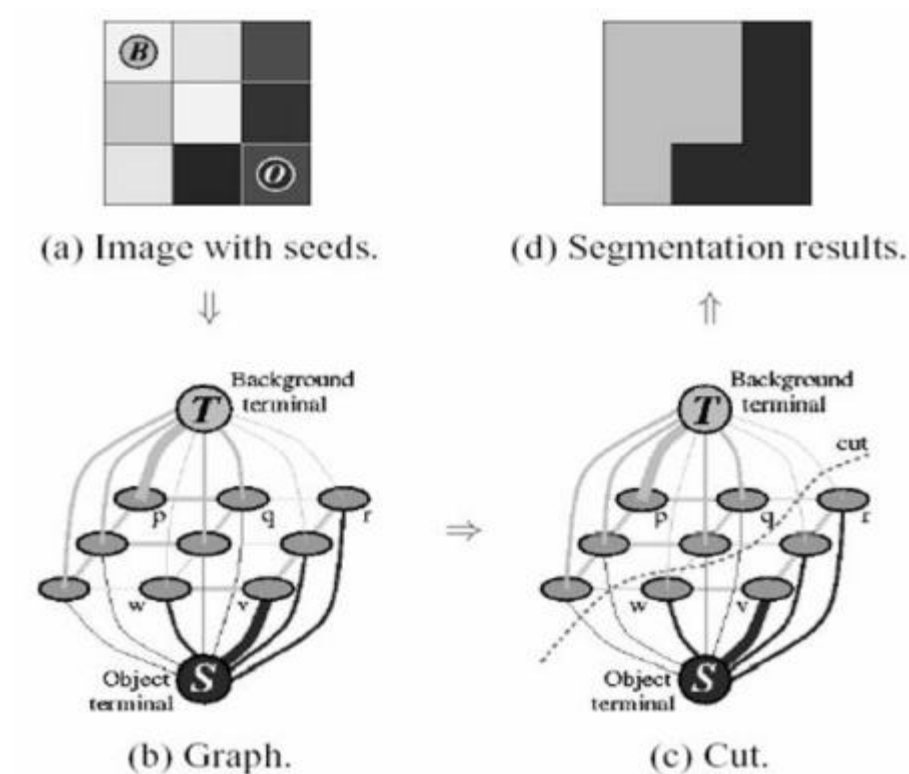


Рисунок 3.1 – Принцип роботи GrabCut [28, image 2]

3.1.2 Використання алгоритму GrabCut на Java

Для практичного застосування даного алгоритму, я користувалася інструкцією з електронного ресурсу [28], а також офіційною документацією OpenCV [25, 29] для Java.

Для роботи нам знадобиться імпортувати з бібліотеки OpenCV наступні її пакети:

```
import org.opencv.core.*;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
```

Алгоритм знаходиться в пакеті Imgproc, всі інші нам потрібні для роботи з зображенням.

Зчитуємо зображення як об'єкт класу `Mat`, що являє собою n -вимірний одноканальний або багатоканальний масив чисел. Тобто, пікселі нашого зображення представляються у вигляді елементів масиву [30].

```
// Завантаження зображення
Imgcodecs imageCodecs = new Imgcodecs();
String file = "src/myCode/img/" + name; // Локальна адреса зображення
Mat image = imageCodecs.imread(file);
```

Після цього треба задати зону (прямокутник), в якій знаходиться об'єкт. Важливо, що цей прямокутник не може займати все зображення - обов'язково мають бути пікселі, які маркуються як стовідсотковий задній план.

Я спробувала два варіанти: задавати обмежувальну коробку кожному зображенню окремо, та задавати коробку яка займає майже всю площу фотографії (відступ на 1 піксель по осі x).

```
// Границі об'єкта я знаходила вручну
Rect rectangle = new Rect(20, 50, 397, 467);

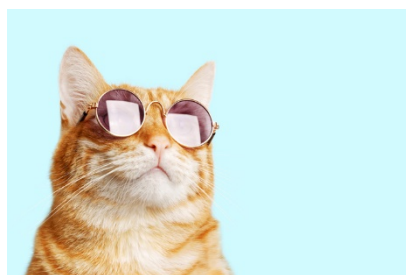
// Границі об'єкта займають майже всю площу фотографії
Rect rectangle = new Rect(1, 0, image.width(), image.height());
```

Після цього створюються допоміжні змінні та змінні для збереження результату і виконується сам алгоритм.

```
Mat result = Mat.zeros(image.size(), CvType.CV_8U); // Змінна для збереження результату
Mat bgdModel = Mat.zeros(1, 65, CvType.CV_64F);
Mat fgdModel = Mat.zeros(1, 65, CvType.CV_64F);

Imgproc.grabCut(image, result, rectangle, bgdModel, fgdModel, 2,
Imgproc.GC_INIT_WITH_RECT);
```

Результати виконання алгоритму можна побачити на рисунках 3.2 – 3.4



(a) Вхідне зображення

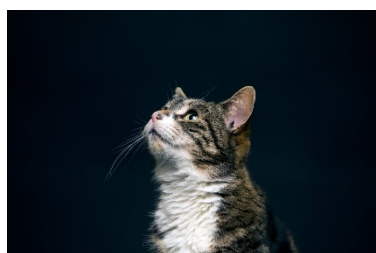


(b) Результати GrabCut з заданою обмежувальною коробкою



(c) Результати GrabCut на всій площі

Рисунок 3.2 - Результати виконання GrabCut



(a) Вхідне зображення

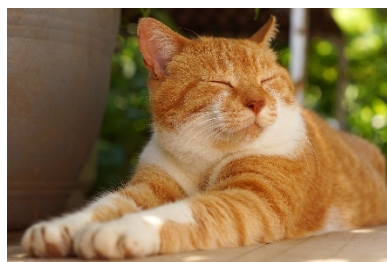


(b) Результати GrabCut з заданою обмежувальною коробкою



(c) Результати GrabCut на всій площі

Рисунок 3.3 - Результати виконання GrabCut



(a) Вхідне зображення



(b) Результати GrabCut на всій площі

Рисунок 3.4 – Результати виконання GrabCut

Як бачимо, даний алгоритм не дає ідеальний результат і потребує подальшого редагування. Було вирішено не користуватися даним методом для розробки застосунку, оскільки його результати не є задовільними.

З повним кодом програми можна ознайомитися в додатку Б.

3.2 Видалення фону використовуючи модель U²-Net

Дослідивши складніші моделі розпізнавання об'єктів з застосуванням методів глибокого навчання, я обрала для подальшої роботи архітектуру

U²-Net [21], що базується на моделі Silent Object Detection. Детальніше про цю архітектуру я розкажу в наступних підрозділах.

3.2.1 Архітектура додатку для видалення фону

Архітектуру програми можна розділити на наступні частини:

- Front-end (React Native, Expo);
- Rest API (Python, Flask);
- Модель для видалення фону (Python, U²-Net).

Таким чином додаток створений з використанням MVVM патерну.

3.2.1.1 View

Як вже було згадано вище, фронт-енд застосунку написаний на React Native та Expo, і першочергово орієнтований на ОС Android, але є можливість розширення до iOS. Тестування проводилось на Android-пристрої. Каркас додатку створювався на базі інструкції з електронного джерела [31].

Точкою входу в додаток є файл App.js, який в свою чергу викликає React.Component CameraPage:

```
import React from 'react';

import CameraPage from './src/camera.page';

export default class App extends React.Component {
  render() {
    return (
      <CameraPage />
    );
  }
};
```

У файлі CameraPage.js розписаний основна логіка програми. Він включає в себе React.Component CameraPage, що повертає наступне:

```

render() {
  const { hasCameraPermission, flashMode, cameraType, capturing, captures } = this.state;

  if (hasCameraPermission === null) {
    return <View />;
  } else if (hasCameraPermission === false) {
    return <Text>Access to camera has been denied.</Text>;
  }

  return (
    <React.Fragment>
      <View>
        <Camera
          type={cameraType}
          flashMode={flashMode}
          style={styles.preview}
          ref={camera => this.camera = camera}
        />
      </View>

      {captures.length > 0 && <Gallery captures={captures}/>}

      <Toolbar
        capturing={capturing}
        flashMode={flashMode}
        cameraType={cameraType}
        setFlashMode={this.setFlashMode}
        setCameraType={this.setCameraType}
        onShortCapture={this.handleShortCapture}
      />
    </React.Fragment>
  );
};

```

На рядках 5-9 відбувається перевірка, чи має додаток дозвіл до користування камери і повертає інтерфейс програми лише в позитивному випадку, інакше – відобразиться текст «Access to camera has been denied».

Компонент View інтерфейсу складається з Camera, Gallery та Toolbar. Camera - вже готовий компонент фреймворку Expo, імпортується з бібліотеки “expo-camera”. Gallery – це міні-галерея історії фотографій, яка

відображається лише у випадку, якщо ми вже робили попередні фото і масив `captures`, що містить ці фото, не пустий. `Toolbar` – це елементи керування інтерфейсом, а саме: кнопка «Зробити фото», кнопка «Ввімкнути/вимкнути спалах», кнопка «Перемкнути камеру». З кодом даних компонентів можна ознайомитися з додатках В і Г.

Обробка івенту натискання на кнопку «Зробити фото» виконує асинхронна функція `handleShortCapture()`, що викликає функцію бібліотеки `'expo-camera'` `takePictureAsync()`

```
handleShortCapture = async () => {
  const photoData = await this.camera.takePictureAsync({
    base64: true
  });
```

За замовчуванням ця функція повертає об'єкт `{uri, width, height}` – зі значеннями локальної адреси фотографії на пристрої, довжини та висоти. Також можна повернути `base64` (передавши в функцію об'єкт з ключем `base64` і значенням `true`) – бітовий код зображення.

Отримавши фотографію, програма формує з неї об'єкт класу `FormData()`:

```
let body = new FormData();
body.append('image', { uri: photoData.uri, name: 'picture.jpg', type:
'image/jpeg' });
```

Даний клас призначений для коректної пересилки файлів на сервер. Після цього робиться `POST`-запит до `API`. Файл передається в тілі запиту.

```
fetch('http://d61f109c4dc9.ngrok.io/post', {
  mode: 'no-cors',
  method: "POST",
  body: body
})
```

Оскільки функція `fetch()` є за замовчуванням асинхронною, відповідь ми будемо очікувати у `.then()` – функція чекатиме, доки запит повністю виконається і лише тоді продовжить роботу.

`.then()` приймає як параметри дві callback-функції – одна викликається в разі успішного виконання запиту, інші – в разі неуспішного.

```
.then(function (response) {

    if (response.ok) {

        response.json().then(async (json) => {
            const url = "http://d61f109c4dc9.ngrok.io" + json["file"];
            let filename = Date.now() + ".png";

            const fileUri = `${FileSystem.documentDirectory}${filename}`;

            const downloadedFile = await FileSystem.downloadAsync(url,
            fileUri);

            if (downloadedFile.status !== 200) {
                handleError();
            }

            MediaLibrary.createAssetAsync(downloadedFile.uri)
            .then(() => {
                console.log('Album created!');
                alert("Downloaded!");
            })
            .catch(error => {
                console.error('err', error);
            });

        })

    } else if (response.status == 401) {
        alert("Oops! ");
    }

},
function (e) {
    alert("Error submitting form! Error: " + e);
});
```

За успішного виконання ми перевіряємо статус відповіді (змінна `response`), і якщо сервер повернув статус 200 (те ж саме, що `response.ok === true`), то ми скачуємо файл з видаленим фоном з серверу (рядки 5-16) і зберігаємо його на пристрій (рядки 19-25). Для цих операцій ми використовуємо бібліотеки “expo-media-library” та “expo-file-system”, в яких вже прописаний відповідний функціонал.

Також паралельно цьому (оскільки запит є асинхронним) ми оновлюємо стан нашого додатку і додаємо зроблене фото в міні галерею (оновлюючи масив `captures`):

```
this.setState({ capturing: false, captures: [photoData, ...this.state.captures] })
```

Окрім описаного вище функціоналу, важливим елементом є функція `componentDidMount()`, що виконується як тільки компонент встановлюється програмою. В цій функції прописані запити доступу до камери та файлової системи. Якщо всі дозволи отримані, то значення `hasCameraPermission` встановлюється як `true`.

```
async componentDidMount() {
  const camera = await Permissions.askAsync(Permissions.CAMERA);
  const audio = await Permissions.askAsync(Permissions.MEDIA_LIBRARY);

  // const medialibrary = camera;
  const hasCameraPermission = (camera.status === 'granted' && audio.status === 'granted');

  this.setState({ hasCameraPermission });
};
```

3.2.1.2 ViewModel

Зв'язок між фронт-ендом і моделлю відбувається за допомогою Flask API.

Основна логіка відбувається в методі, що оброблює POST-запит з фронтенду.

```
@app.route('/post', methods=['Post'])
def webhook():
    file = request.files['image'];
    filename = str(time.time()) + '_' + file.filename
    file.save(os.path.join('uploads', filename))

    mask(filename)
    compress(filename + ".png")

    response = jsonify({
        "file": url_for('uploaded_file', filename=filename+".png")
    })
    print(response)
    return response
```

Даний метод дістає файл з тіла запиту і зберігає його у файлову систему. Після цього викликається імпортована з моделі функція `mask(filename)`, в параметри якої передається ім'я тільки що збереженого файлу. Модель в свою чергу оброблює файл і зберігає зображення з точно таким же ім'ям, але в іншій папці (і у форматі `png`). Далі викликається функція, що стискує зображення (`compress(filename + ".png")`) і зберігає в системі. Фронт-енду повертається JSON-об'єкт, в якому вказана адреса, за якою можна завантажити зображення.

Завантаження відбувається через GET-запит, який оброблюється наступною функцією:

```
@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory('test_data/compressed_images', filename)
```

3.2.1.3 Model

Для свого додатку я використала модель U²-Net [21], що була розроблена студентами Альбертського університету в Канаді. Дана модель розміщена у відкритому доступі на GitHub [32]. U²-Net порівняно нова архітектура, яку можна класифікувати як модель Silent Object Detection (SOD). Розробники прагнули вирішити 2 питання: по-перше, чи можна навчити SOD модель з нуля, без попереднього тренування на існуючих класифікаторах зображень (таких як AlexNet, ImageNet та ін.), і отримати аналогічні або кращі результати, ніж у моделей, які попередньо натреновуються на згаданих класифікаторах; і, по друге, чи можна зробити мережу глибше, ніж у попередників, при цьому залишивши таку ж високу якість карт ознак та низькі потреби в пам'яті та обчислювальній здатності машин, на якій буде використовуватися модель.

У результаті дослідження автори змогли досягти цих результатів, у чому можемо впевнитися, поглянувши на рисунок 3.5, на якому зображена порівняльна характеристика 76 різних метрик SOD – моделей, на трьох різних наборах даних. Червоним позначені найкращі результати, зеленим і синім – другі і треті найкращі результати відповідно.

Method	Backbone	Size(MB)	DUT-OMRON (5168)						DUTS-TE (5019)						HKU-IS (4447)					
			$maxF_{\beta}$	MAE	F_{β}^w	S_m	$relaxF_{\beta}^b$	$maxF_{\beta}$	MAE	F_{β}^w	S_m	$relaxF_{\beta}^b$	$maxF_{\beta}$	MAE	F_{β}^w	S_m	$relaxF_{\beta}^b$			
MDF _{TIP16}	AlexNet	112.1	0.694	0.142	0.565	0.721	0.406	0.729	0.099	0.543	0.723	0.447	0.860	0.129	0.564	0.810	0.594			
UCF _{ICCV17}	VGG-16	117.9	0.730	0.120	0.573	0.760	0.480	0.773	0.112	0.596	0.777	0.518	0.888	0.062	0.779	0.875	0.679			
Amulet _{ICCV17}	VGG-16	132.6	0.743	0.098	0.626	0.781	0.528	0.778	0.084	0.658	0.796	0.568	0.897	0.051	0.817	0.886	0.716			
NLDF _{+CVPR17}	VGG-16	428.0	0.753	0.080	0.634	0.770	0.514	0.813	0.065	0.710	0.805	0.591	0.902	0.048	0.838	0.879	0.694			
DSS _{+CVPR17}	VGG-16	237.0	0.781	0.063	0.697	0.790	0.559	0.825	0.056	0.755	0.812	0.606	0.916	0.040	0.867	0.878	0.706			
RAS _{ICCV18}	VGG-16	81.0	0.786	0.062	0.695	0.814	0.615	0.831	0.059	0.740	0.828	0.656	0.913	0.045	0.843	0.887	0.748			
PAGR _N _{CVPR18}	VGG-19	-	0.771	0.071	0.622	0.775	0.582	0.854	0.055	0.724	0.825	0.692	0.918	0.048	0.820	0.887	0.762			
BMPM _{CVPR18}	VGG-16	-	0.774	0.064	0.681	0.809	0.612	0.852	0.048	0.761	0.851	0.699	0.921	0.039	0.859	0.907	0.773			
PiCANet _{CVPR18}	VGG-16	153.3	0.794	0.068	0.691	0.826	0.643	0.851	0.054	0.747	0.851	0.704	0.921	0.042	0.847	0.906	0.784			
MLMS _{CVPR19}	VGG-16	263.0	0.774	0.064	0.681	0.809	0.612	0.852	0.048	0.761	0.851	0.699	0.921	0.039	0.859	0.907	0.773			
AFNet _{CVPR19}	VGG-16	143.0	0.797	0.057	0.717	0.826	0.635	0.862	0.046	0.785	0.855	0.714	0.923	0.036	0.869	0.905	0.772			
MSWS _{CVPR19}	Dense-169	48.6	0.718	0.109	0.527	0.756	0.362	0.767	0.908	0.586	0.749	0.376	0.856	0.084	0.685	0.818	0.438			
R ² Net _{+DCA118}	ResNeXt	215.0	0.795	0.063	0.728	0.817	0.599	0.828	0.058	0.763	0.817	0.601	0.915	0.036	0.877	0.895	0.740			
CapSal _{CVPR19}	ResNet-101	-	0.699	0.101	0.482	0.674	0.396	0.823	0.072	0.691	0.808	0.605	0.882	0.062	0.782	0.850	0.654			
SRM _{ICCV17}	ResNet-50	189.0	0.769	0.069	0.658	0.798	0.523	0.826	0.058	0.722	0.824	0.592	0.906	0.046	0.835	0.887	0.680			
DGRL _{CVPR18}	ResNet-50	646.1	0.779	0.063	0.697	0.810	0.584	0.834	0.051	0.760	0.836	0.656	0.913	0.037	0.865	0.897	0.744			
PiCANetR _{CVPR18}	ResNet-50	197.2	0.803	0.065	0.695	0.832	0.632	0.860	0.050	0.755	0.859	0.696	0.918	0.043	0.840	0.904	0.765			
CPD _{CVPR19}	ResNet-50	183.0	0.797	0.056	0.719	0.825	0.655	0.865	0.043	0.795	0.858	0.741	0.925	0.034	0.875	0.905	0.795			
PoolNet _{CVPR19}	ResNet-50	273.3	0.808	0.056	0.729	0.836	0.675	0.880	0.040	0.807	0.871	0.765	0.932	0.033	0.881	0.917	0.811			
BASNet _{CVPR19}	ResNet-34	348.5	0.805	0.056	0.751	0.836	0.694	0.860	0.047	0.803	0.853	0.758	0.928	0.032	0.889	0.909	0.807			
U ² -Net (Ours)	RSU	176.3	0.823	0.054	0.757	0.847	0.702	0.873	0.044	0.804	0.861	0.765	0.935	0.031	0.890	0.916	0.812			
U ² -Net [†] (Ours)	RSU	4.7	0.813	0.060	0.731	0.837	0.676	0.852	0.054	0.763	0.847	0.723	0.928	0.037	0.867	0.908	0.794			

Рисунок 3.5 – Порівняння сучасних SOD моделей [21, табл.4]

Враховуючи те, що ця модель давала дуже високі результати, а також є відкритою і активно оновлюваною – я вирішила використати саме її у своєму додатку.

Базово модель має такий функціонал: на вхід подається зображення (рис.3.6), а на виході отримується чорно-біла маска у форматі png (рис.3.7), що розділяє передній і задній плани.



Рисунок 3.6 [32] – Початкове зображення

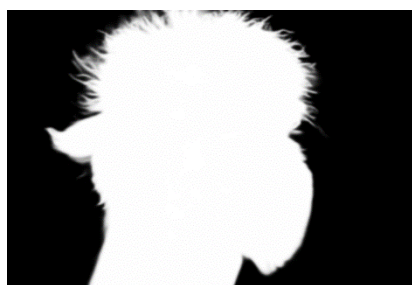


Рисунок 3.7 [32] – Результат виконання U^2 -Net

Поглянемо які результати дає модель на тестових зображеннях з додатку А (рис. 3.8 – 3.11).



Рисунок 3.8 – Результат видалення фону за допомогою U^2 -Net



Рисунок 3.9 – Результат видалення фону за допомогою U^2 -Net



Рисунок 3.10 – Результат видалення фону за допомогою U^2 -Net



Рисунок 3.11 – Результат видалення фону за допомогою U^2 -Net

Як бачимо, результати є дуже високими і конкурентоспроможними з існуючими реалізаціями, проаналізованими у Розділі 1.

До оригінального коду мною були дописані декілька функцій, для того щоб перетворити результат у потрібний мені вигляд. Перш за все, це функції, які накладають отримані маски на зображення і зберігають його з прозорим фоном замість заднього плану, також була додана функція для компресії зображень, і змінено вхідну функцію програми, щоб модель стала працювати з конкретним зображенням, а не набором даних для тестування, як у початковому коді.

Отже, тепер функція `main` приймає на вхід назву файлу, який має обробити (попередньо цей файл завантажує на сервер API).

Перші рядки вхідної функції наступні:

```
def main(name):  
  
    # ----- 1. get image path and name -----  
    model_name='u2net'#u2netp  
  
    image_dir = os.path.join(os.getcwd(), 'uploads')  
    prediction_dir = os.path.join(os.getcwd(), 'test_data', model_name + '_results' + os.sep)  
    model_dir = os.path.join(os.getcwd(), 'saved_models', model_name, model_name + '.pth')  
  
    img_name_list = glob.glob(image_dir + os.sep + name)  
    print(img_name_list)
```

Як бачимо, адреса зображення додається в список `img_name_list`. Оскільки ми працюємо з одним фото, то це завжди буде список з одного елементу. Результати виконання будуть зберігатися у папці `test_data/u2net_results`.

Після цього виконується обробка даних в U2Net і по завершенню викликається функція `save_output`, що відповідає за збереження результату. Як було показано вище, початковий код програми зберігає лише чорно білу маску. Тому в кінці функції `save_output` я викликаю дописану мною функцію `mask`, що накладає отриману маску на початкове зображення:

```
def mask(name):

    nms = name.split("\\")
    curimgpath = nms[len(nms)-1]
    print(curimgpath)
    img1 = cv2.imread('./uploads/'+curimgpath)
    print("IMG1\n")
    print(img1)

    nms2 = curimgpath.split("\\")
    destimgpath = nms2[len(nms2)-1].split(".")[0] + "." + nms2[len(nms2)-1].split(".")[1]+ ".png"

    print('PATH\n')
    print(destimgpath)

    img2 = cv2.imread('./test_data/u2net_results/'+ destimgpath)

    print("IMG2\n")
    print(img2)

    img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    transparent = img1.copy()
    transparent = cv2.cvtColor(transparent, cv2.COLOR_BGR2BGRA)
    transparent[:, :, 3] = img2

    cv2.imwrite('./test_data/masked_images/'+nms[len(nms)-1]+'+'.png', transparent)

    # De-allocate any associated memory usage
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

Результати зберігаються в окрему папку `test_data/masked_images`.

Після цього вже з API викликається функція `compress`, що зменшує розмір зображення і трохи знижує якість. На разі, параметри були підібрані опираючись на якість камери мого смартфона, але в майбутньому їх можна варіювати в залежності від початкової якості вхідного зображення.

```
def compress(filename):  
    img = Image.open("test_data/masked_images/"+filename)  
    print(f"The image size dimensions are: {img.size[0]} {img.size[1]}")  
  
    img = img.resize((int(img.size[0]/4),int(img.size[1]/4)),Image.ANTIALIAS)  
    img.save("test_data/compressed_images/"+filename,optimize=True,quality=30)
```

Зображення так само зберігається в окремій папці - `test_data/compressed_images`. API формує GET-запит саме з цієї папки.

Як результат, користувач отримує зображення без фону на свій смартфон. Обробка в середньому займає 10 секунд, але це також залежить від потужності сервера, на якому виконується програма (в моєму випадку це локальний сервер).

3.2.2 Приклади застосування

Розглянемо отриманий додаток у дії. На рисунках 3.12 – 3.14 зображений інтерфейс програми.

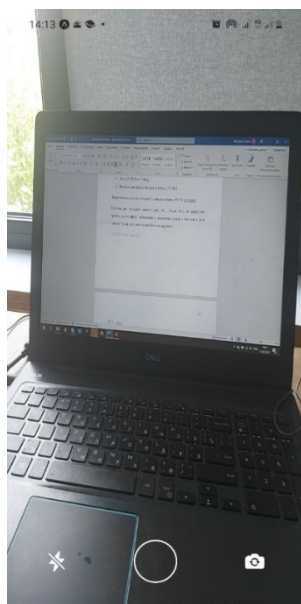


Рисунок 3.12 – Приклад роботи додатку. Основна камера



Рисунок 3.13 – Приклад роботи додатку. Фронтальна камера

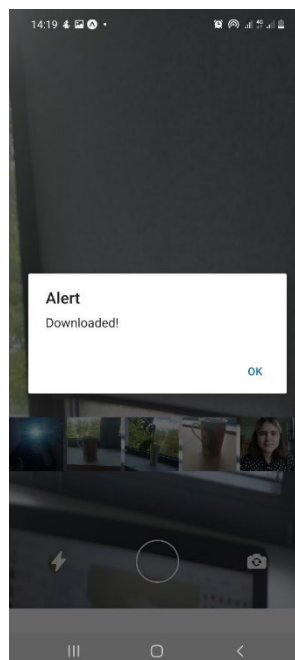


Рисунок 3.14 – Сповіщення про успішну обробку фото

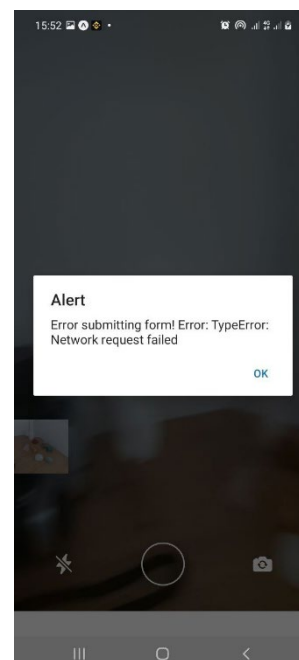


Рисунок 3.15 – Сповіщення про неуспішну обробку фото

Як ми бачимо, користувач має змогу увімкнути спалах, зробити фото та змінити орієнтацію камери. У разі успішного виконання обробки після фотографування, користувач отримує сповіщення (рис. 3.14). Якщо виникла якась помилка – сповіщення з помилкою (рис. 3.15).

Після натискання кнопки сфотографувати, показується невеличка піктограма фотографії в міні-галереї, як на рис. 3.16. Таким чином, користувач розуміє, що фото було зроблено успішно і треба почекати, аби відбулася обробка.

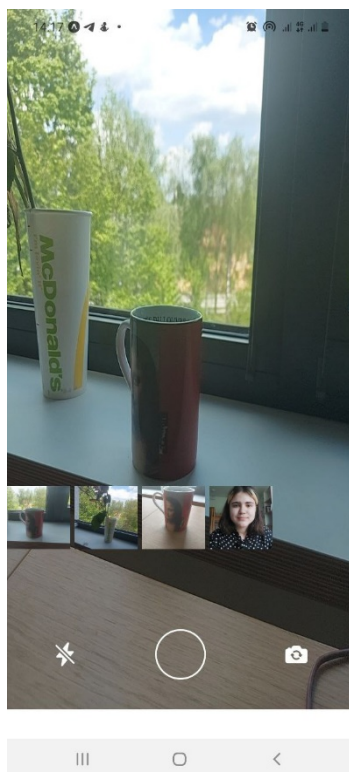


Рисунок 3.16 – Інтерфейс додатку

При активації кнопки спалаху, сам спалах включатиметься лише в момент фотографування (коли користувач натискає на кнопку «зробити фото»).

Поглянемо на результати роботи на рисунках 3.17 – 3.19.



Рисунок 3.17 – Результат видалення фону додатком



Рисунок 3.18 – Результат видалення фону додатком



Рисунок 3.19 – Результат видалення фону додатком

Як бачимо, отримані зображення мають порівняно високу точність і модель непогано справляється навіть з непрофесійними фото, зробленими на телефон.

Розроблений додаток добре справляється з поставленою задачею і виконує усі необхідні функції. Вагомим мінусом, на мою думку, є те, що зображення повертається з такими ж пропорціями, як і вхідне, і виходить, що на деяких фото забагато пустого місця. Хотілося б, щоб результат також обтинався до кордонів отриманого об'єкта. Проте ця проблема ніяк не впливає на функціональність на даному етапі і може бути вирішена у майбутньому, додавши у модель шар, який окреслює обмежувальну коробку об'єкта, перед тим як виконувати розпізнавання.

Висновки

В ході наукової роботи мною було визначено, що розробка автоматизованого застосунку для видалення фону з зображення є перш за все проблемою розпізнавання об'єктів. Проаналізувавши вже існуючі реалізації цієї задачі, я виділила їх плюси та мінуси, а також окреслила приблизну точність, яку має подолати моя програма. Для об'єктивності аналізу було обрано 4 різні фотографії: фото людини, 2 фото з чітко вираженим заднім фоном (розмитий фон та однотонний), фото без переднього плану, для перевірки стійкості моделей проти некоректних даних.

Мною також були проаналізовані теоретичні підходи та моделі вирішення цієї задачі в історичному розрізі, що допомогло краще зрозуміти суть проблеми, як вона розвивалася впродовж часу, і які рішення потенційно можуть бути використані на практиці. Я обрала декілька підходів і спробувала застосувати їх на практиці.

В результаті - я розробила мобільний додаток та серверну частину, які видаляють фон з зробленої в реальному часі фотографії. Обробка фотографії сервером відбувається в середньому від 5 до 10 секунд, а зображення повертається з прозорим фоном у форматі png, та зберігається у сховищі даних смартфона.

Список використаних джерел

1. Remove BG [Електронний ресурс] // Kaleido AI GmbH – Режим доступу до ресурсу: <https://www.remove.bg/>.
2. Canva [Електронний ресурс] // Canva Pty Ltd – Режим доступу до ресурсу: <https://www.canva.com/>.
3. Clipping Magic [Електронний ресурс] // Cedar Lake Ventures, Inc. – Режим доступу до ресурсу: <https://clippingmagic.com/>.
4. Slazzer [Електронний ресурс] // Slazzer – Режим доступу до ресурсу: <https://www.slazzer.com/>.
5. Adobe Photoshop [Електронний ресурс] // Adobe – Режим доступу до ресурсу: <https://www.adobe.com/ua/products/photoshop.html>.
6. Tools & API - remove.bg [Електронний ресурс] – Режим доступу до ресурсу: <https://vak.in.ua/>.
7. A Gentle Introduction to Object Recognition With Deep Learning [Електронний ресурс] // Jason Brownlee. – 2019. – Режим доступу до ресурсу: <https://machinelearningmastery.com/object-recognition-with-deep-learning/>.
8. A Survey on Instance Segmentation: State of the art [Електронний ресурс] // Abdul Mueed Hafiz, Ghulam Mohiuddin Bhat. – 2020. – Режим доступу до ресурсу: <https://arxiv.org/abs/2007.00047>.
9. A Brief History of Object Detection [Електронний ресурс] // Tommi Kerola. – 2019. – Режим доступу до ресурсу: <https://www.slideshare.net/pfi/a-brief-history-of-object-detection-tommi-kerola>.
10. Haar-like feature [Електронний ресурс] // Wikipedia, the free encyclopedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Haar-like_feature.

11. Viola–Jones object detection framework [Электронный ресурс] // Wikipedia, the free encyclopedia – Режим доступа до ресурсу:
https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework.
12. Face Recognition: A Tutorial on Computational Aspects [Электронный ресурс] // Alexander Alling, Nathaniel R Powers, Tolga Soyata. – 2016. – Режим доступа до ресурсу:
https://www.researchgate.net/publication/289252246_Face_Recognition_A_Tutorial_on_Computational_Aspects.
13. Histogram of oriented gradients [Электронный ресурс] // Wikipedia, the free encyclopedia – Режим доступа до ресурсу:
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.
14. Histograms of Oriented Gradients for Human Detection [Электронный ресурс] // Navneet Dalal and Bill Triggs. – 2005. – Режим доступа до ресурсу: <https://ieeexplore.ieee.org/abstract/document/1467360>.
15. Using Histogram of Oriented Gradients (HOG) for Object Detection [Электронный ресурс] – Режим доступа до ресурсу:
<https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/>.
16. Sobel operator [Электронный ресурс] // Wikipedia, the free encyclopedia – Режим доступа до ресурсу:
https://en.wikipedia.org/wiki/Sobel_operator.
17. Простой фильтр для автоматического удаления фона с изображений [Электронный ресурс]. – 2018. – Режим доступа до ресурсу:
<https://habr.com/ru/post/353890/>.
18. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [Электронный ресурс]. – 2018. – Режим доступа до

ресурсы: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.

19. Rich feature hierarchies for accurate object detection and semantic segmentation [Электронный ресурс] // 2014 – Режим доступа до ресурсу: <https://arxiv.org/pdf/1311.2524.pdf>.
20. Mask R-CNN [Электронный ресурс] // Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. – 2018. – Режим доступа до ресурсу: <https://arxiv.org/pdf/1703.06870.pdf>.
21. U²-Net: Going Deeper with Nested U-Structure for Salient Object [Электронный ресурс] // Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, Martin Jagersand. – 2020. – Режим доступа до ресурсу: <https://arxiv.org/abs/2005.09007>.
22. Salient Object Detection: A Benchmark [Электронный ресурс] // Ali Borji, Ming-Ming Cheng, Huaizu Jiang and Jia Li. – 2018. – Режим доступа до ресурсу: <https://arxiv.org/pdf/1501.02741.pdf>.
23. OpenCV [Электронный ресурс] // Wikipedia, the free encyclopedia – Режим доступа до ресурсу: <https://en.wikipedia.org/wiki/OpenCV>.
24. About - OpenCV [Электронный ресурс] – Режим доступа до ресурсу: <https://opencv.org/about/>.
25. OpenCV 4.5.2 Java documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.opencv.org/master/javadoc/index.html>.
26. OpenCV 4.5.2 C++ Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.opencv.org/4.5.2/>.
27. "GrabCut": Interactive Foreground Extraction using Iterated Graph Cuts [Электронный ресурс] // ACM Transactions on Graphics. – 2004. – Режим доступа до ресурсу: <https://doi.org/10.1145/1015706.1015720>.

28. Interactive Foreground Extraction using GrabCut Algorithm
[Електронний ресурс] – Режим доступу до ресурсу:
https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html.https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html.
29. OpenCV documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.opencv.org/java/3.0.0/>.
30. Mat Class Reference [Електронний ресурс] // OpenCV – Режим доступу до ресурсу:
https://docs.opencv.org/master/d3/d63/classcv_1_1Mat.html.
31. Building a Camera App with React Native [Електронний ресурс]. – 2019. – Режим доступу до ресурсу:
<https://www.codementor.io/@foysalit/building-a-camera-app-with-react-native-r8up5685v>.
32. U-2-Net [Електронний ресурс]. – 2021. – Режим доступу до ресурсу:
<https://github.com/xuebinqin/U-2-Net>.
33. Фото кота [Електронний ресурс] – Режим доступу до ресурсу:
https://golos.ua/images/items/2020-08/02/CxJ6myL6cfYB26Mn/img_top.jpg.
34. Фото кросівок [Електронний ресурс] – Режим доступу до ресурсу:
https://images.ua.prom.st/1796998689_w640_h640_zhenskie-letnie-krossovki.jpg.
35. Фото лісу [Електронний ресурс] – Режим доступу до ресурсу:
https://farm66.static.flickr.com/65535/50248613537_fde7faafe9_z.jpg.

Додаток А
(обов'язковий)

Зображення обрані для аналізу веб-застосунків для видалення фону



Рисунок А.1

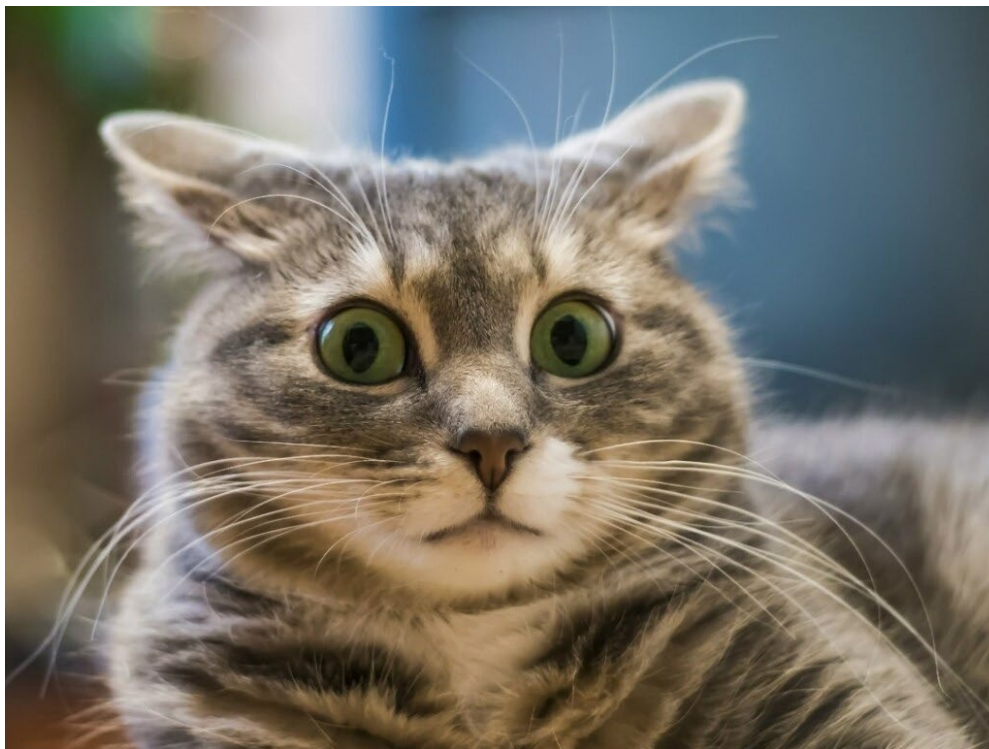


Рисунок А.2 [33]



Рисунок А.3 [34]



Рисунок А.4 [35]

Додаток Б

(ознайомчий)

Код програми для видалення фону методами бібліотеки OpenCV

```
package removeBG;

import java.awt.*;
import org.opencv.core.*;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;

public class Main {
    static{ System.loadLibrary(Core.NATIVE_LIBRARY_NAME); }

    public static void main(String[] args) {

        FileDialog dialog = new FileDialog((Frame)null, "Select File to
Open");
        dialog.setMode(FileDialog.LOAD);
        dialog.setVisible(true);
        String name = dialog.getFile();

        //Завантаження зображення
        Imgcodecs imageCodecs = new Imgcodecs();
        String file = "src/myCode/img/" + name;
        Mat image = imageCodecs.imread(file);

        //Границі об'єкта я вводила вручну
        Rect rectangle = new Rect(20, 50, 397, 467);

        //У другому варіанті границі займають зображення майже повністю:
        //Rect rectangle = new Rect(1, 0, image.width(), image.height());

        Mat result = Mat.zeros(image.size(), CvType.CV_8U);
        Mat bgdModel = Mat.zeros(1, 65, CvType.CV_64F);
        Mat fgdModel = Mat.zeros(1, 65, CvType.CV_64F);
        Imgproc.grabCut(image, result, rectangle, bgdModel, fgdModel, 2,
Imgproc.GC_INIT_WITH_RECT);

        Mat source = new Mat(1, 1, CvType.CV_8U, new Scalar(3));
        Core.compare(result, source, result, Core.CMP_EQ);
        Mat foreground = new Mat(image.size(), CvType.CV_8UC3, new
Scalar(255, 255, 255));
        image.copyTo(foreground, result);

        //Збереження зображення
        int index = 0;

        for(int i = name.length()-1; i >=0; i--)
            if (name.charAt(i) == '.') {
                index = i;
                break;
            }
        String newName = "";
```

```
for(int i = 0; i < name.length(); i++) {  
    if (i == index) newName += "_without_background2RECTANGLE";  
    newName += name.charAt(i);  
}  
  
String file2 = "src/myCode/img/" + newName;  
Imgcodecs.imwrite(file2, foreground);  
}  
}
```