Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(освітньо-кваліфікаційний рівень)
на тему

Розробка програмного забезпечення для видалення заднього фону на зображенні із портретами за допомогою машинного навчання

Виконав: студент 6 курсу, групи 602-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Кухар Р. С.
(прізвище та ініціали)
Керівник Альошин С.П.
(прізвище та ініціали)

Полтава – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122 «Комп'ютерні науки»

на тему

«Розробка програмного забезпечення для видалення заднього фону на зображенні із портретами за допомогою машинного навчання»

Студента групи 602-ТН Кухара Романа Степановича

Керівник роботи
кандидат технічних наук,
доцент Альошин С.П.

Завідувач кафедри
кандидат технічних наук,
доцент Двірна О.А.

Полтава – 2025

РЕФЕРАТ

Кваліфікаційна робота магістра: 86 с., 71 рисунок, 2 додатки, 28 джерела.

Об’єкт дослідження: процес видалення заднього фону з портретних зображень, виконаний за допомогою програмного забезпечення.

Мета роботи: дослідження і реалізація автоматизованого підходу до видалення фону, що дозволяє значно покращити якість обробки зображень і зробити цей процес доступним для широкого кола користувачів, а також розробка програмного забезпечення для автоматичного видалення заднього фону з портретних зображень.

Методи: аналіз існуючих рішень, машинне навчання, обробка зображень, виділення об’єктів, моніторинг використання ресурсів системи.

Ключові слова: МАШИННЕ НАВЧАННЯ, ЗОБРАЖЕННЯ, PYTHON.

ANNOTATION

Qualification work of master's degree: 86 p., 71 figures, 2 applications, 28 sources.

Object of study: the process of removing the background from portrait images, performed using software.

The goal of the work: research and implementation of an automated approach to background removal, which allows to significantly improve the quality of image processing and make this process accessible to a wide range of users, as well as development of software for automatic background removal from portrait images.

Methods: analysis of existing solutions, machine learning, image processing, object selection, monitoring of system resource usage.

Keywords: MACHINE LEARNING, IMAGES, PYTHON.

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

1. ПЗ – програмне забезпечення.
2. МЗ – машинне навчання.
3. ЗФ – задній фон.
4. CNN – Convolutional Neural Network (Згорткова нейронна мережа).
5. RGB – Red, Green, Blue (колірна модель).
6. ML – Machine Learning (Машинне навчання).
7. DL – Deep Learning (Глибоке навчання).
8. AI – Artificial Intelligence (Штучний інтелект).
9. ROI – Region of Interest (Область інтересу).
10. TF – TensorFlow (бібліотека для машинного навчання).
11. PyTorch – фреймворк для глибокого навчання.
12. IoU – Intersection over Union (метрика для оцінки якості сегментації).
13. МН – методи навчання (як варіант заміни ML).
14. DS – Dataset (набір даних).
15. FCN – Fully Convolutional Network (повна згорткова мережа).
16. U-Net – архітектура для сегментації зображень.
17. CV – Computer Vision (Комп'ютерний зір).
18. КК – контроль якості (для оцінки результатів).
19. ІО – Input/Output (введення/виведення).

ВСТУП

Обробка зображень відіграє ключову роль в багатьох сферах, від електронної комерції до соціальних медіа та маркетингу. Один з найважливіших аспектів обробки зображень полягає у видаленні заднього фону з портретних зображень, що дозволяє підготувати їх для подальшого використання в різних контекстах. Традиційні методи видалення фону часто є ручними, трудомісткими і не завжди забезпечують високу якість результату. Це створює потребу в автоматизованих рішеннях, які можуть спростити і прискорити цей процес.

З розвитком машинного навчання і комп'ютерного бачення з'явилася можливість автоматизувати процес видалення заднього фону з зображень. Ці технології дозволяють використовувати алгоритми, що навчаються на великих обсягах даних, для досягнення високої точності та ефективності в обробці зображень.

Метою цієї роботи є дослідження і реалізація автоматизованого підходу до видалення фону, що дозволяє значно покращити якість обробки зображень і зробити цей процес доступним для широкого кола користувачів, а також розробка програмного забезпечення для автоматичного видалення заднього фону з портретних зображень.

Об'єктом дослідження є процес видалення заднього фону з портретних зображень. Це включає як самі зображення, так і різні технічні аспекти їх обробки, такі як алгоритми сегментації, матування фону та побудова програмного забезпечення для автоматизації цього процесу.

Суб'єктом дослідження є методи та технології машинного навчання та комп'ютерного бачення, що використовуються для видалення заднього фону з зображень. Це охоплює розробку та реалізацію моделей згорткових нейронних мереж, архітектур нейронних мереж для сегментації зображень, а також реалізацію програмного забезпечення, яке інтегрує ці технології для автоматизації процесу видалення фону.

РОЗДІЛ 1
ТЕОРЕТИЧНА ЧАСТИНА

1.1. Основні поняття та визначення машинного навчання

Машинне навчання (Machine Learning, ML) – це галузь штучного інтелекту (ШІ) та комп'ютерних наук, яка фокусується на використанні даних та алгоритмів, що дозволяють ШІ імітувати спосіб навчання людини, поступово підвищуючи його точність.

Каліфорнійський університет в Берклі розбиває систему навчання алгоритму машинного навчання на три основні частини.

1. Процес прийняття рішень: загалом, алгоритми машинного навчання використовуються для прогнозування або класифікації. На основі деяких вхідних даних, які можуть бути маркованими або немаркованими, ваш алгоритм зробить оцінку щодо закономірності в даних.

2. Функція помилки: функція помилки оцінює передбачення моделі. Якщо є відомі приклади, функція помилки може зробити порівняння для оцінки точності моделі.

3. Процес оптимізації моделі: якщо модель може краще відповідати точкам даних у навчальному наборі, то ваги коригуються, щоб зменшити розбіжність між відомим прикладом і оцінкою моделі. Алгоритм повторює цей ітеративний процес «оцінки та оптимізації», оновлюючи ваги автономно, доки не буде досягнуто певного порогу точності [1].

Оскільки терміни «глибоке навчання» і «машинне навчання» часто вживаються як взаємозамінні, варто звернути увагу на нюанси між ними. Машинне навчання, глибоке навчання та нейронні мережі – це підгалузі штучного інтелекту. Однак нейронні мережі – це фактично підгалузь машинного навчання, а глибинне навчання – підгалузь нейронних мереж.

Різниця між глибинним і машинним навчанням полягає в тому, як кожен алгоритм навчається. «Глибоке» машинне навчання може використовувати марковані набори даних, також відомі як контрольоване навчання, для інформування свого алгоритму, але йому не обов'язково потрібен маркований набір даних. Процес глибокого навчання може поглинати неструктуровані дані в необробленому вигляді (наприклад, текст або зображення) і автоматично визначати набір ознак, які відрізняють різні категорії даних одна від одної. Це усуває частину необхідного втручання людини і дозволяє використовувати великі обсяги даних. Ви можете думати про глибоке навчання як про «масштабоване машинне навчання», як зазначає Лекс Фрідман у своїй лекції в Массачусетському технологічному інституті [2].

Класичне, або «неглибоке», машинне навчання більше залежить від втручання людини. Експерти-люди визначають набір ознак, щоб зрозуміти відмінності між вхідними даними, що зазвичай вимагає більш структурованих даних для навчання.

Нейронні мережі, або штучні нейронні мережі (ШНМ), складаються з шарів вузлів, що містять вхідний шар, один або кілька прихованих шарів і вихідний шар. Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку цей вузол не передає дані на наступний рівень мережі. Слово «глибокий» у глибокому навчанні означає лише кількість шарів нейронної мережі. Нейронна мережа, яка складається з більш ніж трьох шарів – включаючи вхід і вихід – може вважатися алгоритмом глибокого навчання або глибокою нейронною мережею. Нейронна мережа, яка має лише три шари, є просто базовою нейронною мережею.

Глибинному навчанню та нейронним мережам приписують прискорення прогресу в таких сферах, як комп'ютерний зір, обробка природної мови та розпізнавання мови.

Залежно від вашого бюджету, потреби у швидкості та точності, кожен тип алгоритму – контрольований, неконтрольований, напівконтрольований або посилений – має свої переваги та недоліки. Наприклад, алгоритми дерев рішень використовуються як для прогнозування числових значень (задачі регресії), так і для класифікації даних за категоріями. Дерева рішень використовують розгалужену послідовність пов'язаних між собою рішень, які можна представити у вигляді деревовидної діаграми. Основною перевагою дерев рішень є те, що їх легше перевіряти та перевіряти, ніж нейронні мережі. Погана новина полягає в тому, що вони можуть бути більш нестабільними, ніж інші предиктори рішень.

Загалом, машинне навчання має багато переваг, які компанії можуть використовувати для підвищення ефективності. До них відноситься те, що машинне навчання виявляє закономірності і тенденції у величезних обсягах даних, які людина може взагалі не помітити. І цей аналіз вимагає мінімального втручання людини: просто введіть набір даних, що вас цікавить, і дозвольте системі машинного навчання зібрати і вдосконалити свої власні алгоритми, які з часом будуть постійно вдосконалюватися з надходженням більшої кількості даних. Клієнти та користувачі можуть насолоджуватися більш персоналізованим досвідом, оскільки модель дізнається більше з кожним досвідом роботи з цією людиною.

З іншого боку, машинне навчання вимагає великих навчальних наборів даних, які є точними та неупередженими. GIGO – це оперативний фактор: сміття на вході / сміття на виході. Збір достатньої кількості даних і створення достатньо надійної системи для їх обробки також може призвести до виснаження ресурсів. Машинне навчання також може бути схильне до помилок, залежно від вхідних даних. На занадто малій вибірці система може створити абсолютно логічний алгоритм, який буде абсолютно неправильним або оманливим [3].

1.2. Базові моделі машинного навчання

Моделі машинного навчання поділяються на три основні категорії.

- контрольоване машинне навчання. Навчання під контролем, також відоме як контрольоване машинне навчання, визначається використанням маркованих наборів даних для навчання алгоритмів класифікації даних або точного прогнозування результатів. Коли вхідні дані подаються в модель, модель коригує свої ваги до тих пір, поки вони не будуть підігнані належним чином. Це відбувається в рамках процесу перехресної перевірки, щоб гарантувати, що модель уникає надмірного або недостатнього пристосування. Контрольоване навчання допомагає організаціям вирішувати різноманітні реальні проблеми в масштабі, наприклад, класифікувати спам в окремій папці від вашої поштової скриньки. Деякі методи, що використовуються в керованому навчанні, включають нейронні мережі, наївний байєс, лінійну регресію, логістичну регресію, випадковий ліс і машину опорних векторів (SVM);
- неконтрольоване машинне навчання. Некероване навчання, також відоме як некероване машинне навчання, використовує алгоритми машинного навчання для аналізу та кластеризації немаркованих наборів даних (підмножин, які називаються кластерами). Ці алгоритми виявляють приховані закономірності або групи даних без необхідності втручання людини. Здатність цього методу виявляти подібності та відмінності в інформації робить його ідеальним для дослідницького аналізу даних, стратегій перехресних продажів, сегментації клієнтів, а також розпізнавання зображень і шаблонів. Він також використовується для зменшення кількості ознак у моделі за допомогою процесу зменшення розмірності. Аналіз головних компонент (PCA) і декомпозиція сингулярних значень (SVD) є двома поширеними підходами для цього. Інші алгоритми, що використовуються в неконтрольованому навчанні, включають нейронні мережі, кластеризацію за методом k-середніх та імовірнісні методи кластеризації;

- напівкероване навчання. Напівкероване навчання пропонує золоту середину між керованим і некерованим навчанням. Під час навчання він використовує менший маркований набір даних, щоб керувати класифікацією та вилученням ознак з більшого немаркованого набору даних. Напівкероване навчання може вирішити проблему недостатньої кількості мічених даних для алгоритму керованого навчання. Воно також допомагає, якщо маркування достатньої кількості даних є надто дорогим;
- машинне навчання з підкріпленням – це модель машинного навчання, яка схожа на навчання під наглядом, але алгоритм не тренується на вибіркових даних. Ця модель навчається в процесі роботи, використовуючи метод спроб і помилок. Послідовність успішних результатів буде підкріплена, щоб розробити найкращу рекомендацію або політику для даної проблеми [4].

Хорошим прикладом є система IBM Watson, яка перемогла в конкурсі Jeopardy! у 2011 році. Система використовувала навчання з підкріпленням, щоб дізнатися, коли слід спробувати дати відповідь (або запитання), яку клітину на дошці вибрати і скільки ставити – особливо на щоденні подвоєння [5].

1.3. Приклади застосування машинного навчання

Лише кілька прикладів машинного навчання, з якими стикаються користувачі та розробники програмного забезпечення кожного дня:

- розпізнавання мови: це також відоме як автоматичне розпізнавання мови (asr), комп'ютерне розпізнавання мови або перетворення мови в текст. Це функція, яка використовує обробку природної мови (nlp) для перекладу людської мови в письмовий формат. Багато мобільних пристроїв включають розпізнавання мовлення у свої системи для здійснення голосового пошуку, наприклад, siri, або для покращення доступності текстових повідомлень;

- обслуговування клієнтів: онлайн-боти замінюють людей на всіх етапах взаємодії з клієнтом, змінюючи наше уявлення про взаємодію з клієнтами на веб-сайтах і платформах соціальних мереж. Чат-боти відповідають на поширені запитання (faq) на такі теми, як доставка, або надають персоналізовані поради, перехресні продажі товарів чи підказують розміри для користувачів. Приклади включають віртуальних агентів на сайтах електронної комерції; ботів для обміну повідомленнями, що використовують slack і facebook messenger; і завдання, які зазвичай виконують віртуальні асистенти і голосові помічники;
- комп'ютерний зір: ця технологія ші дозволяє комп'ютерам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних даних, а потім виконувати відповідні дії. Комп'ютерний зір, що працює на основі згорткових нейронних мереж, застосовується у тегуванні фотографій у соціальних мережах, радіологічній візуалізації в охороні здоров'я та безпілотних автомобілях в автомобілебудуванні;
- роботизована автоматизація процесів (rpa): також відома як програмна робототехніка, rpa використовує інтелектуальні технології автоматизації для виконання повторюваних ручних завдань.
- автоматизована торгівля акціями: розроблені для оптимізації портфелів акцій, високочастотні торгові платформи, керовані ші, укладають тисячі або навіть мільйони угод на день без втручання людини;
- виявлення шахрайства: банки та інші фінансові установи можуть використовувати машинне навчання для виявлення підозрілих транзакцій. Контрольоване навчання може навчати модель, використовуючи інформацію про відомі шахрайські транзакції. Виявлення аномалій може ідентифікувати транзакції, які виглядають нетипово і заслуговують на подальше розслідування [6].

1.4. Математичні основи машинного навчання

Математичні основи машинного навчання (ML) охоплюють різні розділи математики, які забезпечують теоретичну основу для алгоритмів та моделей. Ось основні математичні концепції, які використовуються в машинному навчанні:

1. Лінійна алгебра:

- вектори і матриці: багато алгоритмів ml використовують вектори та матриці для представлення даних та операцій з ними;
- власні значення і власні вектори: використовуються в методах зменшення розмірності, таких як pca (principal component analysis);
- сингулярний розклад матриці (svd): важливий для методів, пов'язаних з аналізом даних та компресією.

2. Математичний аналіз:

- диференціальне та інтегральне числення: використовується для оптимізації моделей, наприклад, для знаходження мінімумів функцій втрат;
- градієнти та градієнтний спуск: основні методи для навчання моделей.

3. Ймовірність та статистика:

- ймовірнісні розподіли: для моделювання та оцінки невизначеності в даних;
- байєсовий висновок: для оновлення ймовірностей на основі нових даних;
- гіпотезне тестування та довірчі інтервали: для оцінки значимості результатів моделей.

4. Оптимізація:

- оптимізаційні алгоритми: методи, такі як градієнтний спуск, використовуються для мінімізації або максимізації функцій;
- умовна оптимізація: використовується в задачах з обмеженнями, наприклад, lagrangian multipliers;

5. Дискретна математика:

- комбінаторика: використовується для аналізу кількості можливих моделей або шляхів;
- графи та мережі: для представлення та аналізу даних у вигляді графів, як, наприклад, в алгоритмах графового навчання.

Лінійна алгебра є фундаментальною частиною математичного апарату, який використовується в машинному навчанні. Вона дозволяє працювати з багатовимірними даними, що є надзвичайно важливим для аналізу та обробки великих обсягів інформації [7].

Основні поняття лінійної алгебри:

1. Вектори: вектор – це об'єкт, який має як величину, так і напрям. У машинному навчанні вектори використовуються для представлення даних точок в багатовимірному просторі. Векторні операції: включають додавання векторів, множення на скаляр, скалярний добуток (dot product), який використовується для вимірювання подібності між векторами.

2. Матриці: матриця – це двовимірний масив чисел, який використовується для представлення лінійних перетворень та систем рівнянь. Операції з матрицями: включають додавання, віднімання, множення матриць, транспонування та інверсію.

3. Лінійні перетворення: лінійне перетворення – це функція, яка перетворює один вектор в інший, зберігаючи операції додавання та множення на скаляр. Матриці лінійних перетворень використовуються для представлення лінійних перетворень. Наприклад, обертання, масштабування та відображення можна представити за допомогою матриць.

4. Сингулярний розклад матриці – це факторизація матриці на три матриці: одна діагональна і дві ортогональні. Використовується для зменшення розмірності даних та виявлення основних структур.

5. Нормалізація та ортогоналізація. Нормалізація: процес перетворення вектора до одиничної довжини. Використовується для забезпечення

стабільності числових алгоритмів. Ортогоналізація: процес перетворення набору векторів на ортогональні вектори, що є взаємно перпендикулярними.

Застосування лінійної алгебри в машинному навчанні включає в себе такі пункти, без яких було б неможливе застосування машинного навчання:

1. Представлення даних: матриці часто використовуються для зберігання та обробки наборів даних. Наприклад, вектор ознак, який представляє дану точку, може бути представленим як рядок або стовпець матриці.

2. Регресія та класифікація: лінійна регресія – це алгоритм, який намагається знайти лінійну залежність між незалежними та залежними змінними. Логістична регресія – використовується для класифікації, де прогнозується ймовірність належності до певного класу.

3. Зменшення розмірності: методи використовують лінійну алгебру для виявлення головних компонент у високорозмірних даних, що дозволяє зменшити розмірність без втрати значущої інформації [8].

Математичний аналіз є ключовою частиною теоретичного апарату машинного навчання. Він охоплює вивчення змін та їх поведінки, що є основою для оптимізації алгоритмів та розуміння складних моделей.

Основні поняття математичного аналізу:

1. Диференціальне числення: похідні визначають швидкість зміни функції відносно зміни незалежної змінної. У машинному навчанні похідні використовуються для мінімізації функцій втрат. Часткові похідні використовуються, коли функція залежить від кількох змінних. Вони показують, як зміна однієї змінної впливає на функцію, залишаючи інші незмінними. Градієнт – вектор, що складається з часткових похідних, вказує напрямок найбільшого зростання функції. У машинному навчанні градієнт використовується в алгоритмах оптимізації, таких як градієнтний спуск.

2. Інтегральне числення – використовуються для знаходження площ під кривими та визначення накопичених змін. У машинному навчанні інтеграли

можуть використовуватися для обчислення очікувань та інших агрегатних функцій.

3. Оптимізація: мінімізація та максимізація. Багато задач машинного навчання полягають у мінімізації функції втрат або максимізації функції правдоподібності [9].

Застосування математичного аналізу в машинному навчанні:

1. Навчання моделей. Функція втрат – кількісна міра помилки моделі. Математичний аналіз використовується для мінімізації цієї функції, що дозволяє моделі краще відповідати даним. Градієнтний спуск – метод оптимізації, який використовує градієнт функції втрат для поступового покращення моделі. Існує кілька варіацій, включаючи стохастичний градієнтний спуск та адаптивні методи, такі як adam.

2. Регуляризація: l1 та l2 регуляризація – додають до функції втрат додаткові члени, що запобігають перенавчанню моделі, штрафуючи великі ваги.

3. Аналіз збіжності забезпечують теоретичні підстави для гарантій збіжності алгоритмів навчання. Вони використовують поняття ліміту функцій та рядів.

4. Обробка сигналів та зображень: фур'є та вейвлет-перетворення – використовуються для аналізу частотного вмісту сигналів та зображень. Вони дозволяють виділяти важливі характеристики для подальшого аналізу та обробки.

Ймовірність та статистика є основними галузями математики, які широко використовуються в машинному навчанні для моделювання невизначеності, аналізу даних та розробки алгоритмів. Вони забезпечують теоретичну основу для розуміння та побудови моделей, що працюють з випадковими явищами. Крім того, вони є невід'ємною частиною машинного навчання, надаючи необхідні інструменти для роботи з невизначеністю, моделювання даних та оцінки моделей. Вони дозволяють розробляти надійні

та ефективні алгоритми, які можуть працювати з реальними даними та робити точні прогнози.

Оптимізація є центральною частиною машинного навчання, оскільки вона дозволяє знайти найкращі параметри моделі для мінімізації або максимізації певної цільової функції.

Основні поняття оптимізації:

1. Цільова функція (функція втрат): визначає, наскільки добре модель прогнозує або класифікує дані. Вона вимірює різницю між прогнозованими та фактичними значеннями. Функція корисності: використовується в задачах максимізації, де цільовою є функція, яку потрібно максимізувати.

2. Градієнти та похідні. Градієнт: вектор часткових похідних цільової функції за всіма параметрами моделі. Він вказує напрямок найкрутішого зростання функції. Гессіан: матриця других похідних цільової функції, яка використовується для аналізу кривизни функції та побудови методів другого порядку.

4. Методи другого порядку: метод Ньютона використовує градієнт і гессіан для обчислення кроків оптимізації. Цей метод швидше збігається, але потребує обчислення гессіану. Квазі-ньютонові методи: приклади включають bfgs та l-bfgs, які наближають гессіан для зменшення обчислювальної складності [10].

Дискретна математика відіграє важливу роль у машинному навчанні, особливо в теорії алгоритмів, комбінаторних методах, теорії графів і логіці. Вона надає інструменти та методи для роботи з дискретними структурами та допомагає вирішувати задачі, пов'язані з обробкою даних, оптимізацією та моделюванням.

Основні поняття дискретної математики:

1. Теорія множин: множини та підмножини – основні елементи теорії множин включають визначення множин, підмножин, об'єднання, перетин та різницю множин. Кардинальні числа: вимірюють потужність множин, що важливо для оцінки складності алгоритмів.

2. Комбінаторика: комбінаторні структури розглядають перестановки, комбінації та розміщення елементів множин. Використовуються для оцінки кількості можливих варіантів розташування даних або рішень. Біноміальні коефіцієнти: використовуються для розрахунку кількості комбінацій.

3. Теорія графів: використовуються для моделювання взаємозв'язків між об'єктами. Важливими поняттями є ступінь вершини, шляхи, цикли та компоненти зв'язності. Орієнтовані та неорієнтовані графи: орієнтовані графи мають направлені ребра, тоді як неорієнтовані – ненаправлені. Дерева та ліси: особливі види графів без циклів. Дерева мають важливе значення для структур даних та алгоритмів пошуку.

4. Теорія алгоритмів. Складність алгоритмів: вимірює кількість ресурсів (час, пам'ять), необхідних для виконання алгоритму. Основними поняттями є часова складність (о-нотація) та просторова складність. Жадібні алгоритми: використовуються для знаходження оптимальних рішень шляхом послідовного вибору локально оптимальних рішень. Динамічне програмування: метод для розв'язання складних задач шляхом розбиття їх на підзадачі та зберігання результатів розв'язання підзадач для уникнення повторних обчислень.

5. Булева алгебра: використовується для роботи з логічними виразами та операціями (and, or, not). Основні поняття включають істиннісні таблиці, закони де моргана та мінімізацію логічних виразів. Предикатна логіка: розширення булевої логіки, що включає квантори та предикати, які дозволяють робити висновки про властивості об'єктів [11].

1.5. Алгоритми машинного навчання

Існує загальний принцип, який лежить в основі всіх керованих алгоритмів машинного навчання для прогнозного моделювання. Алгоритми машинного навчання описуються як навчання цільової функції (f), яка найкраще відображає вхідні змінні (X) на вихідну змінну (Y): $Y = f(X)$.

Це загальне навчальне завдання, у якому ми хотіли б робити прогнози в майбутньому (Y), враховуючи нові приклади вхідних змінних (X). Ми не знаємо, як виглядає функція (f) або її форма. Якби ми це зробили, ми б використовували його безпосередньо, і нам не потрібно було б вивчати його з даних за допомогою алгоритмів машинного навчання.

Найпоширенішим типом машинного навчання є вивчення відображення Y = f(X), щоб робити прогнози Y для нового X. Це називається прогнозним моделюванням або прогнозною аналітикою , і наша мета – робити якомога точніші прогнози.

Найпоширеніші алгоритми машинного навчання, які використовуються для дослідження даних:

1. Лінійна регресія (Рис. 1.1) є, мабуть, одним із найвідоміших і добре зрозумілих алгоритмів статистики та машинного навчання.

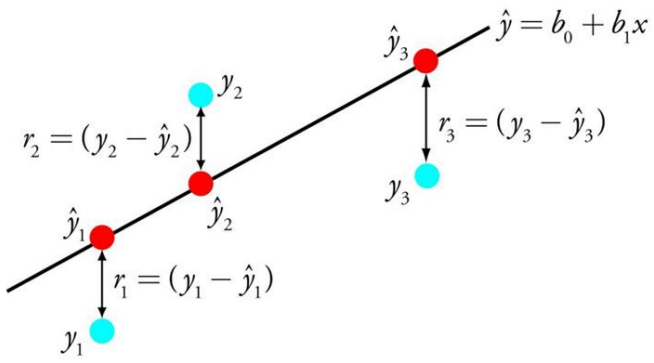

Рисунок 1.1 – Графік лінійної регресії

Прогнозне моделювання в першу чергу пов’язане з мінімізацією помилок моделі машинного навчання або створенням найточніших можливих передбачень за рахунок можливості пояснення. Ми будемо запозичувати, повторно використовувати та викрадати алгоритми з багатьох різних галузей, включаючи статистику, і використовувати їх для досягнення цих цілей.

Представлення лінійної регресії – це рівняння, яке описує лінію, яка найкраще відповідає зв’язку між вхідними змінними (x) і вихідними змінними (y), шляхом знаходження певних вагових коефіцієнтів для вхідних змінних, які називаються коефіцієнтами (B).

Наприклад: y = B0 + B1 * x

Ми спрогнозуємо y, враховуючи вхідні значення x, і метою алгоритму навчання лінійної регресії є знайти значення для коефіцієнтів B0 і B1.

Для вивчення моделі лінійної регресії з даних можна використовувати різні методики, наприклад рішення лінійної алгебри для звичайних методів найменших квадратів і оптимізацію градієнтного спуску.

Лінійна регресія існує вже більше 200 років і широко вивчається. Деякі хороші емпіричні правила під час використання цієї техніки полягають у видаленні змінних, які дуже схожі (корельовані), і усуненні шуму з ваших даних, якщо це можливо. Це швидка та проста техніка, а також хороший перший алгоритм для спроби [12].

2. Логістична регресія (Рис. 1.2) – це ще один метод, запозичений машинним навчанням із галузі статистики. Це основний метод для задач бінарної класифікації (задачі з двома значеннями класу).

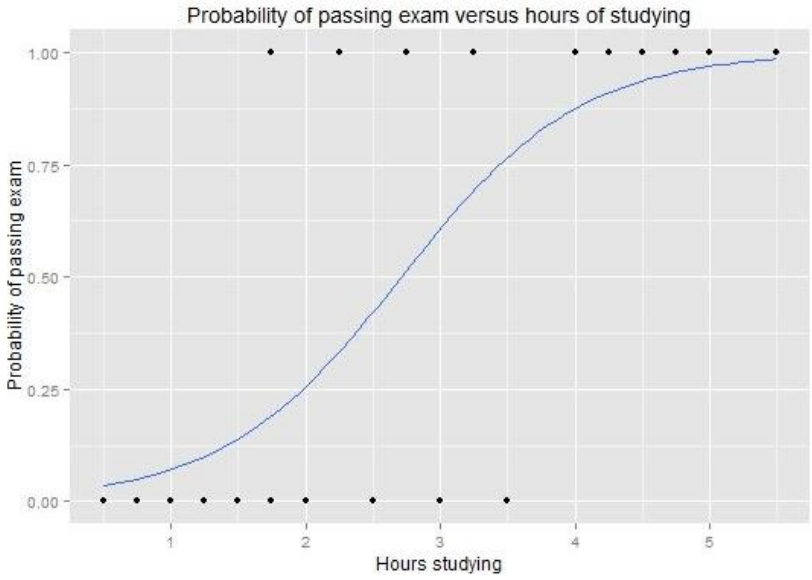

Рисунок 1.2 – Графік логістичної регресії

Логістична регресія схожа на лінійну регресію тим, що метою є знайти значення для коефіцієнтів, які зважують кожну вхідну змінну. На відміну від лінійної регресії, прогнозування результату перетворюється за допомогою нелінійної функції, яка називається логістичною функцією.

Логістична функція виглядає як велика літера S і перетворює будь-яке значення в діапазон від 0 до 1. Це корисно, оскільки ми можемо застосувати правило до вихідних даних логістичної функції, щоб прив’язати значення до 0 і 1 (наприклад, ЯКЩО менше 0,5, то вихід 1) і передбачити значення класу.

Логістична регресія: графік кривої логістичної регресії, що показує ймовірність здачі іспиту в порівнянні з годинами навчання

Через спосіб вивчення моделі прогнози, зроблені за допомогою логістичної регресії, також можна використовувати як ймовірність приналежності даного екземпляра даних до класу 0 або класу 1. Це може бути корисним для проблем, де потрібно надати більше обґрунтувань для передбачення.

Як і лінійна регресія, логістична регресія працює краще, коли ви видаляєте атрибути, які не пов’язані з вихідною змінною, а також атрибути, які дуже схожі (корельовані) один з одним. Це модель, яка швидко вивчається та ефективна в задачах двійкової класифікації [13].

3. Лінійний дискримінантний аналіз – це статистичний метод для розв'язку задачі класифікації (Рис. 1.3). З його допомогою будуються лінійні комбінації предикторів, що відділяють області одного класу від іншого. Якщо у вас більше двох класів, то алгоритм лінійного дискримінантного аналізу є кращим методом лінійної класифікації.

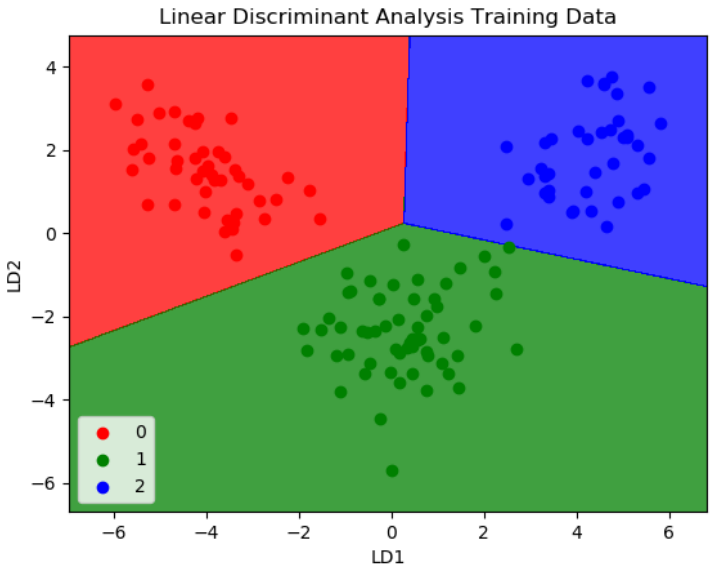

Рисунок 1.3 – Демонстрація лінійного дискримінантного аналізу

Представництво LDA є досить простим. Він складається зі статистичних властивостей ваших даних, розрахованих для кожного класу. Для однієї вхідної змінної це включає:

- середнє значення для кожного класу;
- дисперсія, розрахована для всіх класів.

Прогнози робляться шляхом обчислення дискримінантного значення для кожного класу та створення прогнозу для класу з найбільшим значенням. Ця техніка передбачає, що дані мають розподіл Гауса (дзвоноподібну криву), тому доцільно заздалегідь видалити викиди з даних. Це простий і потужний метод класифікації проблем прогнозного моделювання [13].

4. Дерева рішень (Рис. 1.4) є важливим типом алгоритму для машинного навчання з прогнозним моделюванням.

Відображенням моделі дерева рішень є бінарне дерево. Це ваше бінарне дерево з алгоритмів і структур даних, нічого особливого. Кожен вузол представляє одну вхідну змінну (x) і точку розбиття на цій змінній (припускаючи, що змінна числова).

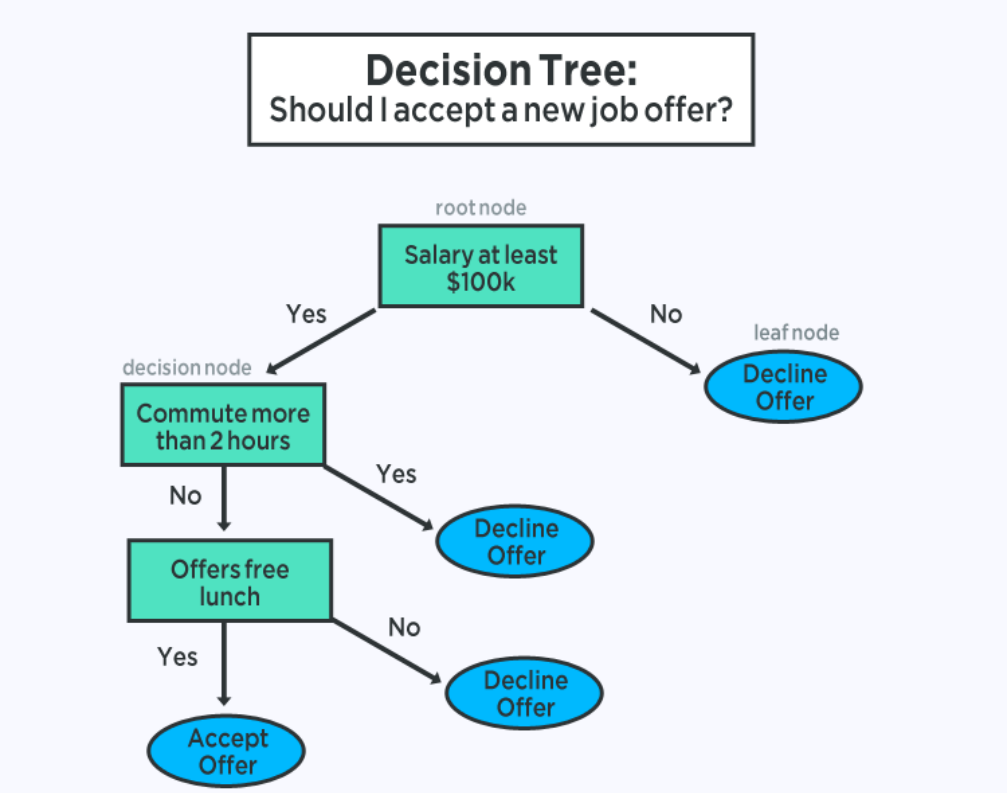

Рисунок 1.4 – Демонстрація дерева рішень

Листкові вузли дерева містять вихідну змінну (y), яка використовується для прогнозування. Прогнози робляться шляхом проходження поділів дерева до досягнення кінцевого вузла та виведення значення класу в цьому листовому вузлі.

Дерева швидко навчаються і дуже швидко роблять прогнози. Вони також часто є точними для широкого кола проблем і не потребують спеціальної підготовки ваших даних [13].

5. Naive Bayes – це простий, але напрочуд потужний алгоритм для прогнозного моделювання (Рис. 1.5).

Модель складається з двох типів ймовірностей, які можна обчислити безпосередньо з даних вашого навчання:

- ймовірність кожного класу;
- умовна ймовірність для кожного класу з урахуванням кожного значення x.

Після обчислення ймовірнісну модель можна використовувати для прогнозування нових даних за допомогою теореми Байєса. Коли ваші дані мають дійсне значення, прийнято вважати розподіл Гауса (дзвоноподібну криву), щоб ви могли легко оцінити ці ймовірності.

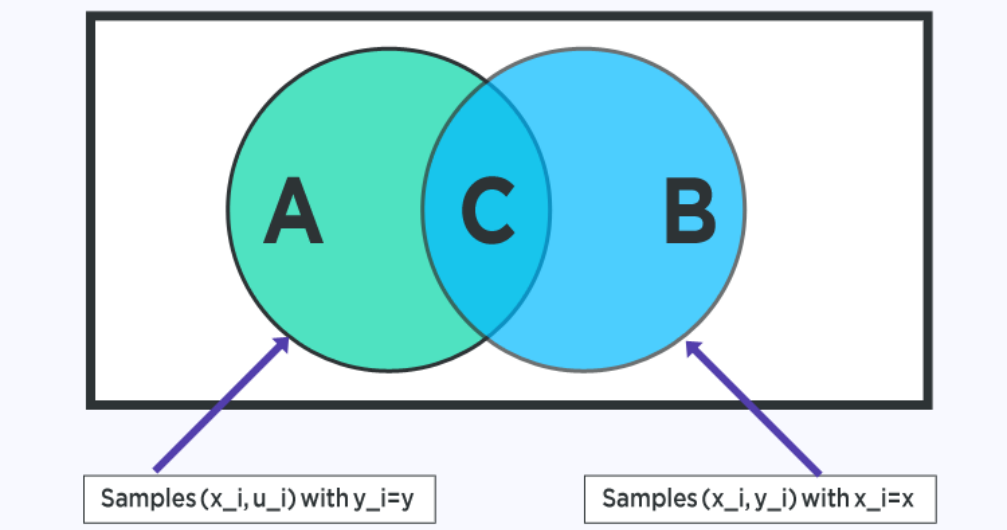

Рисунок 1.5 – Демонстрація теореми Байєса

Наївний Байєс називається наївним, оскільки передбачає, що кожна вхідна змінна є незалежною. Це сильне припущення та нереалістичне для реальних даних, тим не менш, техніка дуже ефективна для широкого спектру складних проблем [14].

6. K-найближчі сусіди. Алгоритм KNN (Рис. 1.6) дуже простий і ефективний. Представленням моделі для KNN є весь навчальний набір даних.

Прогнози робляться для нової точки даних шляхом пошуку в усьому навчальному наборі для K найбільш схожих екземплярів (сусідів) і підсумовування вихідної змінної для цих K екземплярів. Для задач регресії це може бути середня вихідна змінна, для задач класифікації це може бути значення класу режиму (або найпоширенішого).

Хитрість полягає в тому, як визначити подібність між екземплярами даних. Найпростіший спосіб, якщо ваші атрибути мають однаковий масштаб (усі, наприклад, у дюймах), – це використовувати евклідову відстань, число, яке можна обчислити безпосередньо на основі відмінностей між кожною вхідною змінною.

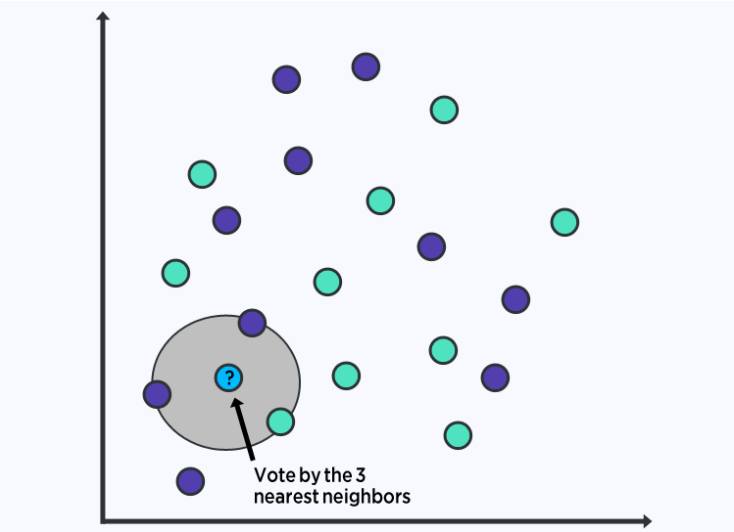

Рисунок 1.6 – Демонстрація K-найближчих сусідів

KNN може потребувати багато пам’яті або місця для зберігання всіх даних, але виконує обчислення (або вивчає) лише тоді, коли потрібне

передбачення, саме вчасно. Ви також можете оновлювати та курувати свої екземпляри навчання з часом, щоб підтримувати точність прогнозів.

Ідея відстані чи близькості може розпадатися на дуже великі розміри (багато вхідних змінних), що може негативно вплинути на продуктивність алгоритму для вашої проблеми. Це називається прокляттям розмірності . Він пропонує використовувати лише ті вхідні змінні, які є найбільш релевантними для прогнозування вихідної змінної [14].

7. Вивчення векторного квантування. Недоліком K-Nearest Neighbors (Рис. 1.7) є те, що вам потрібно триматися всього свого навчального набору даних . Алгоритм квантування векторів навчання (або скорочено LVQ) – це алгоритм штучної нейронної мережі, який дозволяє вам вибирати, скільки екземплярів навчання потрібно використовувати, і вивчає, як саме ці екземпляри мають виглядати.

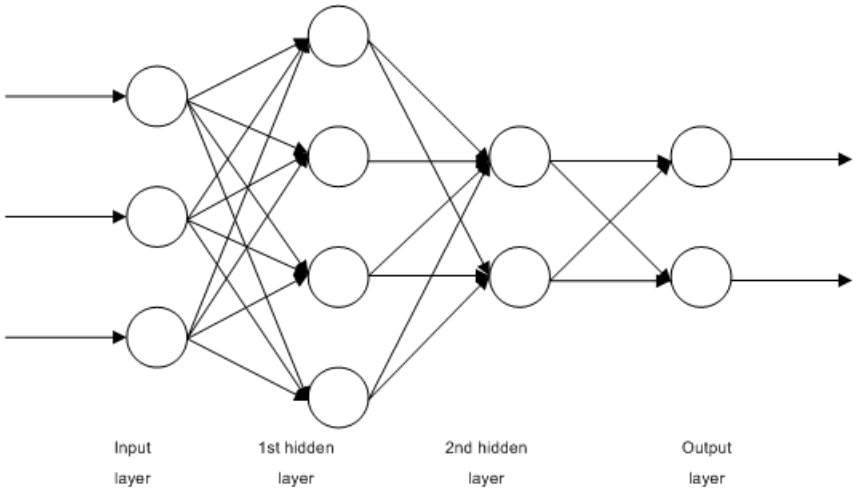

Рисунок 1.7 – Демонстрація векторного квантування

Представлення для LVQ є набором векторів кодової книги. Вони вибираються випадковим чином на початку та адаптуються для найкращого узагальнення навчального набору даних протягом кількох ітерацій алгоритму навчання. Після навчання вектори кодової книги можна використовувати для прогнозування так само, як K-Nearest Neighbors. Найсхожіший сусід (найкраще відповідний вектор кодової книги) визначається шляхом обчислення відстані між кожним вектором кодової книги та новим

екземпляром даних. Значення класу або (дійсне значення у випадку регресії) для найкращої відповідної одиниці потім повертається як прогноз. Найкращі результати досягаються, якщо ви масштабуєте свої дані, щоб мати той самий діапазон, наприклад від 0 до 1.

Якщо ви виявите, що KNN дає хороші результати на вашому наборі даних, спробуйте використати LVQ, щоб зменшити вимоги до пам’яті для зберігання всього навчального набору даних [15].

8. Метод опорних векторів (SVM) (Рис. 1.8) є, мабуть, одним із найпопулярніших і обговорюваних алгоритмів машинного навчання.

Гіперплощина – це лінія, яка розділяє простір вхідних змінних. У SVM вибрано гіперплощину, щоб найкраще розділити точки у просторі вхідних змінних за їхнім класом, або класом 0, або класом 1. У двовимірному просторі ви можете візуалізувати це як лінію, і припустімо, що всі наші вхідні точки можуть бути повністю розділені цією лінією. Алгоритм навчання SVM знаходить коефіцієнти, які призводять до найкращого розділення класів гіперплощиною.

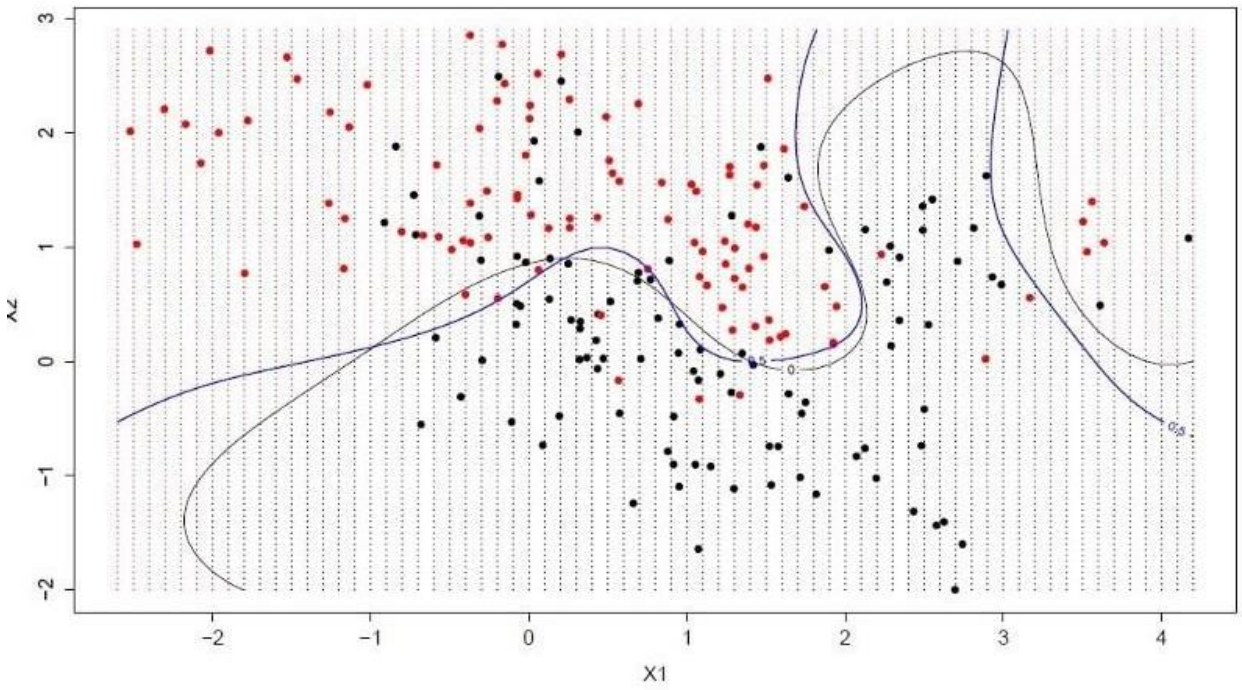

Рисунок 1.8 – Демонстрація методу опорних векторів

Відстань між гіперплощиною та найближчими точками даних називається запасом. Найкращою або оптимальною гіперплощиною, яка може розділити два класи, є лінія, яка має найбільший запас. Тільки ці моменти є релевантними при визначенні гіперплощини та при побудові класифікатора. Ці точки називаються опорними векторами. Вони підтримують або визначають гіперплощину. На практиці алгоритм оптимізації використовується для знаходження значень коефіцієнтів, які максимізують маржу.

SVM може бути одним із найпотужніших стандартних класифікаторів, і його варто спробувати на вашому наборі даних.

9. Random forest (Випадковий ліс) (Рис. 1.9) – один із найпопулярніших і найпотужніших алгоритмів машинного навчання. Це тип алгоритму ансамблевого машинного навчання під назвою Bootstrap Aggregation або bagging.

Bootstrap – це потужний статистичний метод для оцінки кількості на основі вибірки даних. Такий собі підлий. Ви берете багато зразків своїх даних, обчислюєте середнє значення, а потім усереднюєте всі середні значення, щоб отримати точнішу оцінку справжнього середнього значення.

У пакетуванні використовується той самий підхід, але замість нього для оцінки цілих статистичних моделей, найчастіше дерев рішень. Береться кілька зразків ваших навчальних даних, а потім будуються моделі для кожного зразка даних. Коли вам потрібно зробити прогноз для нових даних, кожна модель робить прогноз, і прогнози усереднюються, щоб отримати кращу оцінку справжнього вихідного значення [15].

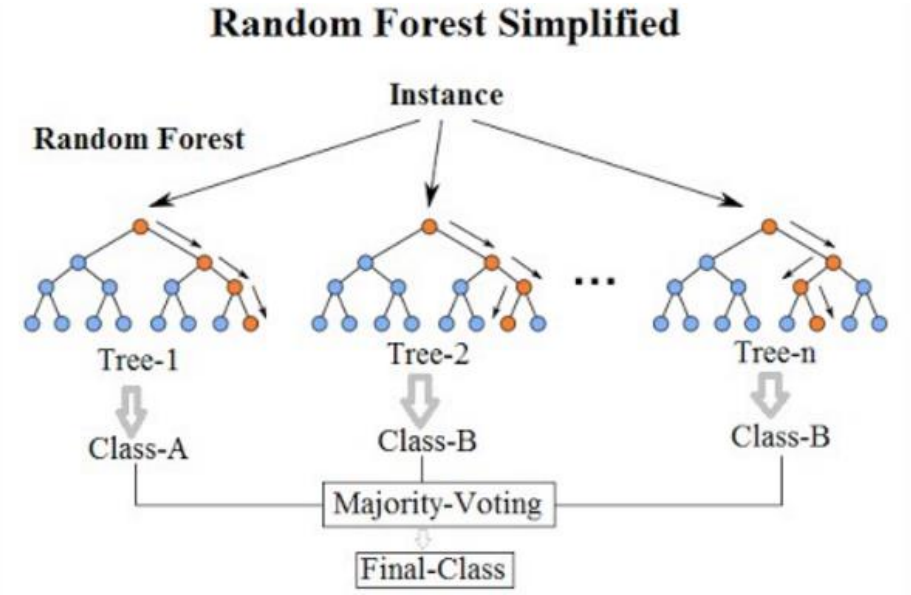

Рисунок 1.9 – Демонстрація випадкового лісу

Випадковий ліс – це зміна цього підходу, де дерева рішень створюються таким чином, що замість вибору оптимальних точок розбиття здійснюються субоптимальні розбиття шляхом введення випадковості.

Таким чином, моделі, створені для кожного зразка даних, відрізняються більше, ніж вони були б за інших умов, але все ще точні своїми унікальними та різними способами. Поєднання їхніх прогнозів призводить до кращої оцінки справжнього основного вихідного значення.

Якщо ви отримуєте хороші результати за допомогою алгоритму з високою дисперсією (як-от дерева рішень), ви часто можете отримати кращі результати, розміщуючи цей алгоритм [16].

10. AdaBoost – це метод ансамблю, який намагається створити сильний класифікатор із кількох слабких класифікаторів. Це робиться шляхом створення моделі з навчальних даних, а потім створення другої моделі, яка намагається виправити помилки з першої моделі. Моделі додаються до тих пір, поки навчальний набір не буде точно передбачено або додано максимальну кількість моделей.

AdaBoost (Рис. 1.10) був першим дійсно успішним алгоритмом підвищення, розробленим для двійкової класифікації. Це найкраща відправна

точка для розуміння бустингу. Сучасні методи підвищення базуються на AdaBoost, особливо на машинах стохастичного градієнтного підвищення.

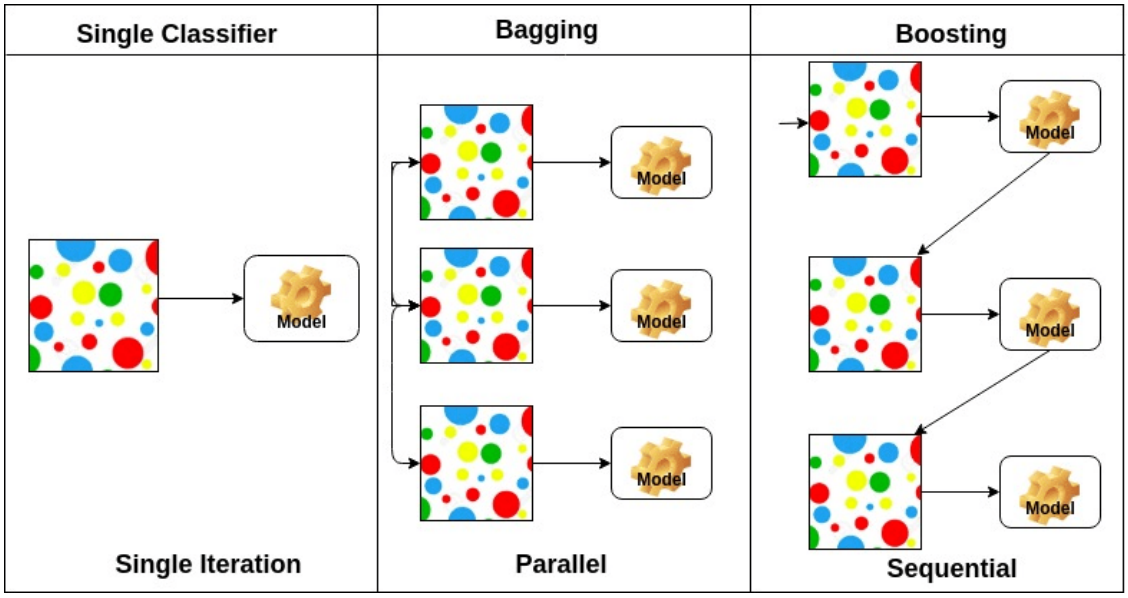

Рисунок 1.10 – Демонстрація AdaBoost

AdaBoost використовується з короткими деревами рішень. Після створення першого дерева продуктивність дерева на кожному екземплярі навчання використовується для зважування, скільки уваги наступне створене дерево має приділяти увагу кожному екземпляру навчання. Даним навчання, які важко передбачити, надається більша вага, тоді як випадкам, які легко передбачити, надається менша вага.

Моделі створюються послідовно одна за одною, кожна з яких оновлює ваги екземплярів навчання, які впливають на навчання, виконане наступним деревом у послідовності. Після того, як усі дерева створено, прогнозуються нові дані, а продуктивність кожного дерева зважується залежно від того, наскільки воно було точним у навчальних даних [17].

1.6. Згорткова нейронна мережа

Згорткова нейронна мережа (CNN) – це тип архітектури нейронної мережі глибокого навчання, який зазвичай використовується в комп’ютерному баченні. Комп’ютерний зір – це сфера штучного інтелекту, яка

дозволяє комп’ютеру розуміти та інтерпретувати зображення або візуальні дані.

Коли справа доходить до машинного навчання, штучні нейронні мережі працюють дуже добре. Нейронні мережі використовуються в різних наборах даних, таких як зображення, аудіо та текст. Різні типи нейронних мереж використовуються для різних цілей, наприклад, для прогнозування послідовності слів ми використовуємо рекурентні нейронні мережі , точніше LSTM , так само для класифікації зображень ми використовуємо згорточні нейронні мережі. У цьому блозі ми збираємося створити базовий будівельний блок для CNN.

У звичайній нейронній мережі є три типи шарів:

- вхідний шар: це шар, на якому ми надаємо вхідні дані для нашої моделі. Кількість нейронів у цьому шарі дорівнює загальній кількості ознак у наших даних (кількості пікселів у випадку зображення);
- прихований шар: вхідні дані з вхідного шару потім подаються в прихований шар. Може бути багато прихованих шарів залежно від нашої моделі та розміру даних. Кожен прихований шар може мати різну кількість нейронів, яка, як правило, перевищує кількість функцій. Вихідні дані кожного шару обчислюються шляхом множення матриці вихідних даних попереднього шару на вагові коефіцієнти цього шару, які можна дізнатися, а потім шляхом додавання зміщень, які можна дізнатися, з наступною функцією активації, яка робить мережу нелінійною;
- рівень виводу: вихідні дані з прихованого шару потім подаються в логістичну функцію, наприклад sigmoid або softmax, яка перетворює вихідні дані кожного класу в оцінку ймовірності кожного класу.

Дані вводяться в модель, і вихідні дані з кожного шару отримують на вищевказаному етапі, який називається прямою передачею . Потім ми обчислюємо похибку за допомогою функції похибок, деякими поширеними функціями похибок є крос-ентропія, похибка квадратичних втрат тощо. Функція похибок вимірює, наскільки добре працює мережа. Після цього ми

повертаємося до моделі шляхом обчислення похідних. Цей крок називається зворотним поширенням , який в основному використовується для мінімізації втрат [18].

Згорткова нейронна мережа – це розширена версія штучних нейронних мереж (ANN), яка переважно використовується для вилучення функції з набору даних матриці, подібної до сітки. Наприклад, візуальні набори даних, такі як зображення чи відео, де шаблони даних відіграють велику роль.

Згортка нейронної мережі складається з кількох рівнів, таких як вхідний рівень, згортковий рівень, рівень об’єднання та повністю зв’язані рівні, як показано на Рис. 1.11.

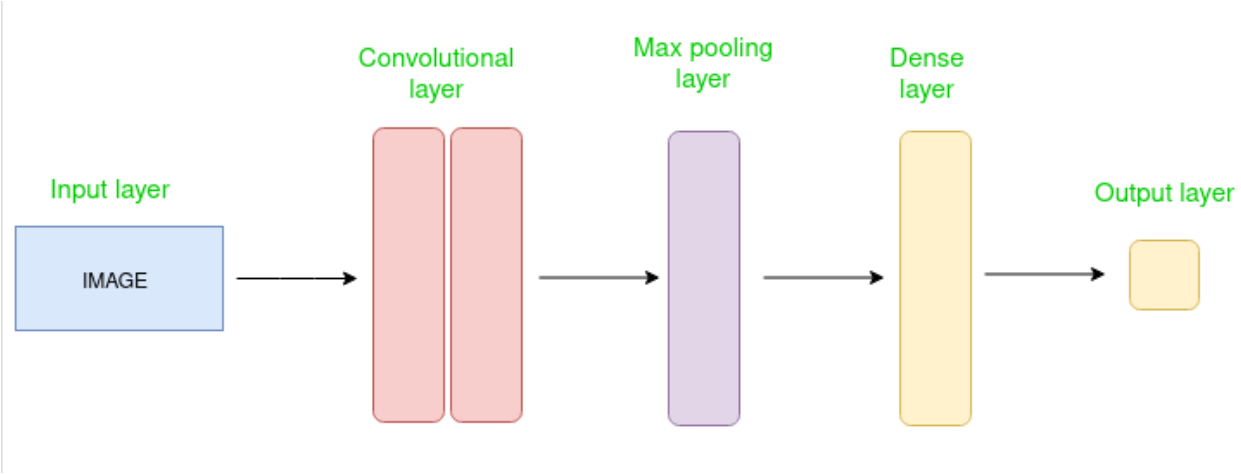

Рисунок 1.11 – Проста архітектура CNN

Згортковий рівень застосовує фільтри до вхідного зображення, щоб виділити особливості, рівень об’єднання зменшує дискретизацію зображення, щоб зменшити обчислення, а повністю підключений рівень робить остаточний прогноз. Мережа вивчає оптимальні фільтри за допомогою зворотного поширення та градієнтного спуску.

Нейронні мережі згортки або ковнети – це нейронні мережі, які мають спільні параметри. Уявіть, що у вас є образ. Його можна представити як прямокутний паралелепіпед, який має свою довжину, ширину (розмір зображення) і висоту (тобто канал як зображення зазвичай має червоні, зелені та сині канали), як показано на Рис. 1.12.

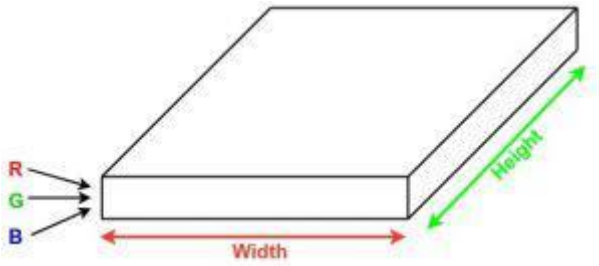

Рисунок 1.12 – Простий образ для CNN

А тепер уявіть, що ви берете невеликий фрагмент цього зображення та запускаєте на ньому невелику нейронну мережу, яка називається фільтром або ядром, із, скажімо, K виходами та представляючи їх вертикально. Тепер проведіть цією нейронною мережею по всьому зображенню, в результаті ми отримаємо інше зображення з різною шириною, висотою та глибиною. Замість лише каналів R, G та B тепер у нас більше каналів, але менша ширина та висота. Ця операція називається згорткою (Рис. 1.13). Якщо розмір патча такий самий, як і зображення, це буде звичайна нейронна мережа. Через цей маленький фрагмент у нас менше ваги.

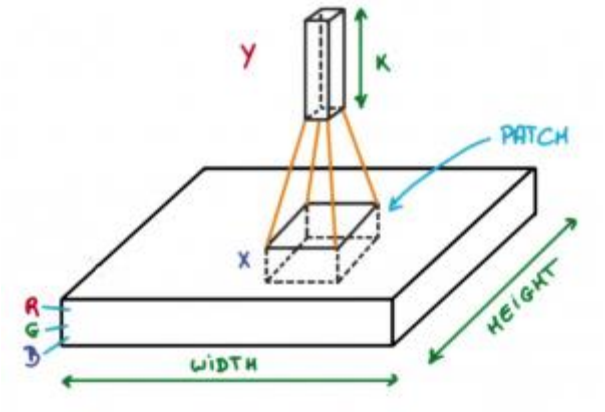

Рисунок 1.13 – Складніший образ для CNN

Згорткові шари складаються з набору доступних для навчання фільтрів (або ядер), які мають малу ширину та висоту та таку саму глибину, що й у вхідного об’єму (3, якщо вхідний рівень є вхідним зображенням).

Наприклад, якщо нам потрібно запустити згортку на зображенні розміром 34x34x3. Можливий розмір фільтрів може бути axax3, де «a» може мати значення 3, 5 або 7, але менше порівняно з розміром зображення.

Під час прямого проходу ми крок за кроком пересуваємо кожен фільтр по всьому вхідному об'єму, де кожен крок називається кроком (який може мати значення 2, 3 або навіть 4 для високорозмірних зображень) і обчислюємо скалярний добуток між ваги ядра та патч із вхідного обсягу.

Коли ми пересуваємо наші фільтри, ми отримаємо 2-D вихід для кожного фільтра, і ми складемо їх разом, як результат, ми отримаємо вихідний об'єм, глибина якого дорівнює кількості фільтрів. Мережа дізнається всі фільтри [19].

Шари, які використовуються для побудови ConvNets. Повна архітектура згорткових нейронних мереж також відома як ковнети. Ковнети – це послідовність шарів, і кожен шар перетворює один об'єм на інший за допомогою диференційованої функції.

Типи шарів: набори даних. Розглянемо приклад, запустивши ковнет на зображенні розміром 32 x 32 x 3.

Вхідні шари: це шар, на якому ми надаємо вхідні дані для нашої моделі. У CNN, як правило, введенням буде зображення або послідовність зображень. Цей шар містить необроблені вхідні дані зображення з шириною 32, висотою 32 і глибиною 3.

Згорткові шари: це шар, який використовується для вилучення об'єкта з вхідного набору даних. Він застосовує набір фільтрів, які можна вивчати, відомих як ядра, до вхідних зображень. Фільтри/ядра – це менші матриці, зазвичай 2×2, 3×3 або 5×5. він ковзає по вхідних даних зображення та обчислює скалярний добуток між вагою ядра та відповідним фрагментом вхідного зображення. Вихідні дані цього шару називають картами функцій. Припустимо, що ми використовуємо загалом 12 фільтрів для цього шару, ми отримаємо вихідний обсяг розміром 32 x 32 x 12.

Рівень активації: додаючи функцію активації до вихідних даних попереднього рівня, рівні активації додають мережі нелінійності. він застосує функцію поелементної активації до виходу шару згортки. Деякі поширені функції активації RELU : max(0, x), Tanh , Leaky RELU тощо. Обсяг залишається незмінним, отже вихідний обсяг матиме розміри 32 x 32 x 12.

Рівень об’єднання: цей рівень періодично вставляється в ковнети, і його основна функція полягає в зменшенні розміру тому, що робить обчислення швидкими, зменшує обсяг пам’яті, а також запобігає переобладнанню. Два поширені типи шарів об’єднання – максимальне об’єднання та середнє об’єднання . Якщо ми використовуємо максимальний пул із фільтрами 2 x 2 і кроком 2, результуючий об’єм матиме розмір 16x16x12 (Рис. 1.14).

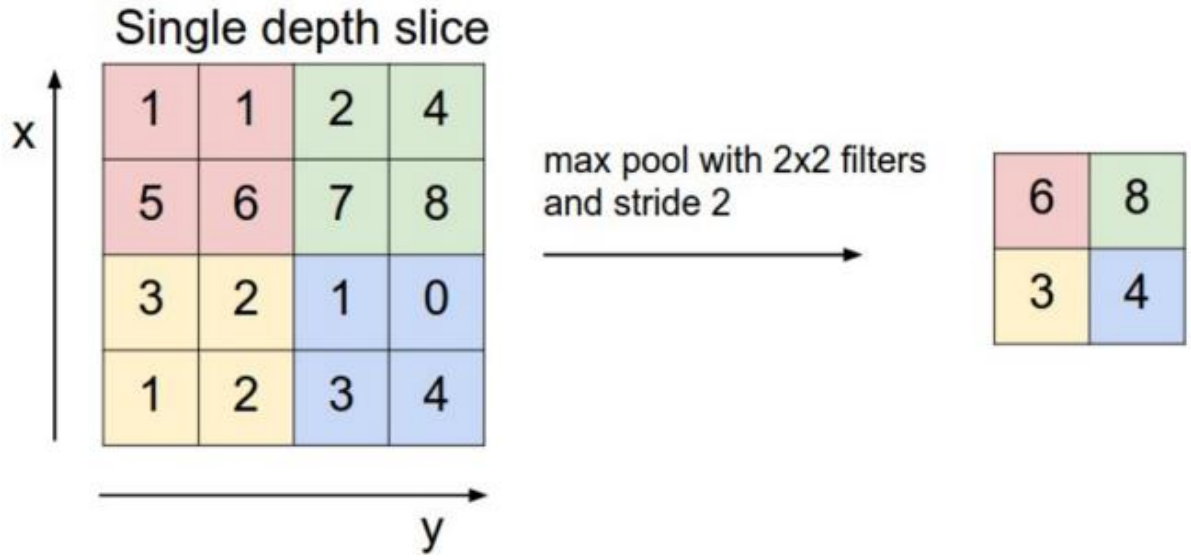

Рисунок 1.14 – Одиниця ConvNets для CNN

Зведення: отримані карти функцій зводяться в одновимірний вектор після шарів згортки та об’єднання, щоб їх можна було передати в повністю зв’язаний шар для категоризації чи регресії.

Повністю підключені шари: він бере вхідні дані з попереднього шару та обчислює остаточну класифікацію або завдання регресії.

Вихідний рівень: вихідні дані з повністю пов’язаних шарів потім подаються в логістичну функцію для завдань класифікації, як-от sigmoid або softmax, яка перетворює вихідні дані кожного класу в оцінку ймовірності

кожного класу [19]. Процес роботи згорткової нейронної мережі показано наглядно на Рис. 1.15.

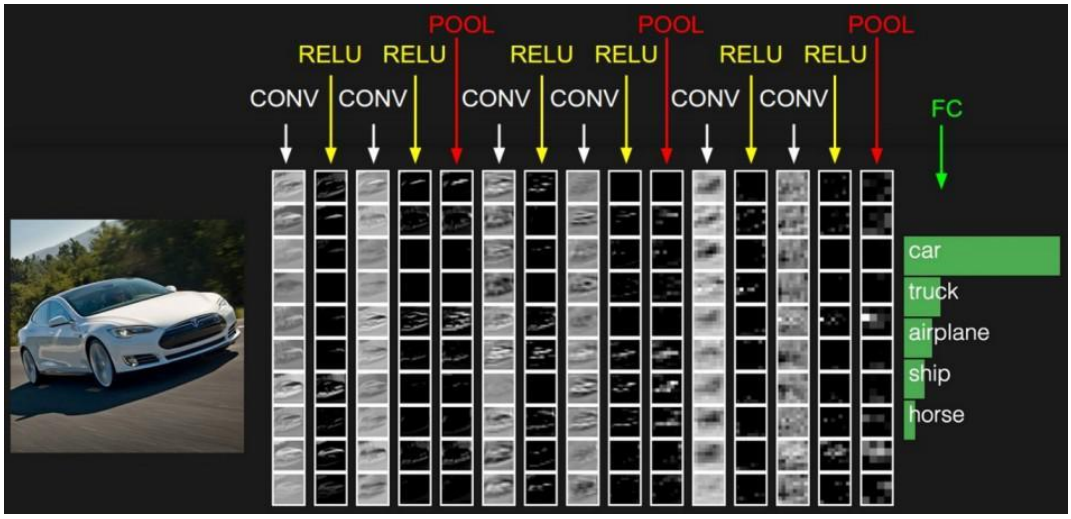

Рисунок 1.15 – Процес роботи згорткової нейронної мережі

1.7. Архітектури нейронних мереж для сегментації зображень

Сегментація зображення – це процес розділення цифрового зображення на кілька сегментів (піксельних груп або об'єктів), щоб спростити або змінити його представлення на більш значуще і легше для аналізу. Основна мета сегментації полягає в тому, щоб локалізувати об'єкти та межі (лінії, криві та інші форми) в зображеннях. Це один із основних завдань у галузі комп'ютерного зору та обробки зображень. Просту схему сегментації зображення показано на Рис. 1.16.

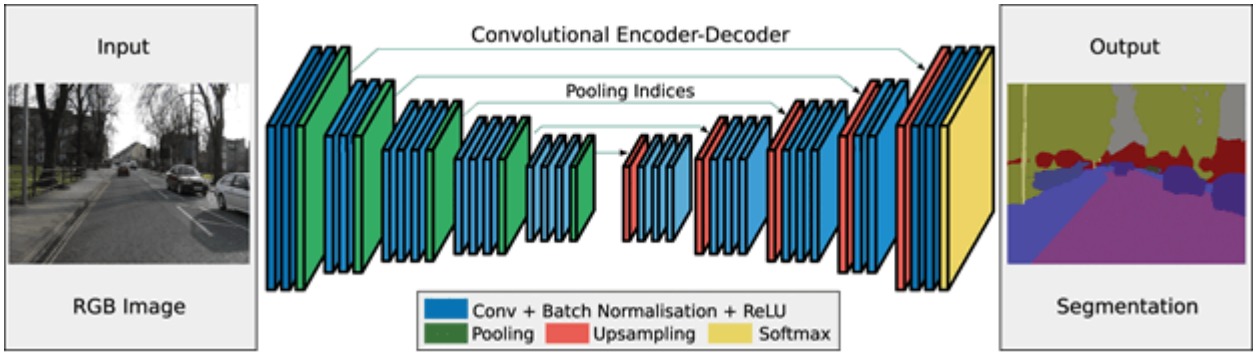

Рисунок 1.16 – Проста схема сегментації зображення

Основна архітектура сегментації зображення складається з кодера та декодера. Кодер витягує особливості із зображення за допомогою фільтрів.

Декодер відповідає за генерацію кінцевого результату, який зазвичай є маскою сегментації, що містить контур об’єкта. Більшість архітектур мають цю архітектуру або її варіант [20].

1. U-Net – це згорткова нейронна мережа, спочатку розроблена для сегментації біомедичних зображень (Рис. 1.17). При візуалізації його архітектура виглядає як буква U і звідси назва U-Net. Його архітектура складається з двох частин: ліва частина – звужуючий шлях і права частина – розширювальний шлях. Метою скорочувального шляху є захоплення контексту, тоді як роль розширеного шляху полягає в тому, щоб допомогти в точній локалізації.

Шлях скорочення складається з двох згорток три на три. За згортками слідує випрямлена лінійна одиниця та обчислення максимального об’єднання два на два для зменшення дискретизації.

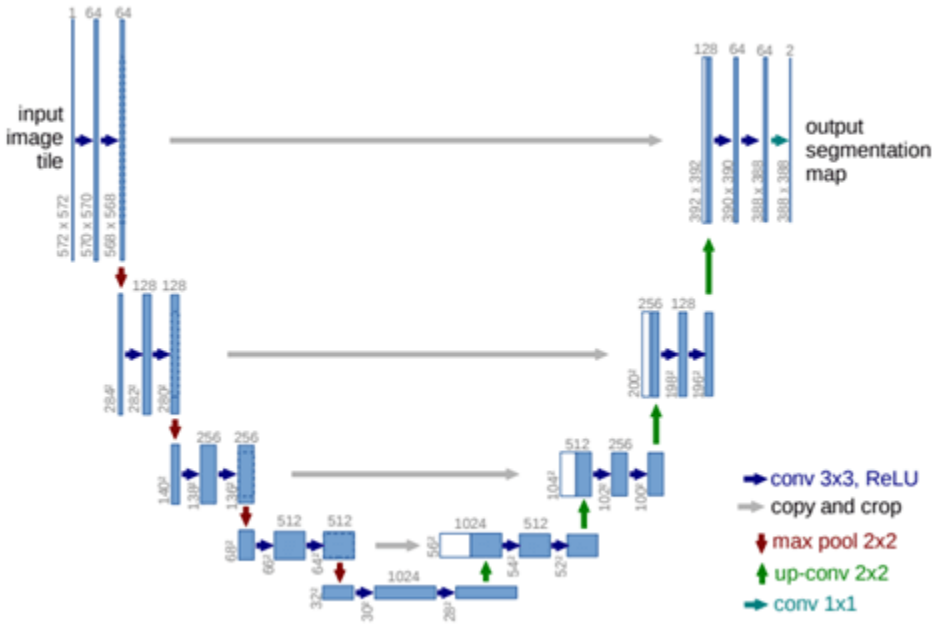

Рисунок 1.17 – Сегментація зображення архітектури U-net

2. FastFCN – швидка повністю згорткова мережа. У цій архітектурі для заміни розширених згорток використовується модуль Joint Pyramid Upsampling (JPU), оскільки вони споживають багато пам’яті та часу . Він використовує повністю підключену мережу в своєму ядрі, застосовуючи JPU

для підвищення дискретизації. JPU підвищує дискретизацію карт функцій із низькою роздільною здатністю до карт функцій із високою роздільною здатністю (див. Рис. 1.18).

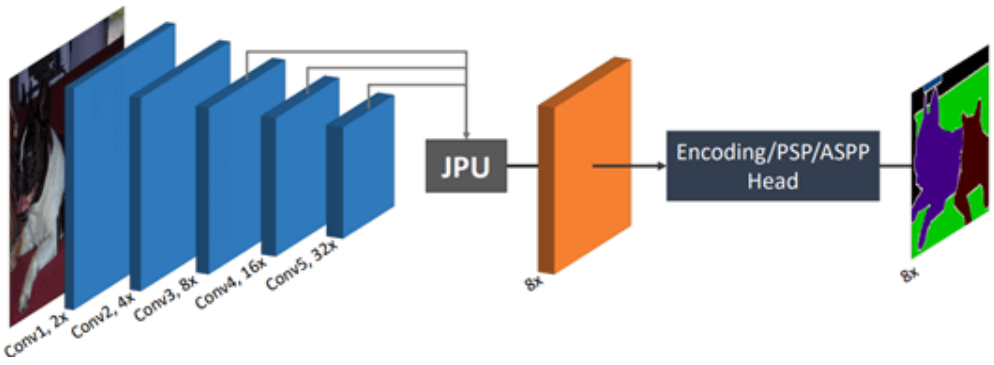

Рисунок 1.18 – Демонстрація FastFCN: переосмислення розширеної згортки в магістралі для семантичної сегментації»

3. Gated-SCNN. Ця архітектура складається з двопотокової архітектури CNN. У цій моделі окрема гілка використовується для обробки інформації про форму зображення [20]. Потік форми використовується для обробки інформації про межі. Gated-SCNN показано на Рис. 1.19.

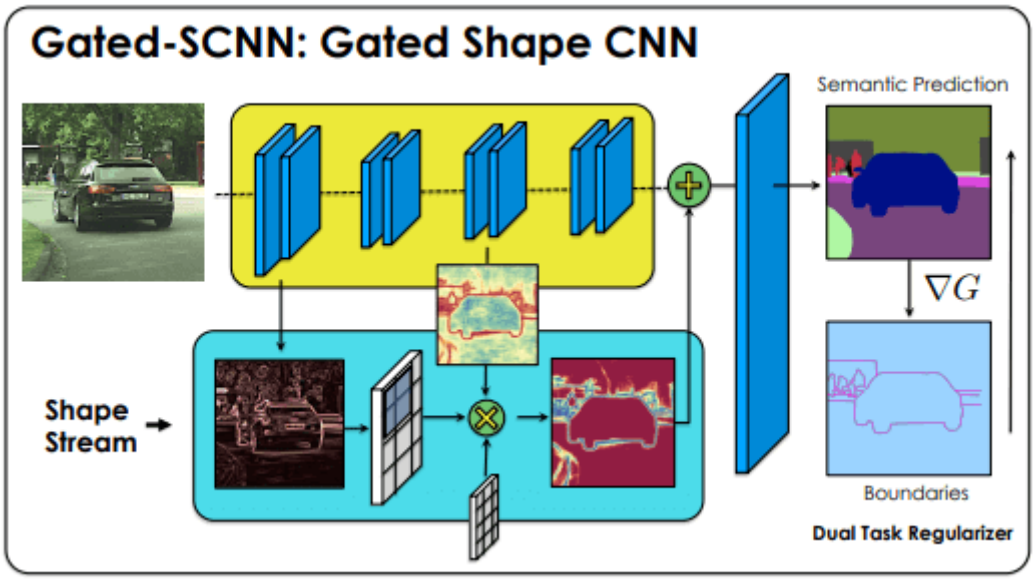

Рисунок 1.19 – Демонстрація Gated-SCNN

4. DeepLab. У цій архітектурі згортки з фільтрами підвищеної дискретизації використовуються для завдань, які включають щільне

прогнозування (див. Рис. 1.20). Сегментація об’єктів у кількох масштабах виконується за допомогою об’єднання просторової піраміди атрусу. Нарешті, DCNN використовуються для покращення локалізації меж об’єктів. Атрусна згортка досягається підвищенням дискретизації фільтрів шляхом вставки нулів або розрідженої вибірки вхідних карт ознак.

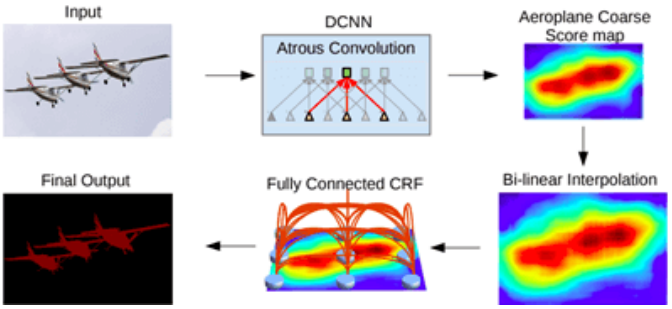

Рисунок 1.20 – Семантична сегментація зображень із глибокими згортковими мережами за допомогою DeepLab

5. Маска R-CNN. У цій архітектурі об’єкти класифікуються та локалізуються за допомогою обмежувальної рамки та семантичної сегментації, яка класифікує кожен піксель за набором категорій. Кожна область інтересу отримує маску сегментації (див. Рис. 1.21). Мітка класу та обмежувальна рамка створюються як кінцевий результат. Архітектура є розширенням Faster R-CNN. Faster R-CNN складається з глибокої згорткової мережі, яка пропонує регіони, і детектора, який використовує регіони [21].

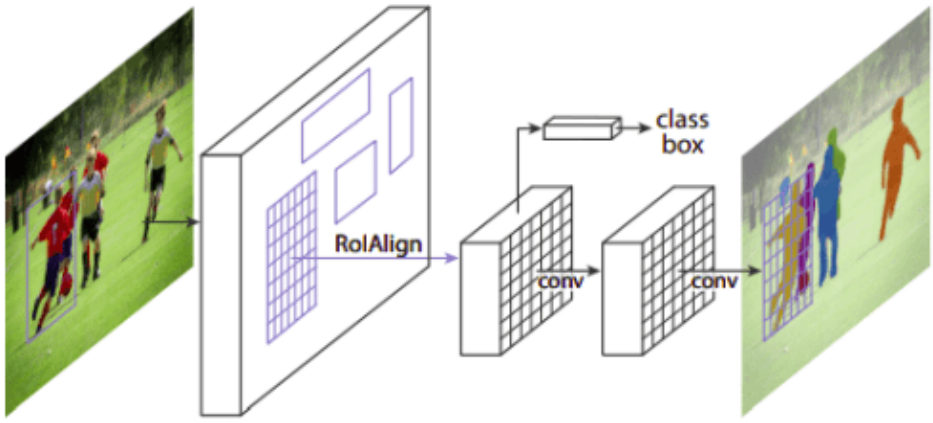

Рисунок 1.21 – Маска R-CNN

1.8. Видалення фону із зображення, або матування, за допомогою машинного навчання

Матування – це процес поділу зображення на передній план і фон, щоб ви могли поєднати передній план із новим фоном. Це ключова техніка ефекту зеленого екрана, і вона широко використовується у виробництві відео, графіки та споживчих програмах. Щоб змоделювати цю проблему, ми представляємо кожен піксель у захопленому зображенні як комбінацію переднього плану та фону за допомогою формули рівняння сполучення:

$$C = F * \alpha + B * (1 - \alpha)$$

Задача матування полягає в розв’язанні переднього плану (F), фону (B) і прозорості (альфа) для кожного пікселя отриманого зображення (C). Зрозуміло, що це дуже невизначено, і оскільки зображення мають канали RGB, це вимагає вирішення 7 невідомих із 3 спостережуваних значень.

Одним із можливих підходів є використання сегментації для виділення переднього плану для компонування. Хоча сегментація досягла величезних успіхів за останні роки, вона не розв’язує повне рівняння матування. Сегментація призначає двійкову мітку (0,1) кожному пікселю для представлення переднього плану та фону замість визначення безперервного значення альфа. Наслідки цього спрощення видно в прикладі, наведеному на Рис. 1.22.

Рисунок 1.22 – Демонстрація проблеми сегментування для матування

Ділянки навколо краю, зокрема у волоссі, мають справжнє значення альфа *від* 0 до 1. Тому бінарна природа сегментації створює різку межу навколо переднього плану, залишаючи видимі артефакти. Рішення для часткової прозорості та кольору переднього плану дає змогу набагато краще компонувати у другому кадрі.

У складному та різноманітному світі редагування зображень завдання матування зображення виділяється як критична проблема. Мета полягає в тому, щоб створити альфа-матовий процес, невід'ємний від досягнення високоякісних результатів у різних застосуваннях. Це завдання включає не лише ідентифікацію кожного пікселя на зображенні, але й визначення його рівня прозорості, що вимагає точності та витонченості.

Складність матування зображення полягає в тому, що він вимагає точного прогнозування стану кожного пікселя, включаючи рівень його прозорості. Це непростий подвиг, оскільки навіть незначні неточності можуть бути яскраво очевидними для людського ока, яке дуже вправно помічає такі розбіжності. Складність отримання візуально приємних результатів робить матування зображень особливо складним аспектом обробки зображень.

Щоб спростити цю складну проблему, використовуються різні допоміжні вхідні дані, такі як трикарти , карти глибини, маски сегментації та грубі альфа-матки. Ці вхідні дані надають додатковий контекст і інформацію, які допомагають у процесі матування, роблячи його дещо легшим завданням.

Історично такі методи, як Chroma Keying, використовувалися, коли був відомий колір фону, пропонуючи просте рішення. Попередні алгоритми ґрунтувалися на екстраполяції невідомих значень пікселів із відомих, головним чином зосереджуючись на сусідніх пікселях. Однак із появою глибокого навчання можливості матування зображень значно розвинулися. Ці моделі пропонують чудові результати, але їх складніше розробити. Тому ця стаття зосереджена на цих передових рішеннях глибокого навчання.

Повністю автоматичні рішення для матування зображення користуються великим попитом, особливо для додатків у режимі реального

часу. Ці рішення є масштабованими та усувають потребу в людському втручанні, наприклад у наданні каракулів або трикарт, що робить їх ідеальними для широкого спектру застосувань.

Архітектури глибокого навчання, які використовуються для матування зображень, зазнали значного прогресу з 2017 року через складність завдання. Для вирішення тонкощів точного визначення властивостей кожного пікселя потрібні складні рішення.

Послідовний модельний підхід: один із підходів передбачає послідовне використання двох моделей, де результати першої моделі уточнюються або покращуються другою. Цей послідовний процес забезпечує більш детальний і точний результат.

Мультимодельний підхід із конкретними цілями: інший метод передбачає використання кількох моделей, кожна з яких має певний фокус, наприклад, одна модель зосереджується на семантичному відображенні, а інша – на виявленні меж. Потім модель злиття об’єднує ці вихідні дані для отримання вишуканого альфа-матового кольору.

Таким чином, матування зображення залишається складною, але захоплюючою проблемою в області редагування зображень. Глибоке навчання принесло значний прогрес, пропонуючи дедалі ефективніші рішення. Хоча досконалість у матуванні зображень може бути недосяжною метою, досягнення на даний момент забезпечують «достатньо хороше» рішення, яке постійно розширює межі того, що можливо у редагуванні зображень [22].

РОЗДІЛ 2
ПРАКТИЧНА ЧАСТИНА

2.1 Обгрунтування вибору середовища розробки

У якості середовища розробки для проектування програмного забезпечення для видалення заднього фону на зображенні із портретами було обрано хмарний сервіс Google Colab. Він може бути зручним середовищем для розробки програмного забезпечення, яке видаляє задній фон на зображеннях, зокрема через його особливості і переваги.

Google Colaboratory, або Colab, – це сервісна версія Jupyter Notebook, яка дозволяє вам писати і виконувати код на Python через браузер.

Jupyter Notebook – це безкоштовна розробка з відкритим вихідним кодом від проекту Jupyter. Блокнот Jupyter схожий на інтерактивний лабораторний зошит, який містить не лише нотатки та дані, але й код, який може маніпулювати цими даними. Код може виконуватися всередині блокнота, який, у свою чергу, може фіксувати вихідний код. Такі програми, як Matlab і Mathematica, були першопрохідцями цієї моделі, але на відміну від них, Jupyter – це веб-додаток на основі браузера.

Google Colab побудований на основі коду Project Jupyter і містить блокноти Jupyter, не вимагаючи встановлення локального програмного забезпечення. Але в той час як блокноти Jupyter підтримують декілька мов, включаючи Python, Julia та R, Colab наразі підтримує лише Python.

Ноутбуки Colab зберігаються в обліковому записі Google Диску і можуть бути доступними для інших користувачів, як і інші файли Google Диску. Блокноти також мають функцію автозбереження, але вони не підтримують одночасне редагування, тому співпраця має бути послідовною, а не паралельною.

Colab є безкоштовним, але має обмеження. Існують деякі типи коду, які заборонені, наприклад, медіа-сервіси та криптомайнінг. Доступні ресурси

також обмежені і змінюються залежно від попиту, хоча Google Colab пропонує про-версію з більш надійними ресурсами.

Обґрунтування вибору Google Colab як середовища для розробки цього програмного забезпечення включає в себе такі аспекти:

1. Доступ до GPU/TPU. Google Colab надає безкоштовний доступ до графічних процесорів (GPU) і тензорних процесорів (TPU). Для задач, що потребують високої обчислювальної потужності, таких як обробка зображень і тренування моделей машинного навчання, це може суттєво пришвидшити процес. Перевага: швидше оброблення зображень і тренування моделей, оскільки вилучення заднього фону може використовувати глибокі нейронні мережі, які потребують GPU для ефективної роботи.

2. Вбудовані бібліотеки та підтримка Python. Google Colab дозволяє працювати з Python, що є одним з найпопулярніших мов програмування для обробки зображень і машинного навчання. Colab вже включає багато корисних бібліотек, таких як TensorFlow, Keras, PyTorch, OpenCV, що полегшує розробку.

3. Безкоштовний доступ і простота використання. Google Colab надає безкоштовний доступ до ресурсів з можливістю зберігати свої проекти на Google Drive. Це робить його зручним для початкових розробок і експериментів без потреби в значних витратах.

4. Інтерактивне середовище. Google Colab забезпечує інтерактивну роботу з кодом і можливість швидкого тестування різних підходів до обробки зображень. Це дозволяє швидко перевіряти результати та налагоджувати алгоритми.

5. Можливість інтеграції з іншими сервісами. Colab можна інтегрувати з різними API та сервісами для додаткової обробки зображень або зберігання даних, що може бути корисним для розширення функціональності програми.

Таким чином, Google Colab є потужним і зручним середовищем для розробки програмного забезпечення, зокрема для того, яке видаляє задній фон на зображеннях, завдяки своїм можливостям для обчислень, інтеграції з

бібліотеками, безкоштовному доступу та легкому обміну кодом. Тож це середовище підходить як для експериментів, так і для розробки повноцінних рішень [23].

2.2 Обгрунтування вибору мови програмування

Вибір мови програмування для розробки програмного забезпечення, яке видаляє задній фон на зображеннях, залежить від різних факторів.

Для даного завдання було обрано мову програмування Python, яка є однією з найпопулярніших виборів для таких завдань, і ось обґрунтування для цього:

1. Багата екосистема бібліотек, зокрема бібліотеки для обробки зображень:

- opencv: потужна бібліотека для комп'ютерного зору, яка забезпечує широкий спектр інструментів для обробки зображень;
- pillow: простий у використанні інструмент для базової обробки зображень.

2. Бібліотеки для машинного навчання:

- tensorFlow: Популярна бібліотека для створення і тренування моделей глибокого навчання;
- pyTorch: Ще одна потужна бібліотека для глибокого навчання, зокрема для реалізації складних нейронних мереж.

3. Простота і читабельність коду. Python має чистий і зрозумілий синтаксис, що полегшує читання та написання коду. Простота синтаксису Python зменшує час на розробку та налагодження коду, що важливо для швидкого прототипування і тестування алгоритмів обробки зображень.

4. Активна спільнота і підтримка. Велика спільнота розробників Python, численні форуми, документація та навчальні ресурси, що допоможуть вирішити будь-яку проблему, що виникла під час розробки програмного забезпечення. Наявність великої спільноти та ресурсів дозволяє швидко

знаходити рішення для можливих проблем і отримувати допомогу у реалізації функцій.

5. Зручність інтеграції з іншими системами. Інтеграція з API та сервісами: Python легко інтегрується з різними API і зовнішніми сервісами для додаткової обробки або зберігання даних. Можливість інтеграції з API та сервісами дозволяє розширювати функціональність програми, наприклад, для зберігання результатів або використання сторонніх сервісів для обробки зображень.

6. Підтримка різних платформ: Python підтримує розробку для різних платформ, таких як Windows, macOS і Linux. Крос-платформеність Python дозволяє створювати програмне забезпечення, яке може бути запущено на різних операційних системах без значних змін у коді [24].

Підсумовуючи, Python є відмінним вибором для розробки програмного забезпечення, яке видаляє задній фон на зображеннях, завдяки своїй багатій екосистемі бібліотек, простому синтаксису, активній спільноті, можливості інтеграції з іншими системами та високій продуктивності при роботі з великими даними. Це робить його зручним інструментом для розробки, тестування і впровадження таких рішень.

2.3 Функціональні вимоги для програмного забезпечення

Функціональні вимоги для програмного забезпечення, яке видаляє задній фон на зображеннях, описують, що саме має робити система і як вона повинна взаємодіяти з користувачами. Перелік основних функціональних вимог для такого програмного забезпечення:

1. Завантаження зображення: користувач має змогу завантажити зображення, з якого потрібно видалити задній фон. Функціональність: підтримка різних форматів зображень (jpeg, png, gif тощо). Інтерфейс: функція завантаження через консольний інтерфейс користувача (gui), консольний ввід назви файла або через перетягування зображення.

2. Обробка зображення: система повинна автоматично обробляти завантажене зображення для видалення заднього фону. Функціональність: використання алгоритмів машинного навчання або комп'ютерного зору для сегментації зображення і видалення фону. Методи: застосування алгоритмів, таких як сегментація на основі глибоких нейронних мереж, методи контурного детектування.

3. Збереження та експорт зображення: користувач може зберегти оброблене зображення на своєму пристрої.

4. Підтримка різних пристроїв і платформ: програмне забезпечення має бути доступним на різних платформах і пристроях. Функціональність: підтримка веб-браузерів, мобільних додатків або десктопних додатків. Інтерфейс: адаптивний інтерфейс для різних пристроїв.

5. Безпека та конфіденційність: забезпечення безпеки даних користувачів і конфіденційності зображень.

2.4 Нефункціональні вимоги для програмного забезпечення

Нефункціональні вимоги описують якості та атрибути системи, які забезпечують її ефективність, надійність, зручність використання і підтримку. Для програмного забезпечення, яке видаляє задній фон на зображеннях, такі вимоги можуть включати:

1. Продуктивність: швидкість обробки зображень – система повинна обробляти зображення з видаленням заднього фону в межах певного часу (наприклад, не більше 2 хвилин на зображення). Масштабованість: програмне забезпечення повинно ефективно обробляти велику кількість запитів одночасно, не знижуючи продуктивності.

2. Надійність. Безвідмовність: система повинна мати високий рівень безвідмовності (наприклад, не менше 90% доступності).

3. Масштабованість: система повинна підтримувати додавання нових ресурсів (серверів, обчислювальних вузлів) для збільшення обчислювальної

потужності. Вертикальне масштабування: система повинна підтримувати збільшення ресурсів (процесор, пам'ять) на існуючих вузлах для підвищення продуктивності.

4. Обслуговуваність: програмне забезпечення повинно легко оновлюватися без перерви у роботі або з мінімальними перервами. Відстеження помилок: система повинна мати інструменти для моніторингу і відстеження помилок, а також забезпечувати швидке виправлення виявлених дефектів.

5. Портативність: система повинна бути легко переносимою на інші середовища або хмарні платформи.

6. Економічність: система повинна бути економічно ефективною з точки зору витрат на розробку, розгортання та експлуатацію.

7. Оптимізація ресурсів: програмне забезпечення повинно ефективно використовувати обчислювальні ресурси, мінімізуючи витрати.

8. Масштабованість розробки: код програми повинен бути модульним і легко розширюваним, щоб спростити додавання нових функцій і модифікацію існуючих. Програмне забезпечення повинно бути легко тестованим для забезпечення якості і стабільності системи.

Тобто нефункціональні вимоги є критичними для забезпечення успішної реалізації будь-якого програмного забезпечення, зокрема для того, що видаляє задній фон на зображеннях.

Вони визначають й встановлюють необхідний мінімум того, наскільки добре система буде виконувати свої функції з точки зору продуктивності, надійності, зручності використання, безпеки та інших важливих атрибутів. Врахування цих вимог допоможе створити ефективне і надійне рішення, що задовольнить потреби користувачів.

2.6 Діаграма прецедентів

Діаграма прецедентів (use case diagram) – це один з основних інструментів моделювання в мові UML (Unified Modeling Language), який використовується для опису функціональних вимог системи та її взаємодії з зовнішніми акторами (користувачами або іншими системами). Вона відображає систему з точки зору користувача і показує, які дії (прецеденти) можуть бути виконані в системі.

Основні елементи діаграми прецедентів:

1. Актори (Actors). Актор представляє користувача або іншу систему, яка взаємодіє з моделлю системи. Зображується у вигляді фігури людини або простого значка.

2. Прецеденти (Use Cases). Прецеденти описують специфічні дії або функції, які можуть бути виконані актором у системі. Кожен прецедент відповідає певній бізнес-функції або операції. Зображується у вигляді овала з назвою всередині.

3. Зв’язки (Associations). Лінії, які з'єднують акторів з прецедентами, вказують на взаємодію між ними. Вони показують, які актори мають доступ до яких прецедентів. Прості лінії, що з'єднують актора з прецедентом.

4. Системні межі (System Boundaries). Прямокутник, який обрамляє всі прецеденти, представляє межі системи. Все, що знаходиться всередині прямокутника, є частиною системи, а все, що поза ним, — зовнішні актори або інші системи. Прямокутник, в якому знаходяться прецеденти [25].

Діаграма прецедентів для програмного забезпечення, що розроблюється в межах даної роботи зображена на Рис. 2.1.

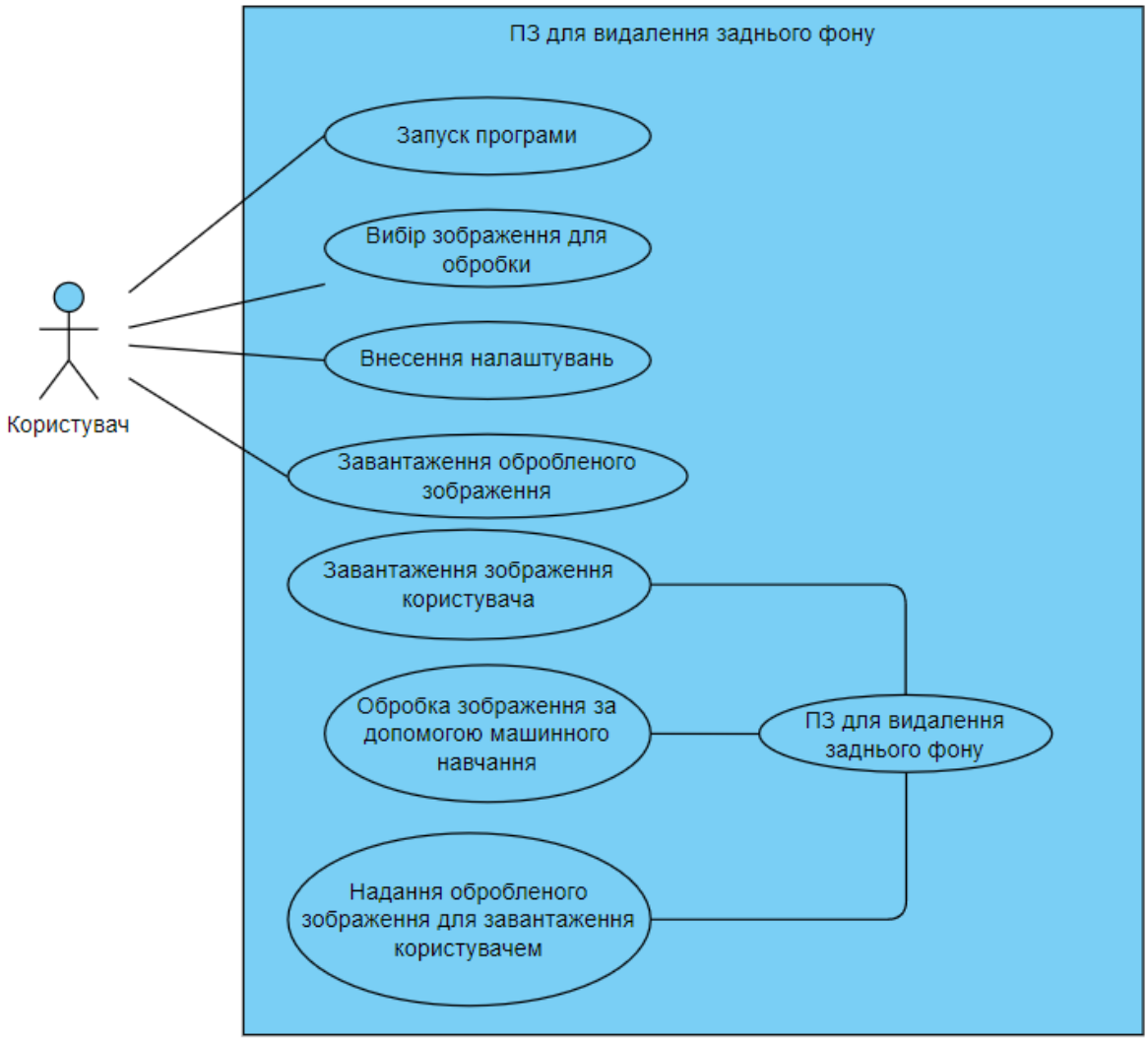

Рисунок 2.1 – Зображення діаграми прецедентів розроблюваного ПЗ

2.7 Діаграма активностей

Діаграма активностей (activity diagram) — це тип діаграми в мові uml (unified modeling language), який використовується для моделювання динамічних аспектів системи, зокрема бізнес-процесів і робочих процесів. Вона відображає послідовність дій або етапів, що відбуваються в системі або процесі, включаючи розгалуження і потоки управління.

Основні елементи діаграми активностей:

1. Дія (activity): окрема операція або крок у процесі, позначається у вигляді прямокутника з закругленими кутами.

2. Початковий вузол (initial node): вказує на початок процесу, позначається заповненою чорною круглою точкою.

3. Кінцевий вузол (final node): вказує на завершення процесу, позначається колом з обведеною чорною крапкою всередині.

4. Потік управління (control flow): стрілки, які з'єднують дії і показують порядок їх виконання.

5. Вузол прийняття рішення (decision node): точка розгалуження потоку управління на основі умов, позначається ромбом з вихідними стрілками.

6. Вузол злиття (merge node): об'єднує кілька потоків управління в один, позначається ромбом.

7. Паралельні дії (fork and join nodes): розподіл потоку управління на кілька паралельних гілок і їх об'єднання, позначаються вертикальною або горизонтальною лінією з кількома вихідними або вхідними стрілками.

Призначення діаграми активностей:

1. Моделювання бізнес-процесів: відображення бізнес-процесів і операційних процедур, демонструючи послідовність і взаємозв'язок між діями.

2. Аналіз і проектування: допомога в аналізі та проектуванні процесів, визначення вузьких місць і оптимізація робочих потоків.

3. Документація: служить документом, що описує динамічні аспекти системи, допомагаючи в розробці та підтримці системи.

4. Покращення комунікації: сприяє кращому розумінню процесів серед зацікавлених сторін, забезпечуючи наочне представлення робочих потоків [26].

Діаграма активностей для програмного забезпечення, що розроблюється в межах даної роботи зображена на Рис. 2.2.

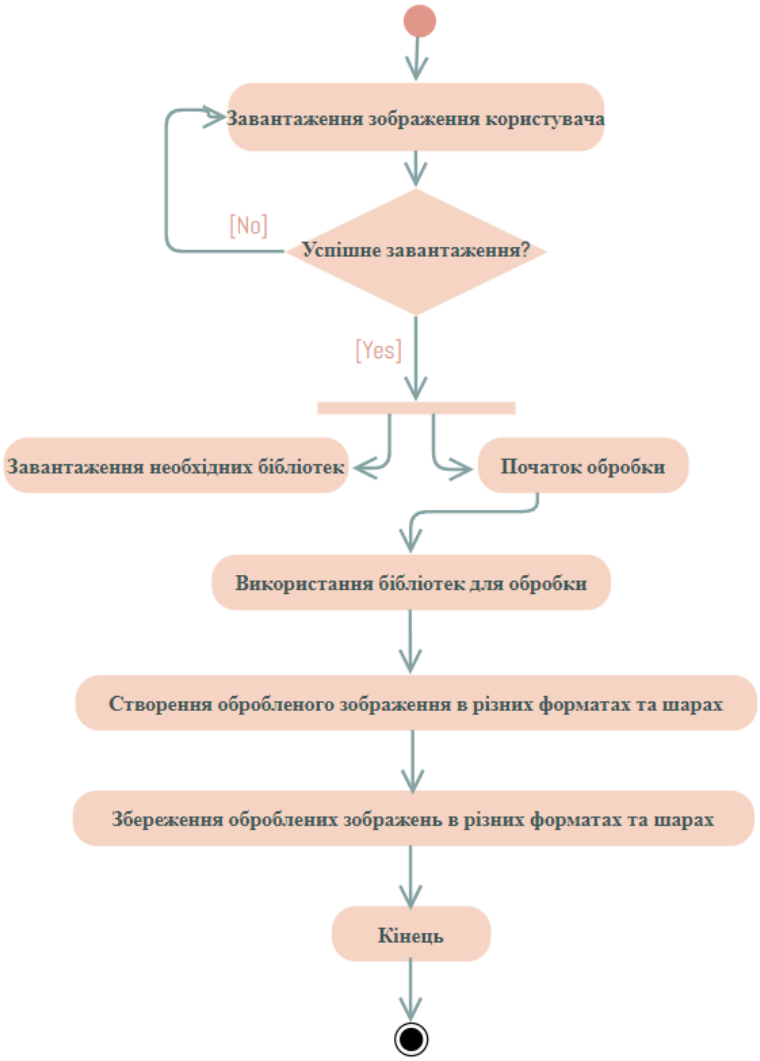

Рисунок 2.2 – Зображення діаграми активностей розроблюваного ПЗ

2.8 Класова діаграма

Класова діаграма (class diagram) є структурною діаграмою uml, яка описує структуру системи, показуючи її класи, їх атрибути, методи і взаємозв'язки між об'єктами [27]. Нижче наведено приклад класової діаграми для бібліотек pillow, image, torch, і pil.

Пакет pillow є форком бібліотеки pil (python imaging library), який додає підтримку python 3.x і більше сучасні функціональні можливості.

Основні класи в pillow:

- image: атрибути: width, height, mode, format. методи: open(), save(), show(), resize(), crop(), rotate(), convert();

- imagedraw: методи: draw(), line(), rectangle(), ellipse(), text();
- imagefilter: фільтри: blur, contour, detail, edge_enhance;
- imageenhance: методи: color(), contrast(), brightness(), sharpness();
- torch. бібліотека torch використовується для машинного навчання, особливо для роботи з глибокими нейронними мережами.

Основні класи в torch:

- tensor: атрибути: shape, dtype, device. методи: to(), cpu(), cuda(), numpy(), view(), permute(), transpose();
- nn.module: методи: forward(), backward(), parameters(), to();
- optimizer: методи: step(), zero_grad();
- dataset: методи: len(), getitem();
- dataloader: атрибути: batch_size, shuffle. методи: iter(), next().

Діаграма класів для програмного забезпечення, що розроблюється в межах даної роботи зображена на Рис. 2.3.

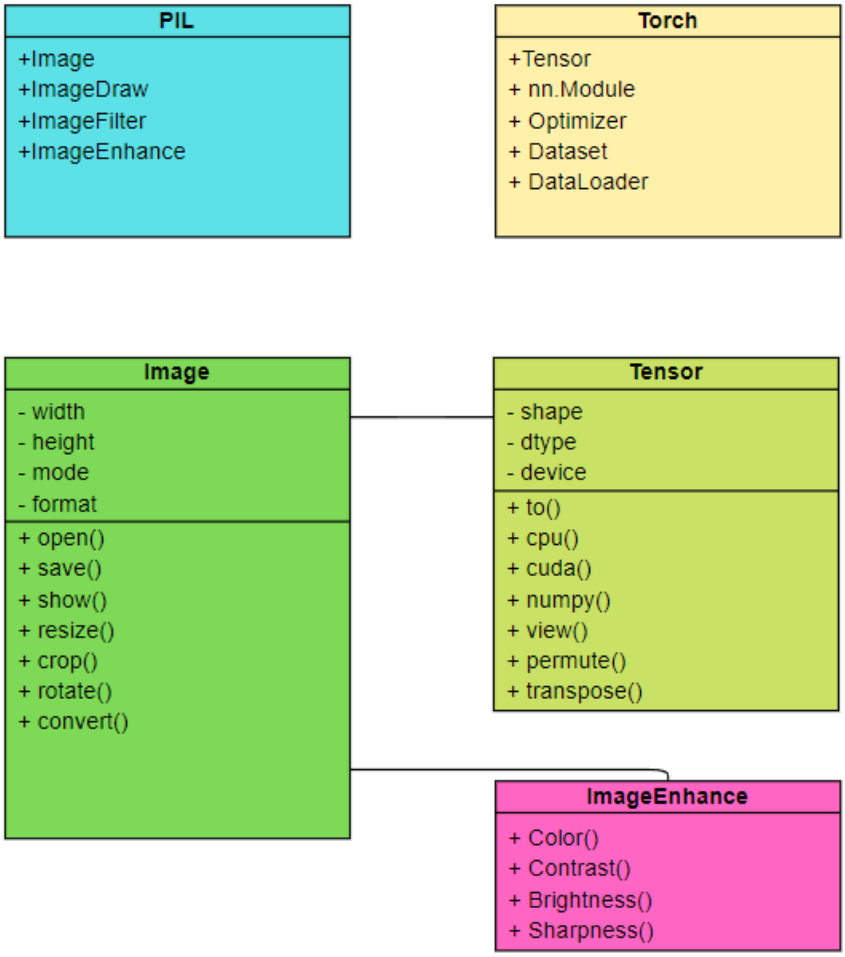

Рисунок 2.3 – Зображення діаграми класів розроблюваного ПЗ

2.9 Програмна реалізація

Програмна реалізація програмного забезпечення для видалення фону проводилось у хмарному середовищі Google Colab на мові програмування Python, переваги яких були описані вище.

Стартова сторінка та вікно Google Colab мають вигляд, як показано на Рис. 2.4.

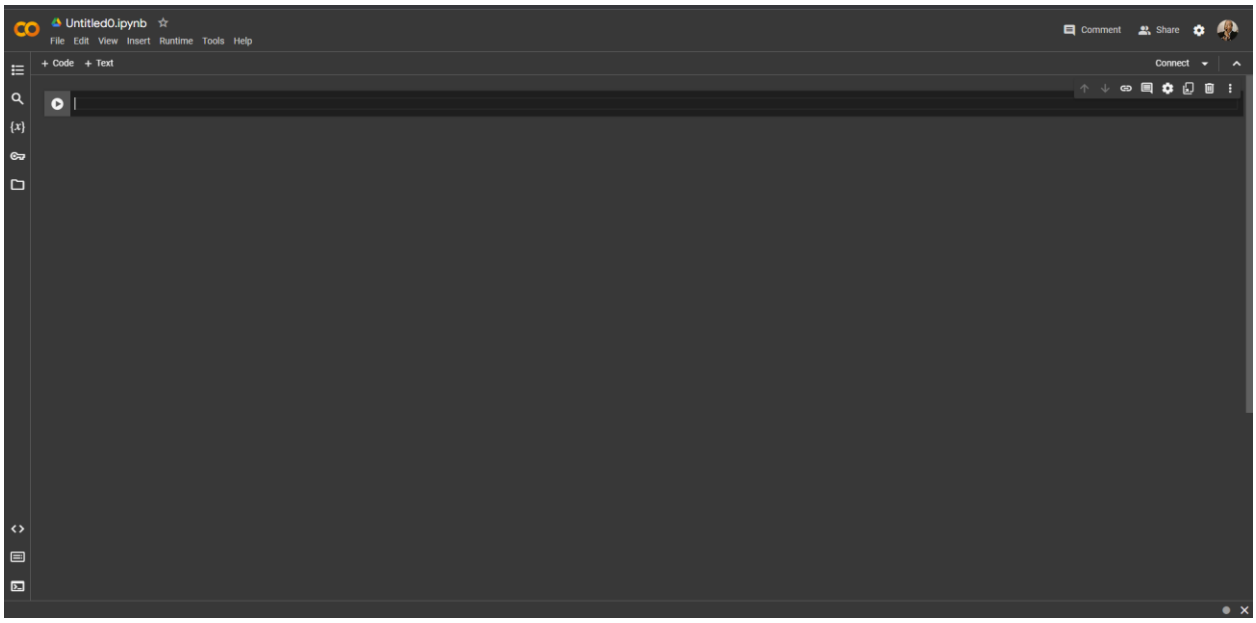

Рисунок 2.4 – Стартова сторінка та вікно Google Colab

За допомогою спеціальних кнопок можна додати блоки коду та тексту, як показано на Рис. 2.5.

Рисунок 2.5 – Додавання блоків коду та тексту

Також хмарний сервіс має інші інструменти для розробки та роботи із проектом, як показано на Рис. 2.6-2.13.

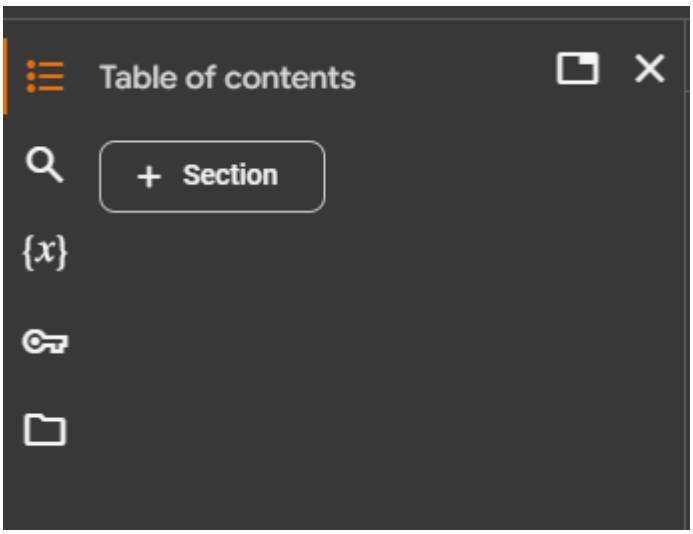

Рисунок 2.6 – Інструменти для роботи із проектом в Google Colab

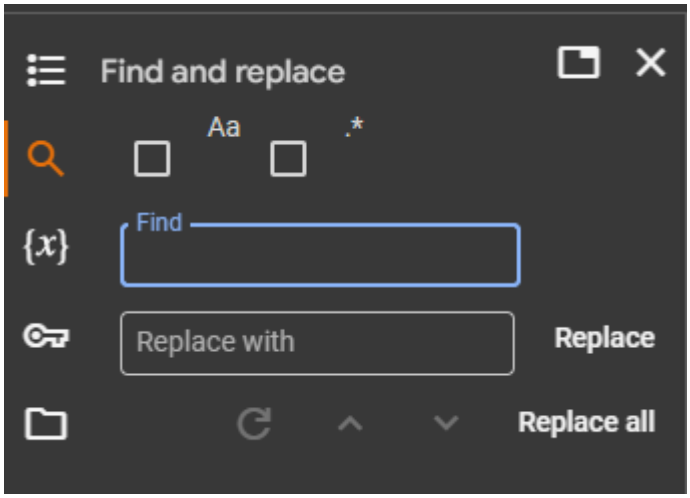

Рисунок 2.7 – Інструменти для роботи із проектом в Google Colab

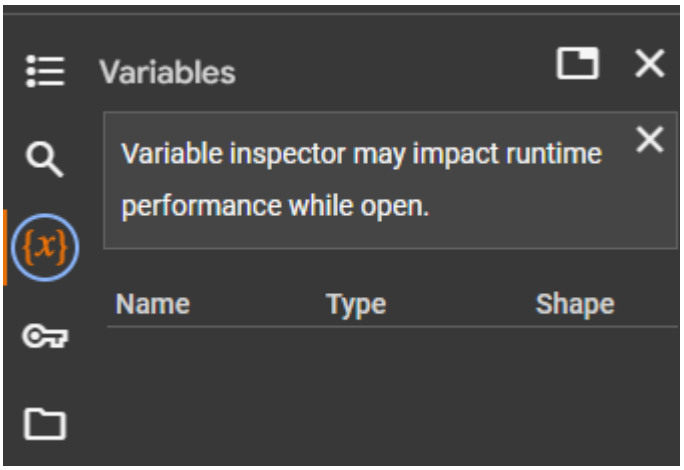

Рисунок 2.8 – Інструменти для роботи із проектом в Google Colab

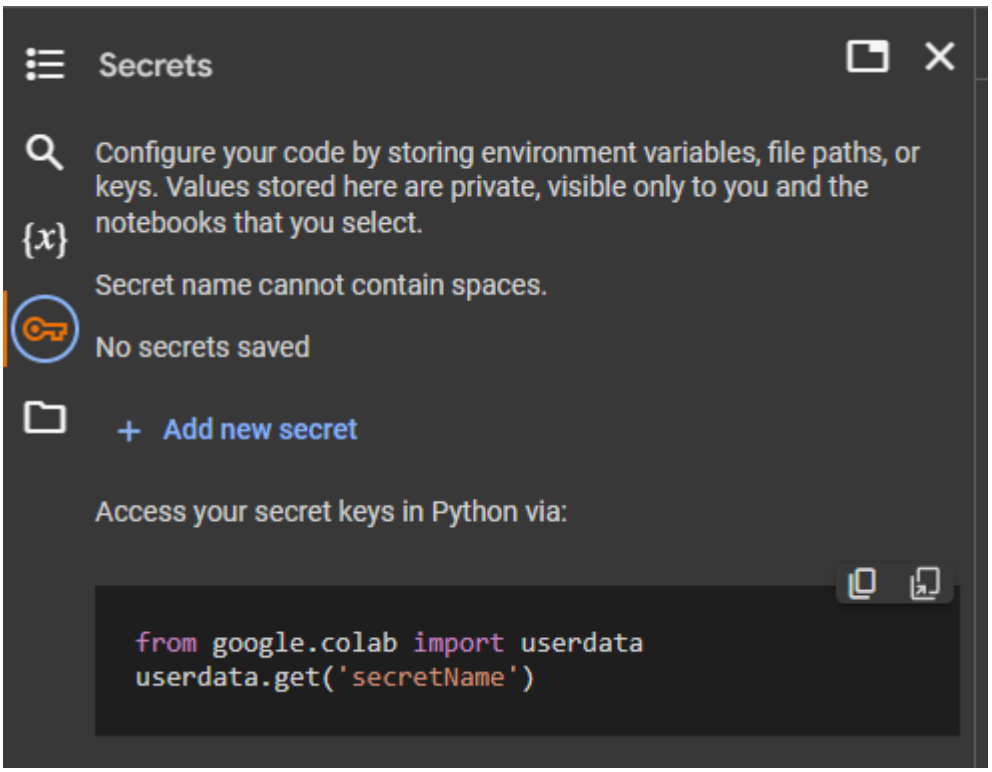

Рисунок 2.9 – Інструменти для роботи із проектом в Google Colab

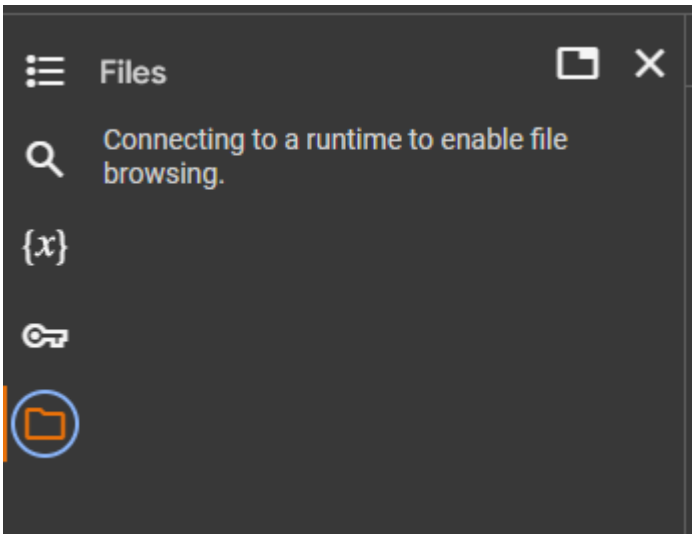

Рисунок 2.10 – Інструменти для роботи із проектом в Google Colab

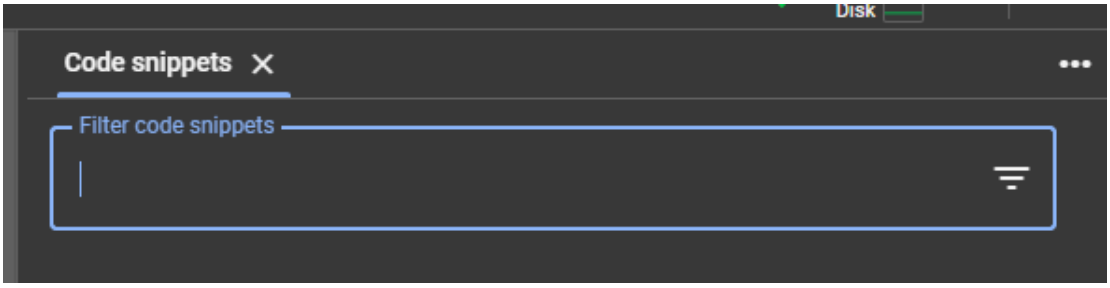

Рисунок 2.11 – Інструменти для роботи із проектом в Google Colab

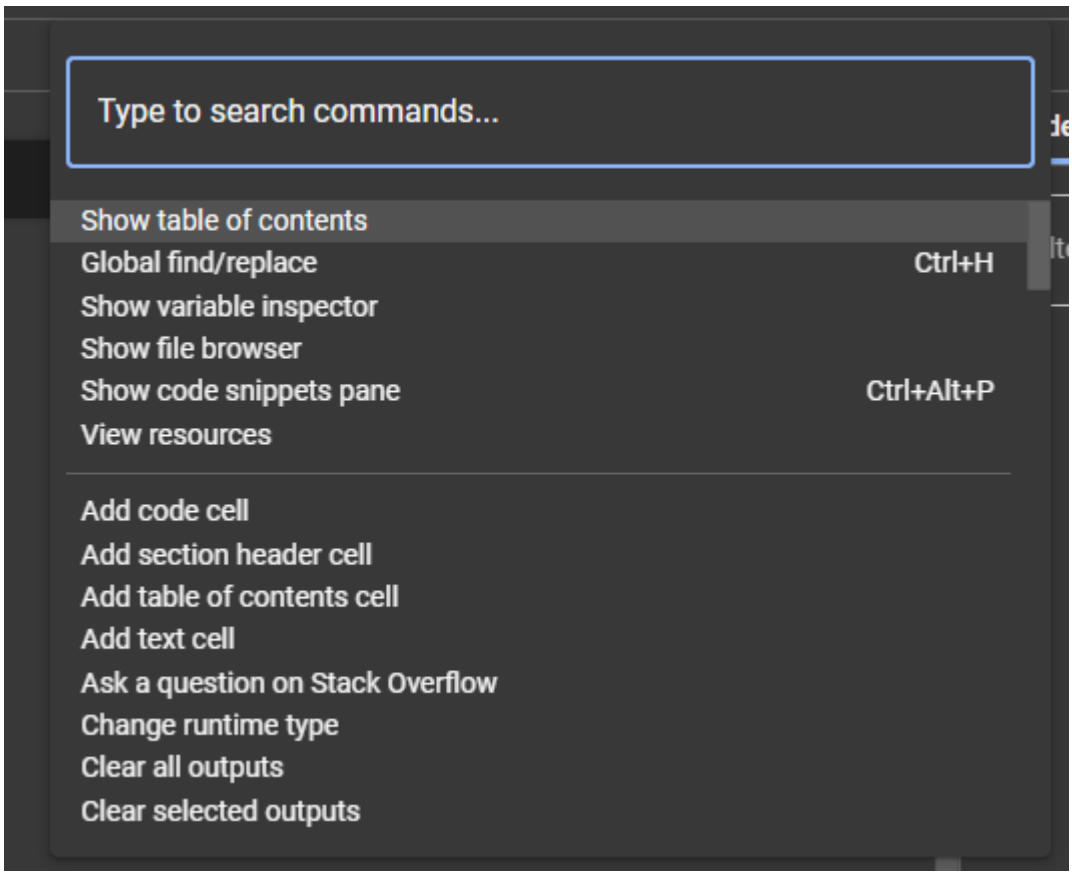

Рисунок 2.12 – Інструменти для роботи із проектом в Google Colab

Рисунок 2.13 – Інструменти для роботи із проектом в Google Colab

Текстовий опис усіх ключових етапів коду та функцій, що були написані й використані в програмному забезпеченні для видалення заднього фону на зображенні із портретами:

1. Клонування репозиторію, що містить необхідні бібліотеки для матування зображень (Рис. 2.14):

- переходимо до директорії `/content`;
- якщо репозиторій `modnet` ще не існує, клонуємо його з github за допомогою команди `git clone`;
- переходимо до директорії `modnet`.

```
import os

%cd /content

if not os.path.exists('MODNet'):
  !git clone https://github.com/ZHKKKe/MODNet

%cd MODNet/

pretrained_checkpoint = 'pretrained/modnet_photographic_portrait_matting.ckpt'
```

Рисунок 2.14 – Частина коду програмного забезпечення

2. Завантаження попередньо навчених ваг (Рис. 2.15):

- вказуємо шлях до файлу, необхідного для роботи програми `pretrained/modnet_photographic_portrait_matting.ckpt`;
- якщо файл не існує, завантажуємо його за допомогою утиліти `gdown`.

```
pretrained_checkpoint = 'pretrained/modnet_photographic_portrait_matting.ckpt'

if not os.path.exists(pretrained_checkpoint):
  !gdown --id 1mcr7ALciuAsHCpLnrtG_eop5-EYhbCmz -O pretrained/modnet_photographic_portrait_matting.ckpt

import shutil
from google.colab import files
```

Рисунок 2.15 – Частина коду програмного забезпечення

3. Очищення та створення директорій для зображень (Рис. 2.16):

- встановлюємо шляхи до директорій для вхідних зображень (`input`), вихідних зображень (`output`) та переднього плану зображень (`foreground`);
- якщо директорії існують, видаляємо їх та створюємо заново за допомогою `shutil`.

```
import shutil
from google.colab import files

input_directory = 'demo/image_matting/colab/input'

if os.path.exists(input_directory):
  shutil.rmtree(input_directory)
os.makedirs(input_directory)

output_directory = 'demo/image_matting/colab/output'

if os.path.exists(output_directory):
  shutil.rmtree(output_directory)
os.makedirs(output_directory)

foreground_directory = 'demo/image_matting/colab/foreground'

if os.path.exists(foreground_directory):
  shutil.rmtree(foreground_directory)
os.makedirs(foreground_directory)
```

Рисунок 2.16 – Частина коду програмного забезпечення

4. Завантаження зображень (Рис. 2.17):

- завантажуємо зображення з локального комп’ютера або іншого джерела за допомогою `files.upload()`;
- переміщуємо завантажені зображення до директорії `input`.

```
uploaded_images = list(files.upload().keys())
for image_name in uploaded_images:
  shutil.move(image_name, os.path.join(input_directory, image_name))
```

Рисунок 2.17 – Частина коду програмного забезпечення

5. Запуск моделі для інференсу (Рис. 2.18):

- виконуємо команду для запуску інференсу моделі, передаючи шляхи до вхідних зображень, вихідних зображень та файлу ваг.

```
!python -m demo.image_matting.colab.inference \
        --input-path demo/image_matting/colab/input \
        --output-path demo/image_matting/colab/output \
        --ckpt-path ./pretrained/modnet_photographic_portrait_matting.ckpt
```

Рисунок 2.18– Частина коду програмного забезпечення

6. Візуалізація та збереження переднього плану (Рис. 2.19):

- використовуємо бібліотеки `numpy` та `pil` для обробки зображень;
- функція `combined_display`: розраховує розмір для відображення зображення. Перетворює вхідне зображення та маттинг-зображення в масиви `numpy`. Розраховує передній план зображення на основі маттинг-зображення. Об'єднує вхідне зображення, передній план та альфа-канал в одну лінію для візуалізації. Змінює розмір об'єднаного зображення для зручного відображення;
- функція `save_foreground`: зберігає передній план зображення у вказану директорію.

```
def combined_display(image, matte):
  width, height = image.width, image.height
  display_width, display_height = 800, int(height * 800 / (3 * width))

  image = np.asarray(image)
  if len(image.shape) == 2:
    image = image[:, :, None]
  if image.shape[2] == 1:
    image = np.repeat(image, 3, axis=2)
  elif image.shape[2] == 4:
    image = image[:, :, 0:3]
```

Рисунок 2.19 – Частина коду програмного забезпечення

7. Обробка всіх зображень (Рис. 2.20):

для кожного завантаженого зображення:

- відкриваємо вхідне зображення та відповідне маттинг-зображення;
- отримуємо об'єднане зображення та передній план за допомогою функції `combined_display`;
- відображаємо об'єднане зображення;
- зберігаємо передній план за допомогою функції `save_foreground`.

```
matte = np.repeat(np.asarray(matte)[:, :, None], 3, axis=2) / 255
foreground = image * matte + np.full(image.shape, 255) * (1 - matte)

combined = np.concatenate((image, foreground, matte * 255), axis=1)
combined = Image.fromarray(np.uint8(combined)).resize((display_width, display_height))
return combined, foreground
```

Рисунок 2.20 – Частина коду програмного забезпечення

8. Архівування та завантаження результатів (Рис. 2.21):

- вказуємо ім'я файлу архіву `output.zip`;
- якщо файл архіву вже існує, видаляємо його;
- створюємо архів zip з переднім планом зображень за допомогою системної команди `zip`;
- завантажуємо архів за допомогою `files.download()`.

```
zip_filename = 'output.zip'

if os.path.exists(zip_filename):
  os.remove(zip_filename)

os.system(f"zip -r -j {zip_filename} {foreground_directory}/*")
files.download(zip_filename)
```

Рисунок 2.21 – Частина коду програмного забезпечення

Вихідний код основних компонентів наведено у Додатку А.

2.10 Робота програмного забезпечення

Після запуску коду програмного забезпечення, пропонується завантажити файл для обробки, як показано на Рис. 2.22-2.23.

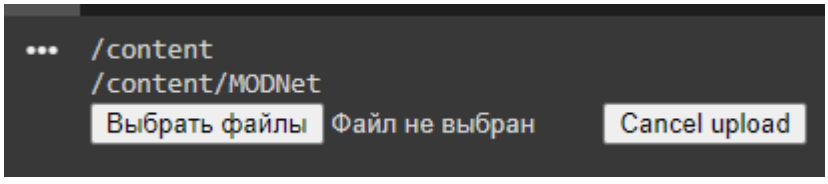

Рисунок 2.22 – Завантаження файлу для обробки

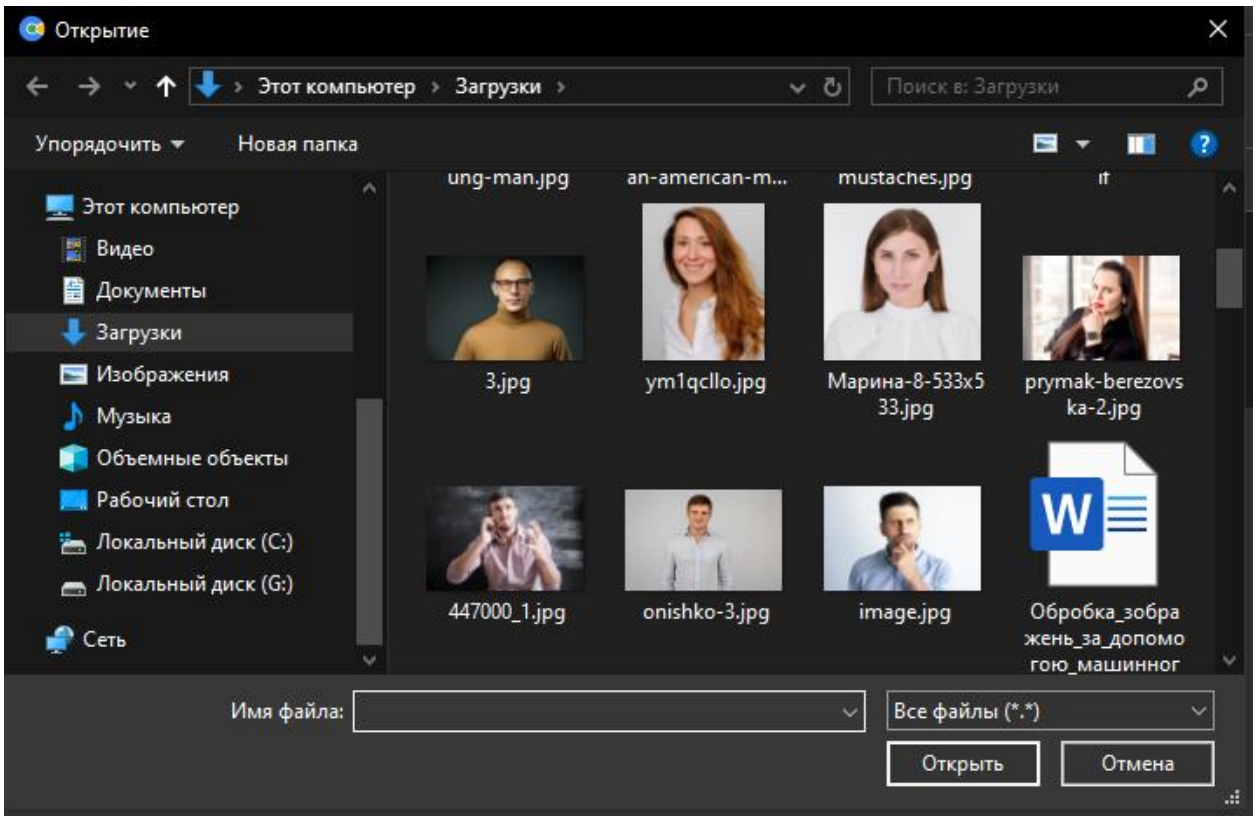

Рисунок 2.23 – Завантаження файлу для обробки

Далі програма починає обробку зображення, що зрозуміло із виводу в консоль, як показано на Рис. 2.24-2.25.

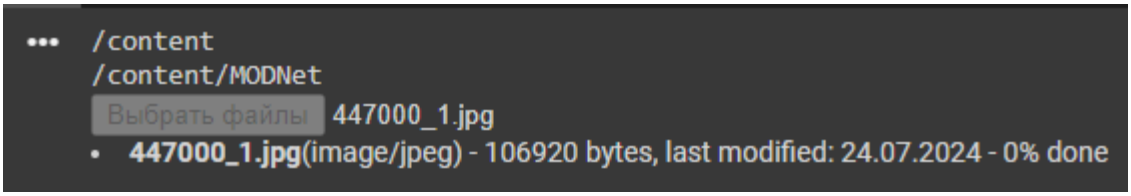

Рисунок 2.24 – Процес обробки

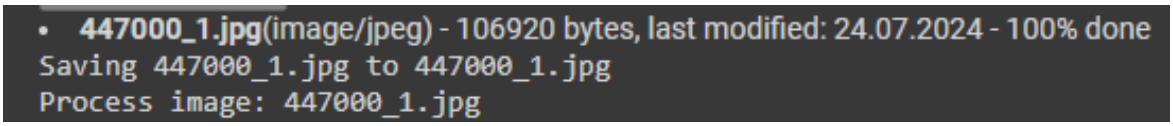

Рисунок 2.25 – Процес обробки

Коли обробка зображення завершена, програма виводить у консоль макет із трьох зображень: оригінальне зображення, виділена людина на білому фоні і силуети людини та фону для більш повного розуміння роботи програмного забезпечення. Цей процес показаний на Рис. 2.26.

Рисунок 2.26 – Вивід у консоль макет із трьох зображень

Після цього програма формує zip-архів, у який запаковує зображення із виділеною людиною на білому фоні й починає завантаження на жорсткий носій користувача, як показано на Рис. 2.27.

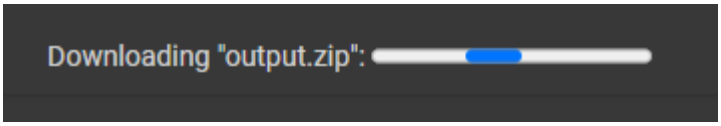

Рисунок 2.27 – Завантаження на жорсткий носій користувача архіва із обробленим зображенням виділеної людини.

Завантажений архів доступний через провідник і містить у собі зображення виділеної людини на білому фоні, як показано на Рис. 2.28-2.30.

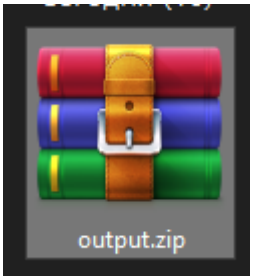

Рисунок 2.28 – Сформований програмою архів

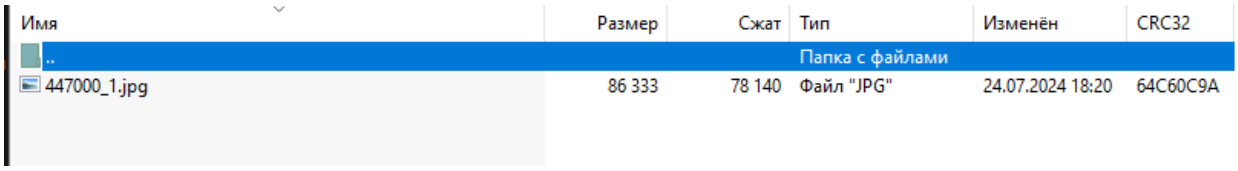

Рисунок 2.29 – Вміст сформованого програмою архіва

Рисунок 2.30 – Оброблене зображення

Оригінальне зображення подано на Рис. 2.31.

Рисунок 2.31 – Оригінальне зображення

Як можна зазначити, програмне забезпечення досить якісно впоралось із поставленим завдання виділення людини із зображення, навіть не зважаючи на факт, що на оригінальній фотографії присутні перешкоди у вигляді розмитої руки, що не у фокусі, а також диму, який місцями перекриває фігуру чоловіка.

Кілька результатів роботи програмного забезпечення подано на Рис. 2.32-2.41 у форматі макета із трьох зображень: оригінальне зображення, виділена людина на білому фоні (потрібний кінцевий результат, що пакується в zip-архів) і чорно-білого силуету людини та фону для більш повного розуміння роботи програмного забезпечення.

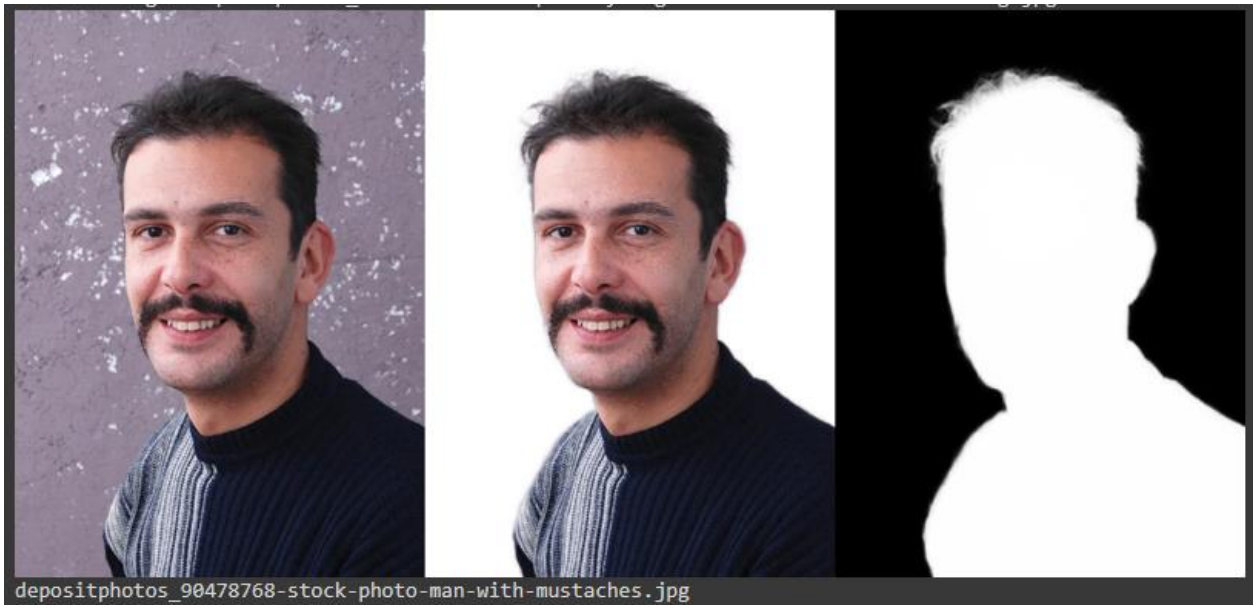

Рисунок 2.32 – Результат роботи програмного забезпечення

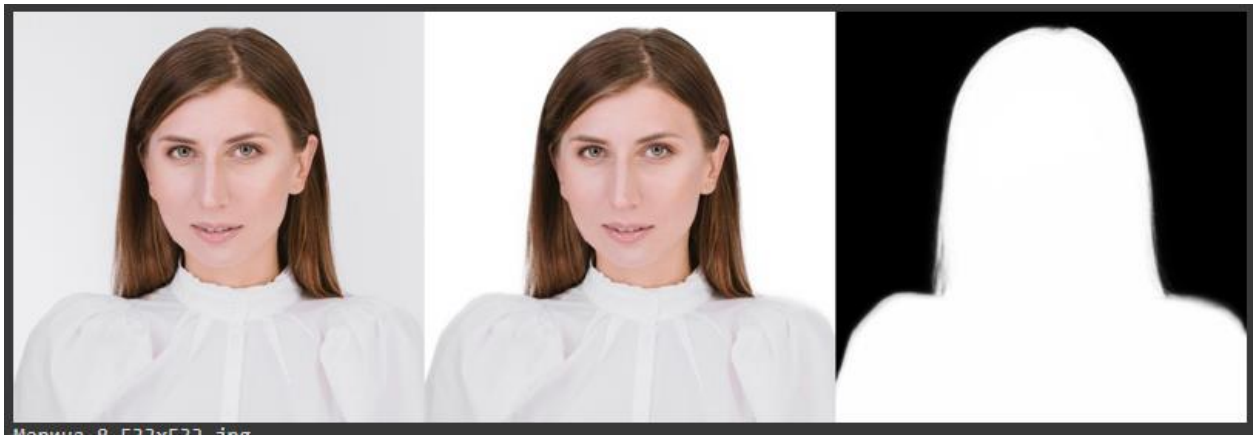

Рисунок 2.33 – Результат роботи програмного забезпечення

Рисунок 2.34 – Результат роботи програмного забезпечення

Рисунок 2.35 – Результат роботи програмного забезпечення

Рисунок 2.36– Результат роботи програмного забезпечення

Рисунок 2.37 – Результат роботи програмного забезпечення

Рисунок 2.38 – Результат роботи програмного забезпечення

Рисунок 2.39 – Результат роботи програмного забезпечення

Рисунок 2.40 – Результат роботи програмного забезпечення

Рисунок 2.41 – Результат роботи програмного забезпечення

Крім того, програма підтримує завантаження й обробку більше, ніж одного зображення, як показано на Рис. 2.42.

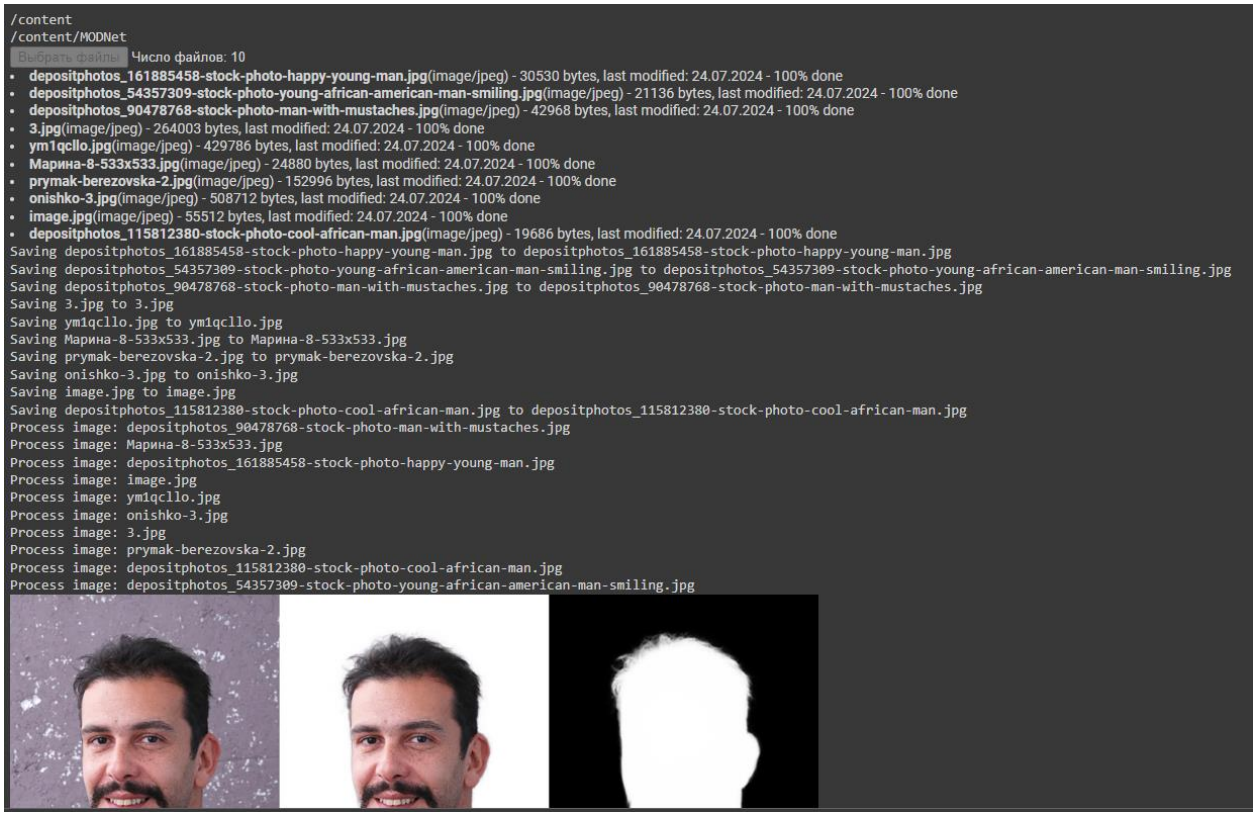

Рисунок 2.42 – Обробка кількох зображень

2.11 Розробка графічного інтерфейсу

Щоб додати графічний інтерфейс для розробленої програми для видалення заднього фону на зображенні із портретами, було обрано бібліотеку gradio, яка дозволяє інтеграцію в Google Colab, де й розроблялось ПЗ.

Gradio – це бібліотека Python, яка дозволяє швидко і просто створювати графічні інтерфейси для моделей машинного навчання, скриптів, або будь-якого іншого коду, що виконує обробку даних. Основні можливості Gradio:

1. Просте створення інтерфейсу. Gradio дозволяє легко створити інтерактивний графічний інтерфейс для будь-якої функції, що приймає вхідні дані і повертає результати. Це дозволяє користувачам взаємодіяти з моделями та іншими скриптами через простий веб-інтерфейс.

2. Типи вхідних та вихідних даних. Gradio підтримує різні типи вхідних і вихідних даних, включаючи зображення, текст, аудіо, відео і числові дані. Це дає змогу створювати інтерфейси для різних типів завдань.

3. Вбудовані компоненти. Gradio надає велику кількість вбудованих компонентів, таких як `gr.Image`, `gr.Textbox`, `gr.Audio`, `gr.Video`, `gr.Slider` та інші, для легкого створення інтерфейсів для різних форматів даних.

4. Швидкий запуск. Gradio дозволяє швидко запустити локальний веб-сервер для перегляду і тестування вашого інтерфейсу. Це корисно для швидкого прототипування та тестування.

5. Розгортання в Інтернеті. Готові інтерфейси можна легко розгорнути в Інтернеті для спільного використання. Gradio інтегрується з платформами для хостингу, такими як Hugging Face Spaces, для публікації інтерфейсів.

Gradio спрощує процес створення інтерфейсів для моделювання і надає зручні інструменти для тестування та демонстрації ваших моделей і скриптів [28].

Доданий інтерфейс для нашого програмного забезпечення має вигляд, що продемонстровано на Рис. 2.44-2.49.

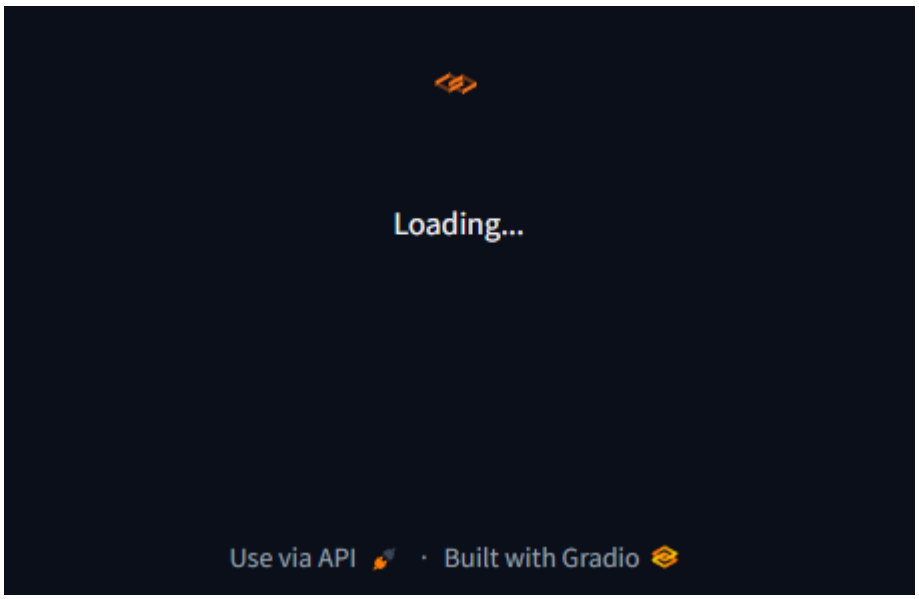

Рисунок 2.44 – Завантаження бібліотеки

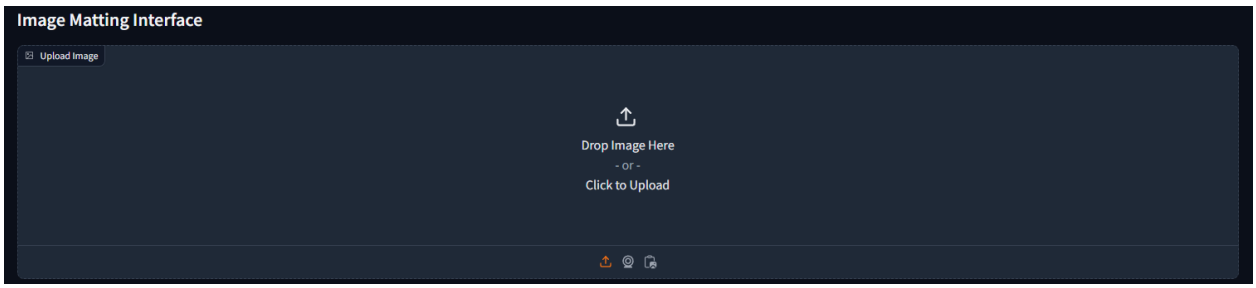

Рисунок 2.45 – Поле для завантаження вхідних файлів

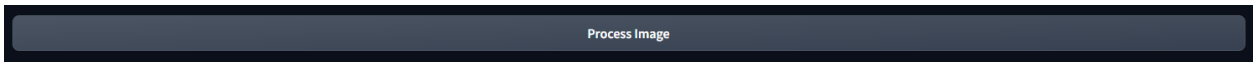

Рисунок 2.46 – Кнопка для обробки вхідних файлів

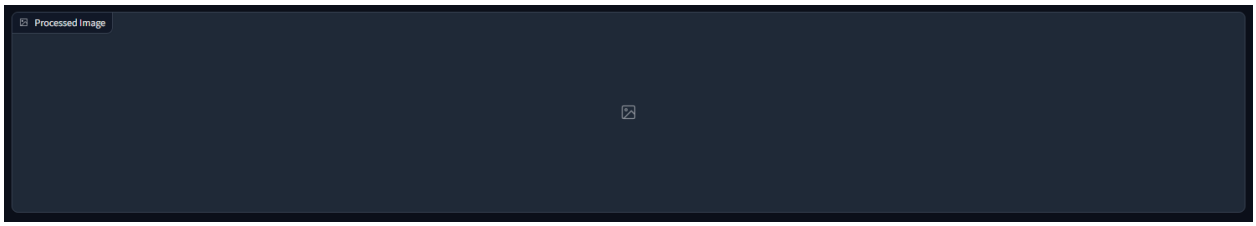

Рисунок 2.47 – Поле для оброблених файлів

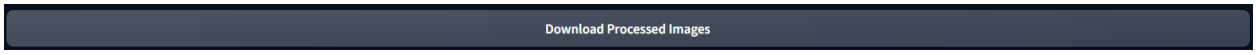

Рисунок 2.48 – Кнопка для збереження оброблених файлів

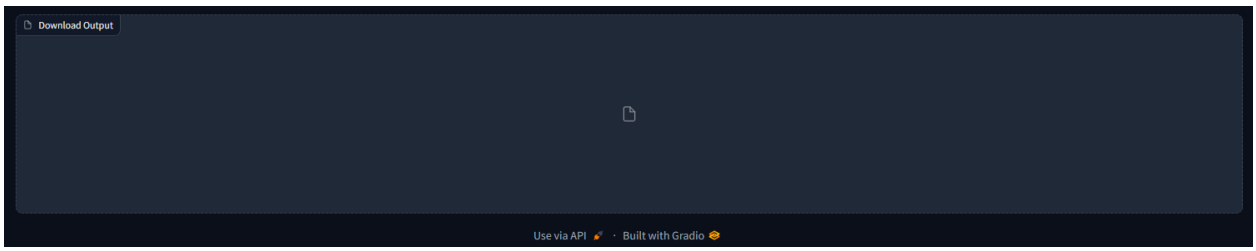

Рисунок 2.49 – Поле для збереження оброблених файлів

Вихідний код основних компонентів графічного інтерфейсу наведено у Додатку Б.

2.12 Тестування ПЗ

Тестування готового програмного забезпечення буде здійснюватися за допомогою тест-кейсів. Test Case (тестовий випадок) є інструментом для перевірки функціональності програмної системи шляхом виконання певних дій і/або умов. Структура тестового випадку зазвичай включає три основні компоненти:

1. Дія (Action) – конкретні кроки, які потрібно виконати.
2. Очікуваний результат (Expected Result) – результат, який очікується після виконання дій.
3. Фактичний результат (Test Result) – реальний результат, отриманий після виконання тесту.

Тестовий випадок складається з трьох частин:

- передумови (Preconditions) – це список кроків, які налаштовують систему до необхідного стану для проведення тесту, або перевірка того, що система вже перебуває в цьому стані;
- опис тестового випадку (Test Case Description) – опис дій, які слід виконати для перевірки функціональності, після чого порівнюються фактичний і очікуваний результати;
- постумови (Postconditions) – дії, які відновлюють систему до початкового стану після тестування.

Опис і структура тест-кейсів можуть варіюватися в залежності від компанії або команди, включаючи різну глибину опису дій і результатів, а також різні структурні елементи. Проте чітка структура і зручні шаблони тест-кейсів можуть значно скоротити час на рутинне заповнення форм і підвищити загальну ефективність команди.

Для тестування програмного забезпечення для видалення заднього фону на зображенні із портретами було створено основні тест-кейси, що наведені у вигляді Табл. 2.1-2.7.

Таблиця 2.1 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка завантаження невірного формату вхідного файлу для обробки програмою**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати невірний формат даних	Виділено обрані файли
3	Натиснути кнопку «ОК»	Відображено повідомлення про помилку

Таблиця 2.2 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка завантаження пустого вхідного файлу для обробки програмою**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Не обрати жодного файлу	Файли не виділено

3	Натиснути кнопку «ОК»	Відображено повідомлення про помилку

Таблиця 2.3 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка завантаження вхідного файлу для обробки програмою з кирилицею в назві**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати файл із кирилицею в назві	Файл виділено
3	Натиснути кнопку «ОК»	Програма завантажила обраний файл

Таблиця 2.4 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка швидкості обробки великого зображення**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати файл із роздільною здатністю більше, ніж 4000 на 4000 пікселів	Файл виділено
3	Натиснути кнопку «ОК»	Програма завантажила обраний файл і обробила не довше, ніж за 180 секунд

Таблиця 2.5 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка видалення фонового зображення з простим фоном**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати файл з однорідним фоном (наприклад, однотонний колір)	Файл виділено
3	Натиснути кнопку «ОК»	Програма завантажила обраний файл і обробила так, що задній фон зображення повністю видалено, залишився лише портрет людини.

Таблиця 2.6 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка видалення фонового зображення з складним фоном**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати файл з деталізованим фоном (наприклад, пейзаж або текстура).	Файл виділено
3	Натиснути кнопку «ОК»	Задній фон видалений без пошкодження портрету, фон очищено від деталей.

Таблиця 2.7 – Тест-кейс для розробленого ПЗ

Опис	**Перевірка точності видалення фону на зображеннях з неповними краями портрету**	
Передумови	Програма завантажена	
№	Дія	Очікуваний результат
1	Натиснути кнопку завантаження	Відкрилось вікно для вибору файлу для обробки
2	Обрати файл з частково непрозорими або нечіткими краями портрету.	Файл виділено
3	Натиснути кнопку «ОК»	Краї портрету залишаються чіткими, фон видалено без спотворення обрисів портрету.

Програмне забезпечення для видалення заднього фону на зображеннях із портретами пройшло всі тест-кейси успішно, підтверджуючи свою коректність у різних сценаріях використання. Програма ефективно видаляє фон з однорідних і складних фонових зображень, підтримує точність при обробці нечітких країв портретів і працює з великими зображеннями в прийнятний час. Результати тестування свідчать про високу якість та надійність програми.

ВИСНОВКИ

Підсумовуючи, у ході виконання даної магістерської дипломної роботи було досліджено автоматизовані підходи до видалення фону, що дозволяє значно покращити якість обробки зображень і зробити цей процес доступним для широкого кола користувачів.

Було розглянуто процес видалення заднього фону з портретних зображень, а також методи та технології машинного навчання та комп'ютерного бачення, що використовуються для видалення заднього фону з зображень.

У теоретичній частині були охоплені такі теми:

- основні поняття та визначення машинного навчання;
- базові моделі машинного навчання;
- приклади застосування машинного навчання;
- математичні основи машинного навчання;
- алгоритми машинного навчання;
- згорткова нейронна мережа;
- архітектури нейронних мереж для сегментації зображень;
- видалення фону із зображення, або матування, за допомогою машинного навчання.

У практичній же частині було розроблене програмне забезпечення, що досить якісно впоралось із поставленим завдання виділення людини із зображення, навіть не зважаючи на факт, що на деяких оригінальних фотографіях присутні перешкоди для роботи програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Machine Learning, Tom M. Mitchell, McGraw-Hill, Science/Engineering/Math, (March 1, 1997) ISBN: 0070428077

2. What is Machine Learning? Azmir Alam University of Dhaka, August 2023, DOI:10.5281/zenodo.8231580

3. Introduction to Machine Learning with Python A Guide for Data Scientists, Andreas C. Müller and Sarah Guido, 978-1-449-36941-5

4. An overview of the supervised machine learning methods, Vladimir Nasteski, DOI 10.20544/HORIZONS.B.04.1.17.P05, UDC 004.85.021:519.718

5. Building Cognitive Applications with IBM Watson Services: Volume 1 Getting Started, Dr. Alfio Gliozzo, Chris Ackerson, Rajib Bhattacharya, International Technical Support Organization, 2017

6. Automatic Speech Recognition, Levis, John & Suvorov, Ruslan, 2012, 10.1002/9781405198431.wbeal0066

7. Mathematical Foundations of Machine Learning, Seongjai Kim, Mississippi State, MS 39762 USA, 2024

8. L. KAUFMAN AND P. ROUSSEEUW, Clustering by means of medoids, in Statistical Data Analysis Based on the L 1 -Norm and Related Methods, Y. Dodge, ed., NorthHolland, 1987

9. D. G. LOWE, Object recognition from local scale-invariant features, in Proceedings of the Seventh IEEE International Conference on Computer Vision, IEEE, 1999

10. J. NOCEDAL AND S. J. WRIGHT, Numerical Optimization (2nd Ed.), Springer-Verlag, Berlin, New York, 2006

11. L. ELDÉN, Numerical linear algebra in data mining, Acta Numerica, 15 (2006)

12. Simple Linear Regression [Електронний ресурс] – Режим доступу до ресурсу: https://www.colorado.edu/amath/sites/default/files/attached-files/ch12_0.pdf

13. Understanding Machine Learning: From Theory to Algorithms, Shai Shalev-Shwartz and Shai Ben-David, 2014

14. Circuit complexity and neural networks, Parberry, I., The MIT press, 1994

15. A statistical study of on-line learning, Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK, Murata, N., 1998

16. Machine learning: a probabilistic perspective, Murphy, K. P., The MIT Press, 2012

17. On learning sets and functions, Natarajan, B. K., Mach. 1989

18. Introductory lectures on convex optimization: A basic course, Vol. 87, Nesterov, Y. & Nesterov, I., Springer Netherlands, 2004.

19. On convergence proofs on perceptrons, in ‘Proceedings of the Symposium on the Mathematical Theory of Automata’, Novikoff, A. B. J., Vol. XII, 1962

20. Image Segmentation: Architectures, Losses, Datasets, and Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://neptune.ai/blog/image-segmentation

21. How to Do Data Exploration for Image Segmentation and Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

22. Deep Learning-Based Automated Background Removal for Structural Exterior Image Stitching, Kang, Myung Soo & An, Yun-Kyu, Applied Sciences, 2021, 11. 3339. 10.3390/app11083339.

23. Introduction to Colab Enterprise [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/colab/docs/introduction

24. PyTorch vs TensorFlow in 2024: A Comparative Guide of AI Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://opencv.org/blog/pytorch-vs-tensorflow/

25. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

26. Activity Diagrams | Unified Modeling Language (UML) [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/

27. Class Diagrams vs Object Diagrams | Unified Modeling Language(UML) [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/class-diagrams-vs-object-diagrams-unified-modeling-languageuml/?ref=oin_asr1

28. How to Build Machine Learning Web Application Using Gradio on Ubuntu [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/how-to-build-machine-learning-web-application-using-gradio-on-ubuntu-22-04

ДОДАТОК А
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

Файл main.py

```
import os

# Клонуємо репозиторій
# Переходимо до директорії /content
%cd /content
# Якщо репозиторій MODNet не існує, клонуємо його
if not os.path.exists('MODNet'):
  !git clone https://github.com/ZHKKKe/MODNet
# Переходимо до директорії MODNet
%cd MODNet/

# Завантажуємо попередньо навчені ваги для матування зображень
pretrained_checkpoint = 'pretrained/modnet_photographic_portrait_matting.ckpt'
# Якщо файл ваг не існує, завантажуємо його за допомогою gdown
if not os.path.exists(pretrained_checkpoint):
  !gdown --id 1mcr7ALciuAsHCpLnrtG_eop5-EYhbCmz -O
pretrained/modnet_photographic_portrait_matting.ckpt

import shutil
from google.colab import files

# Очищаємо і створюємо заново папки для зображень
# Шлях до директорії з вхідними зображеннями
input_directory = 'demo/image_matting/colab/input'
# Якщо директорія існує, видаляємо її і створюємо заново
if os.path.exists(input_directory):
  shutil.rmtree(input_directory)
os.makedirs(input_directory)

# Шлях до директорії з вихідними зображеннями
output_directory = 'demo/image_matting/colab/output'
if os.path.exists(output_directory):
  shutil.rmtree(output_directory)
os.makedirs(output_directory)

# Шлях до директорії з переднім планом зображень
foreground_directory = 'demo/image_matting/colab/foreground'
if os.path.exists(foreground_directory):
  shutil.rmtree(foreground_directory)
os.makedirs(foreground_directory)

# Завантажуємо зображення (PNG або JPG)
# Завантажуємо файли і переміщуємо їх до директорії input
uploaded_images = list(files.upload().keys())
for image_name in uploaded_images:
```

```
  shutil.move(image_name, os.path.join(input_directory, image_name))

# Запускаємо модель для інференсу
# Виконуємо команду для інференсу, передаючи шляхи до вхідних і вихідних директорій і файлу ваг
!python -m demo.image_matting.colab.inference \
        --input-path demo/image_matting/colab/input \
        --output-path demo/image_matting/colab/output \
        --ckpt-path ./pretrained/modnet_photographic_portrait_matting.ckpt

import numpy as np
from PIL import Image

def combined_display(image, matte):
  # Розраховуємо роздільну здатність для відображення
  width, height = image.width, image.height
  display_width, display_height = 800, int(height * 800 / (3 * width))

  # Отримуємо передній план зображення
  # Перетворюємо зображення на масив numpy
  image = np.asarray(image)
  if len(image.shape) == 2:
    image = image[:, :, None]
  if image.shape[2] == 1:
    image = np.repeat(image, 3, axis=2)
  elif image.shape[2] == 4:
    image = image[:, :, 0:3]
  # Перетворюємо маттинг-зображення на масив numpy і повторюємо його для кожного каналу RGB
  matte = np.repeat(np.asarray(matte)[:, :, None], 3, axis=2) / 255
  # Розраховуємо передній план зображення
  foreground = image * matte + np.full(image.shape, 255) * (1 - matte)

  # Об'єднуємо зображення, передній план і альфа-канал в одну лінію
  combined = np.concatenate((image, foreground, matte * 255), axis=1)
  # Перетворюємо об'єднане зображення назад у зображення PIL і змінюємо його розмір для
відображення
  combined = Image.fromarray(np.uint8(combined)).resize((display_width, display_height))
  return combined, foreground

def save_foreground(image_name, foreground):
  # Зберігаємо передній план зображення
  foreground_image = Image.fromarray(np.uint8(foreground))
  foreground image.save(os.path.join(foreground directory, image name))

# Візуалізуємо всі зображення
for image name in os.listdir(input directory):
  matte_name = image_name.split('.')[0] + '.png'
  # Відкриваємо вихідне зображення і маттинг-зображення
  image = Image.open(os.path.join(input_directory, image_name))
  matte = Image.open(os.path.join(output_directory, matte_name))
  # Отримуємо об'єднане зображення і передній план
  combined, foreground = combined_display(image, matte)
```

```
  # Відображаємо об'єднане зображення
  display(combined)
  print(image_name, '\n')
  # Зберігаємо передній план
  save_foreground(image_name, foreground)

# Архівуємо і завантажуємо результати
# Ім'я файлу архіву
zip_filename = 'output.zip'
# Якщо файл архіву вже існує, видаляємо його
if os.path.exists(zip filename):
  os.remove(zip_filename)

# Створюємо архів zip з переднім планом зображень
os.system(f"zip -r -j {zip_filename} {foreground_directory}/*")
# Завантажуємо архів
files.download(zip_filename)
```

ДОДАТОК Б
ВИХІДНИЙ КОД ГРАФІЧНОГО ІНТЕРФЕЙСУ

Файл main.py

```
import os
import shutil
import numpy as np
from PIL import Image
import gradio as gr

# Шлях до директорії з вхідними зображеннями
input_directory = 'demo/image_matting/colab/input'
# Шлях до директорії з вихідними зображеннями
output_directory = 'demo/image_matting/colab/output'
# Шлях до директорії з переднім планом зображень
foreground_directory = 'demo/image_matting/colab/foreground'

def combined_display(image, matte):
    print("Об'єднання зображення і маттинг-зображення...")
    width, height = image.width, image.height
    display_width, display_height = 800, int(height * 800 / (3 * width))

    image = np.asarray(image)
    if len(image.shape) == 2:
        image = image[:, :, None]
    if image.shape[2] == 1:
        image = np.repeat(image, 3, axis=2)
    elif image.shape[2] == 4:
        image = image[:, :, 0:3]

    matte = np.repeat(np.asarray(matte)[:, :, None], 3, axis=2) / 255
    foreground = image * matte + np.full(image.shape, 255) * (1 - matte)

    combined = np.concatenate((image, foreground, matte * 255), axis=1)
    combined = Image.fromarray(np.uint8(combined)).resize((display_width, display_height))
    print("Об'єднання завершено.")
    return combined, foreground

def save_foreground(image_name, foreground):
    print(f"Збереження переднього плану для {image_name}...")
    foreground_image = Image.fromarray(np.uint8(foreground))
    foreground_image.save(os.path.join(foreground_directory, image_name))
    print("Збереження завершено.")

# Функція для обробки зображень
def process_image(image):
    print("Початок обробки зображення...")

    # Очищаємо і створюємо заново папки для зображень
    for directory in [input_directory, output_directory, foreground_directory]:
```

```
        if os.path.exists(directory):
            shutil.rmtree(directory)
        os.makedirs(directory)
    print("Папки очищені і створені заново.")

    # Зберігаємо вхідне зображення
    image_name = "input_image.png"
    image.save(os.path.join(input_directory, image_name))
    print(f"Зображення збережено у {input_directory}.")

    # Запускаємо модель для інференсу
    print("Запуск моделі для інференсу...")
    os.system(f"python -m demo.image_matting.colab.inference --input-path {input_directory} --
output-path {output_directory} --ckpt-path
./pretrained/modnet_photographic_portrait_matting.ckpt")
    print("Модель завершила роботу.")

    # Відкриваємо вихідне зображення і маттинг-зображення
    image = Image.open(os.path.join(input_directory, image_name))
    matte_path = os.path.join(output_directory, image_name.split('.')[0] + '.png')

    if not os.path.exists(matte_path):
        print(f"Помилка: файл маттинг-зображення не знайдено за шляхом {matte_path}")
        return None

    matte = Image.open(matte path)
    print("Зображення та маттинг-зображення відкриті.")

    # Отримуємо передній план зображення
    combined, foreground = combined_display(image, matte)

    # Зберігаємо передній план
    save foreground(image name, foreground)

    print("Обробка завершена.")
    return combined

def download foreground():
    zip_filename = 'output.zip'
    if os.path.exists(zip filename):
        os.remove(zip_filename)
    os.system(f"zip -r -j {zip filename} {foreground directory}/*")
    print("Файли запаковані у zip-архів.")
    return zip_filename

# Створюємо графічний інтерфейс за допомогою Gradio
with gr.Blocks() as demo:
    gr.Markdown("## Image Matting Interface")

    # Компонент для завантаження зображень
    input_image = gr.Image(label="Upload Image", type="pil")
```

```
    # Кнопка для обробки зображень
    process_btn = gr.Button("Process Image")

    # Поле для виводу обробленого зображення
    output_image = gr.Image(label="Processed Image")

    # Кнопка для завантаження оброблених зображень
    download_btn = gr.Button("Download Processed Images")

    # Зв'язуємо компоненти з функціями
    process_btn.click(process_image, inputs=input_image, outputs=output_image)
    download_btn.click(download_foreground, outputs=gr.File(label="Download Output"))

# Запускаємо інтерфейс
demo.launch()
```