МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Знаходження відмінностей при порівнянні зображень з дрону, знаходження потенційної цілі»

Виконав студент групи ЗІПЗм-22-1
КАРАПЕТ Богдан Васильович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
зав. кафедри інженерії програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: КАРАПЕТ Богдан Васильович

Керівник роботи: САВІЦЬКИЙ Роман Святославович

Тема роботи: «Знаходження відмінностей при порівнянні зображень з дрону, знаходження потенційної цілі», затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. №** 479**/с**

Вихідні дані для роботи: зображення з дронів, отримані з відкритих джерел; колекції зображень з ресурсу kaggle; методи порівняння, трансформації та аналізу зображень у комп'ютерному зорі.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1			
2			
3			

РЕФЕРАТ

Магістерська атестаційна робота, що представлена на розгляд, присвячена темі «Знаходження відмінностей при порівнянні зображень з дрону, знаходження потенційної цілі». Робота включає вступ, три основні розділи, висновки, список використаних джерел із 41 найменування, 28 рисунків та 12 лістингів. Загальний обсяг роботи складає 63 сторінки, з яких 57 припадає на основний текст.

Ціль роботи полягає у розробці методів порівняння та трансформації зображень з використанням інструментів Computer Vision та їх інтеграції в моделі машинного навчання. Для досягнення цієї мети було використано різноманітні методи обробки зображень, зокрема порівняння зображень, їх трансформація та виявлення об'єктів за допомогою машинного навчання.

Отримані результати включають розробку ефективних алгоритмів та підходів для аналізу та порівняння зображень, що можуть бути використані у різних областях, таких як військова, безпекова, сільськогосподарська тощо. Особливу увагу приділено використанню технік глибокого навчання та їх застосуванню у комп'ютерному зорі.

Структура роботи включає детальний огляд інструментів порівняння зображень, методів трансформації фото, та розробку моделей машинного навчання для виявлення об'єктів.

Ключові слова, що описують зміст роботи: Computer Vision, порівняння зображень, трансформація зображень, машинне навчання, YOLOv3, обробка зображень.

ABSTRACT

The master's thesis presented for consideration is dedicated to the topic «Finding differences when comparing images from a drone, identifying a potential target» The thesis comprises an introduction, three main sections, conclusions, a list of 41 references, 28 images, and 12 listings. The total volume of the thesis is 63 pages, of which 57 are allocated to the main text.

The aim of the thesis is to develop methods for image comparison and transformation using Computer Vision tools and integrate them into machine learning models. To achieve this goal, various image processing methods were employed, including image comparison, transformation, and object detection using machine learning.

The obtained results include the development of effective algorithms and approaches for image analysis and comparison, which can be applied in various fields such as military, security, agriculture, and more. Special attention is given to the use of deep learning techniques and their application in computer vision.

The structure of the thesis includes a detailed overview of image comparison tools, image transformation methods, and the development of machine learning models for object detection.

Keywords describing the content of the thesis: Computer Vision, image comparison, image transformation, machine learning, YOLOv3, image processing.

ЗМІСТ

РЕФЕРАТ 3

ЗМІСТ 5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6

ВСТУП 7

РОЗДІЛ 1. МЕТОДИ ТА ТЕХНОЛОГІЇ АНАЛІЗУ ЗОБРАЖЕНЬ У COMPUTER VISION 10

1.1 Наукова концепція та методологія Computer Vision 10

1.2 Завдання, що вирішуються за допомогою Computer Vision 12

1.3 Інтеграція методів Computer Vision в програмні продукти 13

1.4 Сучасні проблеми та перспективи розвитку Computer Vision 14

Висновки до розділу 1 15

РОЗДІЛ 2. ІНСТРУМЕНТИ ПОРІВНЯННЯ, ТРАНСФОРМАЦІЇ ЗОБРАЖЕНЬ. МОДЕЛІ ML 16

2.1 Інструменти порівняння, computer vision 16

2.2. Трансформація зображено з подальшим порівнянням 22

2.3. Пошук об’єктів за допомогою моделей ML 31

Висновки до розділу 2 34

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПОРІВНЯННЯ, ТРАНСФОРМАЦІЇ ЗОБРАЖЕНЬ. МОДЕЛЕЙ ML 35

3.1 Попіксельне порівняння за допомогою Computer Vision 35

3.2 Трансформація зображень з послідуючим порівнянням 42

3.3 Пошук об’єктів за допомогою моделі ML 51

Висновки до розділу 3 56

ВИСНОВКИ 57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CV – Computer Vision, комп’ютерний зір

PIL – Python Imaging Library

OpenCV – Open Source Computer Vision Library

YOLO – You Only Look Once

SSD – Single Shot MultiBox Detector

R-CNN – Region-based Convolutional Neural Networks

МРТ – Магнітно-резонансна томографія

ВСТУП

Актуальність роботи полягає у аналізі інструментів порівняння та трансформації зображень у Computer Vision та їх застосуванню у моделях машинного навчання, а також у вирішенні низки теоретичних та практичних проблем, що стоять перед сучасним науковим співтовариством у сфері комп'ютерного зору. Зокрема, важливим є розвиток ефективних алгоритмів для обробки та аналізу зображень, що має значне застосування у багатьох галузях.

Важливість дослідження також підкреслюється аналізом робіт науковців у цій сфері. Ряд дослідників зосереджують свою увагу на вдосконаленні алгоритмів глибокого навчання для аналізу зображень, зокрема на розвитку конволюційних нейронних мереж. Інші фокусуються на практичному застосуванні цих методів у реальних умовах, виявляючи та вирішуючи практичні проблеми.

Мета роботи полягає у розробці та впровадженні методів порівняння та трансформації зображень у сфері Computer Vision, а також їх інтеграції у моделі машинного навчання для підвищення ефективності аналізу візуальних даних у різних сферах застосування.

Завдання роботи. На основі мети роботи завдання можна сформулювати таким чином:

1. Розробка методів порівняння зображень: розробити інструменти для порівняння зображень, які дозволяють ефективно виявляти відмінності, використовуючи алгоритми, такі як cv2.absdiff і модуль PIL.
2. Трансформація зображень: впровадити методи трансформації зображень, включаючи трансформацію перспективи та порогову зміну, для покращення якості та точності аналізу зображень.
3. Інтеграція моделей машинного навчання: реалізувати та інтегрувати моделі машинного навчання, такі як YOLOv3, для виявлення та класифікації об'єктів на зображеннях.

4. Аналіз та оптимізація розроблених методів: провести аналіз та оптимізацію розроблених методів та інструментів для забезпечення їх високої ефективності та точності.

Поставлені завдання спрямовані на досягнення основної мети роботи – розробки та впровадження ефективних методів порівняння та трансформації зображень в області комп'ютерного зору і їх застосування у моделях машинного навчання.

Об'єктом дослідження у магістерській роботі є процеси та явища, пов'язані з аналізом та обробкою зображень у сфері комп'ютерного зору, зокрема використання інструментів порівняння зображень, методів трансформації фотографій, а також впровадження моделей машинного навчання для виявлення та класифікації об'єктів на зображеннях.

Предмет дослідження роботи включає розробку і впровадження алгоритмів та методів для порівняння, трансформації та аналізу зображень в рамках комп'ютерного зору, а також їх інтеграцію в моделі машинного навчання.

Методи дослідження у магістерській роботі включають: аналіз комп'ютерного зору, програмування та алгоритмічний аналіз, використання бібліотек обробки зображень, машинне навчання.

Інформаційну базу дослідження магістерської роботи складають наукові публікації, статті, та дослідження у галузі комп'ютерного зору та машинного навчання, опубліковані у періодичних та спеціалізованих виданнях.

Наукова новизна роботи полягає у розвитку та оптимізації моделей та алгоритмів для аналізу та обробки зображень комп'ютерним зором. Це включає модифікацію та удосконалення існуючих методів порівняння та трансформації зображень, а також розвиток нових підходів у використанні моделей машинного навчання. Результати дослідження характеризуються інноваційністю, оскільки вони вносять вклад у розробку нових технічних рішень, що дозволяють значно покращити процеси аналізу зображень, збільшуючи їхню точність та ефективність.

Практичне значення отриманих результатів магістерської роботи полягає у реалізації теоретичних положень у формі програмного продукту, який здатен ефективно вирішувати задачі порівняння, трансформації та аналізу зображень. Розроблені моделі та алгоритми можуть знайти широке застосування у різних сферах, зокрема у сфері безпеки, медицини, автоматизованого керування, розпізнавання об'єктів та інших областях, де важливий швидкий та точний аналіз візуальної інформації.

Публікації. За темою атестаційної магістерської роботи було опубліковано тези [1].

РОЗДІЛ 1. МЕТОДИ ТА ТЕХНОЛОГІЇ АНАЛІЗУ ЗОБРАЖЕНЬ У COMPUTER VISION

1.1 Наукова концепція та методологія Computer Vision

Computer Vision, або комп'ютерний зір, є однією з найбільш динамічних областей в комп'ютерних науках та інженерії. Його коріння можна простежити з 1960-х років, коли вчені почали розробляти алгоритми для інтерпретації та розуміння візуальної інформації. Перші експерименти в цій області були простими та обмеженими, зосереджуючись на базовому розпізнаванні форм та об'єктів.

З розвитком технологій, зокрема підвищенням обчислювальної потужності та поліпшенням алгоритмів, комп'ютерний зір значно еволюціонував. У 1980-х та 1990-х роках виникли ключові підходи, такі як обробка зображень, розпізнавання об'єктів, та стереозору. З появою нейронних мереж та глибокого навчання у 2000-х роках, комп'ютерний зір зробив суттєвий прорив, дозволяючи розробникам створювати системи, здатні точно розпізнавати та класифікувати об'єкти на зображеннях та відео [2].

Методи аналізу зображень еволюціонували від простих технік обробки зображень до складних нейронних мереж. Початкові методи включали фільтрацію та підвищення якості зображень, детекцію країв, та просте розпізнавання форм. Згодом, з розвитком алгоритмів машинного навчання, були введені методи для класифікації та локалізації об'єктів, що дозволило точніше ідентифікувати та аналізувати візуальні дані.

Computer Vision відіграє ключову роль у багатьох сучасних технологіях. Від автоматизованого контролю якості на виробництві до розпізнавання облич у смартфонах, від систем безпеки до автономних транспортних засобів, комп'ютерний зір забезпечує необхідну технологічну підтримку. Ця область стала невід'ємною частиною багатьох промислових, наукових, та побутових застосувань, зокрема в медицині, рекламі, розвагах та освіті [3].

Розвиток комп'ютерного зору тісно пов'язаний з розвитком програмних бібліотек та фреймворків, які полегшують розробку та імплементацію алгоритмів аналізу зображень:

OpenCV (Open Source Computer Vision Library): Це одна з найвідоміших бібліотек для обробки зображень та комп'ютерного зору. OpenCV підтримує мови програмування C++, Python, Java та інші, і використовується для розробки реальних застосунків в різних галузях [3].

Pillow (PIL, Python Imaging Library): це бібліотека для обробки зображень у Python. Вона забезпечує широкий спектр інструментів для редагування, обробки та представлення графічних даних [4].

Dlib: це сучасна бібліотека машинного навчання, яка включає інструменти для розпізнавання облич, обробки зображень та оптимізації алгоритмів.

TensorFlow та Keras: TensorFlow, розроблений Google, є одним з найпопулярніших фреймворків для глибокого навчання. Keras, який часто використовується разом з TensorFlow, є високорівневим API, який полегшує розробку та експериментування з нейронними мережами.

YOLO (You Only Look Once): це система реального часу для детекції об'єктів, яка відома своєю швидкістю та точністю. YOLO аналізує зображення в один прохід, швидко ідентифікуючи та класифікуючи об'єкти [5].

SSD (Single Shot MultiBox Detector): це метод детекції об'єктів, який може використовувати один раз із зображенням для виявлення об'єктів.

Faster R-CNN (Region-based Convolutional Neural Networks): це потужний метод для точного виявлення та класифікації об'єктів на зображеннях, що використовує конволюційні нейронні мережі.

Ці інструменти і фреймворки відіграють важливу роль у розвитку і застосуванні комп'ютерного зору, дозволяючи розробникам створювати все більш складні та ефективні системи для аналізу зображень.

1.2 Завдання, що вирішуються за допомогою Computer Vision

Комп'ютерний зір застосовується у низці різноманітних сфер, від безпеки та реклами до автоматизованого водіння та медичної діагностики. У сфері безпеки, комп'ютерний зір інтегрований у системи відеоспостереження для ідентифікації підозрілих дій, виявлення небезпечних предметів, а також розпізнавання облич. Це також ключовий компонент в системах контролю доступу, де використовується біометричне розпізнавання.

У рекламній індустрії, комп'ютерний зір використовується для аналізу візуальних даних, отриманих з камер, дозволяючи рекламодавцям оптимізувати свої рекламні матеріали на основі вікових груп та статі цільової аудиторії. Інтерактивні рекламні щити, які використовують комп'ютерний зір, забезпечують персоналізацію та інтерактивність рекламних повідомлень.

У секторі автоматизованого водіння, комп'ютерний зір стає необхідністю для безпечного водіння. Від систем допомоги водієві, які виявляють дорожні знаки, пішоходів, та інших транспортних засобів, до розвитку повністю автономних транспортних засобів, комп'ютерний зір відіграє центральну роль.

У медичній сфері, комп'ютерний зір застосовується для аналізу медичних зображень, таких як МРТ, рентгенівські знімки, допомагаючи в діагностиці захворювань. Також, ця технологія використовується у хірургічних процедурах для точного керування хірургічними інструментами.

Окремо слід згадати про аналіз зображень, отриманих з дронів. Тут комп'ютерний зір використовується для моніторингу змін у місцевості, виявлення змін у ландшафті, ідентифікації об'єктів та визначення потенційних цілей. Це може бути особливо корисним у військовій справі, сільському господарстві для моніторингу урожаю, будівництві для нагляду за прогресом проєктів, а також у безпеці та рятувальних операціях [6].

Таким чином, комп'ютерний зір є ключовою технологією, що знаходить застосування у багатьох сферах сучасного життя, від повсякденної діяльності до високоспеціалізованих галузей.

1.3 Інтеграція методів Computer Vision в програмні продукти

Інтеграція методів комп'ютерного зору в програмні продукти є важливою частиною сучасних технологічних розробок. Використання алгоритмів комп'ютерного зору в програмному забезпеченні дозволяє розширити його функціональність та забезпечити додаткову цінність для користувачів.

Перший крок у цьому процесі полягає у виборі та інтеграції відповідних бібліотек та фреймворків, які забезпечують необхідні інструменти для розробки. Наприклад, бібліотека OpenCV використовується для різноманітних задач обробки зображень та відео, включаючи детекцію об'єктів, відстеження руху та аналіз облич [7]. TensorFlow та Keras є популярним вибором для реалізації складних моделей глибокого навчання, що потрібні для більш складних задач аналізу зображень, таких як класифікація зображень або виявлення об'єктів [8].

Інтеграція цих інструментів вимагає глибокого розуміння як самої технології комп'ютерного зору, так і особливостей програмування. Розробники повинні не тільки вміти використовувати ці інструменти, але й знати, як найкраще їх застосувати для досягнення цілей конкретного проекту.

Що стосується прикладів успішних програмних рішень, що використовують комп'ютерний зір, то їх досить багато. Наприклад, системи розпізнавання облич, які використовуються для безпеки пристроїв та аутентифікації користувачів. Інший приклад - мобільні додатки для сканування та перекладу тексту, які використовують алгоритми комп'ютерного зору для виявлення тексту на зображенні та його подальшої обробки.

В галузі здоров'я, програмні продукти, що використовують комп'ютерний зір, здатні аналізувати медичні зображення, допомагаючи лікарям у діагностиці захворювань. У роздрібній торгівлі та складській логістиці системи на основі комп'ютерного зору використовуються для відстеження товарів, управління запасами та автоматизації процесів вибору та пакування.

Загалом, інтеграція комп'ютерного зору в програмні продукти відкриває нові можливості для підвищення ефективності, точності та інноваційності в широкому спектрі галузей. Використання цієї технології дозволяє створювати рішення, які раніше були неможливі або вимагали значних зусиль та ресурсів

1.4 Сучасні проблеми та перспективи розвитку Computer Vision

Комп'ютерний зір – галузь технології, яка стрімко розвивається. Однак, разом з швидким розвитком, ця галузь стикається з рядом викликів, які потребують рішення. Однією з основних проблем є підвищення точності аналізу зображень. Незважаючи на значні досягнення у цій області, існують ситуації, в яких системи комп'ютерного зору все ще можуть допускати помилки, особливо в складних умовах освітлення або при виявленні дуже дрібних деталей.

Ще одним важливим викликом є швидкість обробки зображень. Для багатьох застосувань, особливо тих, які вимагають реального часу, таких як автоматизоване водіння або системи відеонагляду, швидкість обробки є критично важливою. Вдосконалення алгоритмів та оптимізація обчислювальних ресурсів є ключовими аспектами вирішення цієї проблеми [9].

Обробка великих обсягів даних також є значною проблемою. З розвитком цифрових технологій кількість генерованих зображень та відео зростає експоненційно. Ефективне управління, зберігання та аналіз цих даних вимагають нових підходів та рішень.

Що стосується майбутніх напрямів розвитку в області комп'ютерного зору, одним з найбільш обнадійливих є глибоке навчання. Нейронні мережі та алгоритми машинного навчання вже довели свою ефективність в різних задачах аналізу зображень, і їхня роль лише зростатиме. Розвиток глибоких нейронних мереж дозволить створювати ще більш точні та надійні системи комп'ютерного зору [10].

Поєднання комп'ютерного зору з іншими технологіями, такими як штучний інтелект, Інтернет речей, та великі дані, також відкриває нові

перспективи. Це може призвести до створення інтелектуальних систем, які будуть здатні не тільки аналізувати зображення, але й робити складні висновки, приймати рішення та навіть передбачати майбутні події.

В цілому, хоча існують значні виклики в області комп'ютерного зору, перспективи розвитку цієї технології залишаються надзвичайно обнадійливими. Продовжуючи розвиток та інтеграцію нових технологічних рішень, можна очікувати значного прогресу у точності, швидкості обробки та здатності систем комп'ютерного зору ефективно впоратися з великими обсягами даних.

Висновки до розділу 1

У цьому розділі було докладно розглянуто різні інструменти та методи, які використовуються для порівняння зображень у сфері комп'ютерного зору. Основною метою розділу є огляд і аналіз наявних підходів до порівняння візуальних даних, які є важливими у великій кількості практичних застосувань, включаючи військові, медичні, та сферу безпеки.

Перш за все, було розглянуто основні методи попіксельного порівняння зображень, які базуються на аналізі кожного пікселя та порівнянні його значення між двома зображеннями.

Далі ми детально розглянули використання різних бібліотек та алгоритмів, таких як OpenCV та бібліотека Python Imaging Library (PIL), для порівняння та обробки зображень.

Додатково, проаналізовано процес накладання та трансформації зображень, оскільки це часто використовується для покращення якості та аналізу візуальних даних.

Отже, цей розділ є основою для розуміння різних аспектів порівняння та аналізу зображень у сфері комп'ютерного зору. Він створює фундамент для подальших досліджень та розвитку методів, які використовуються в подальших розділах.

РОЗДІЛ 2. ІНСТРУМЕНТИ ПОРІВНЯННЯ, ТРАНСФОРМАЦІЇ ЗОБРАЖЕНЬ. МОДЕЛІ ML

2.1 Інструменти порівняння, computer vision

Попіксельне порівняння за допомогою cv2.absdiff з підсвічуванням різниці

Попіксельне порівняння зображень - це метод, що дозволяє виявляти різницю між двома зображеннями на найдрібнішому рівні. Однією з популярних бібліотек для реалізації такого порівняння є OpenCV (або cv2 у Python). Функція cv2.absdiff() цієї бібліотеки дає змогу швидко отримати зображення різниці між двома іншими зображеннями.

OpenCV (Open Source Computer Vision Library) — бібліотека комп’ютерного зору та машинного навчання з відкритим кодом. OpenCV було створено, щоб забезпечити загальну інфраструктуру для програм комп’ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах. Будучи ліцензованим продуктом Apache 2, OpenCV дозволяє компаніям легко використовувати та змінювати код [11].

Функція cv2.absdiff() приймає на вхід два зображення і повертає нове зображення, яке представляє абсолютну різницю між ними. Для кожного пікселя вхідних зображень відбувається віднімання відповідних пікселів, результат якого потім береться за модулем, формуючи піксель вихідного зображення [12].

Щоб зробити різницю більш помітною на отриманому зображенні, можна застосувати певні методи підсвічування:

- бінаризація: після отримання зображення різниці можна використовувати порогову обробку (cv2.threshold()), щоб виділити змінені області;

- колірна мапа: для підсвічування можна використовувати колірну мапу (cv2.applyColorMap()), що дозволить зробити різницю більш помітною завдяки контрастним кольорам;
- дилатація: щоб зробити різницю ширшою і більш помітною, можна застосувати дилатацію (cv2.dilate()).

Нижче наведено порівняння оригінального фото та фото, в яке штучно був доданий об’єкт (підсвічений зеленим).



Рис 2.1 Знаходження різниці у фотографіях за допомогою cv2.absdiff() без зміни місця зйомки

Рисунок 2.1 показує хороший результат у порівнянні, якщо фотографія була зроблена з одного місця, під тим самим кутом та за аналогічних умов.

Отже, попіксельне порівняння за допомогою cv2.absdiff() є ефективним способом виявлення різниці між двома зображеннями, у разі, коли умови зйомки були майже 100% подібні (стаціонарні). Коли фотографії були зроблені під різними кутами, зі зміною місця зйомки або зі зміненими умовами, результат буде некоректний.

Накладання двох BGR2GRAY фотографій

Накладання фотографій, чи то в колірному, чи в монохромному форматі, може створити візуальний ефект, який може використовуватися для аналізу, демонстрації змін або створення художніх композицій. Переведення зображень з формату BGR (Blue-Green-Red) до формату GRAY (монохромного) допомагає зосередитися на яскравості та контрасті, відкидаючи колірну інформацію.

Коли ми говоримо про накладання двох зображень, ми маємо на увазі комбіноване зображення, де одне зображення накладається на інше. Це може бути зроблено з використанням різних методів злиття, таких як просте додавання, взяття середнього значення, множення тощо [13].

OpenCV – це потужна бібліотека комп'ютерного зору, яка має ряд інструментів для роботи з зображеннями. Для накладання двох монохромних (GRAY) зображень можна використовувати такі етапи:

1. Завантаження зображень і переведення їх у монохромний формат [14].
2. Накладання зображень. Один із способів - використання середньої квадратичної помилки.
3. Відображення або збереження результату.

Накладання монохромних зображень може використовуватися в ряді сценаріїв:

- аналіз різниці: якщо у вас є дві подібні фотографії, зроблені в різний час, накладання може допомогти виявити зміни між ними;
- художнє вираження: накладання зображень може створити цікаві візуальні ефекти, що викликають різноманітні емоції та асоціації;

- злиття інформації: наприклад, якщо є топографічна карта та карта дощів, їх можна комбінувати для отримання зображення, яке показує обидві інформації разом.

Виконаємо порівняння двох BGR2GRAY фотографій на прикладі даних з минулого розділу та схожих даних.

Наступний рисунок (рис. 2.2) відображає порівняння двох фотографій, на одну з яких було штучно додано об’єкт.

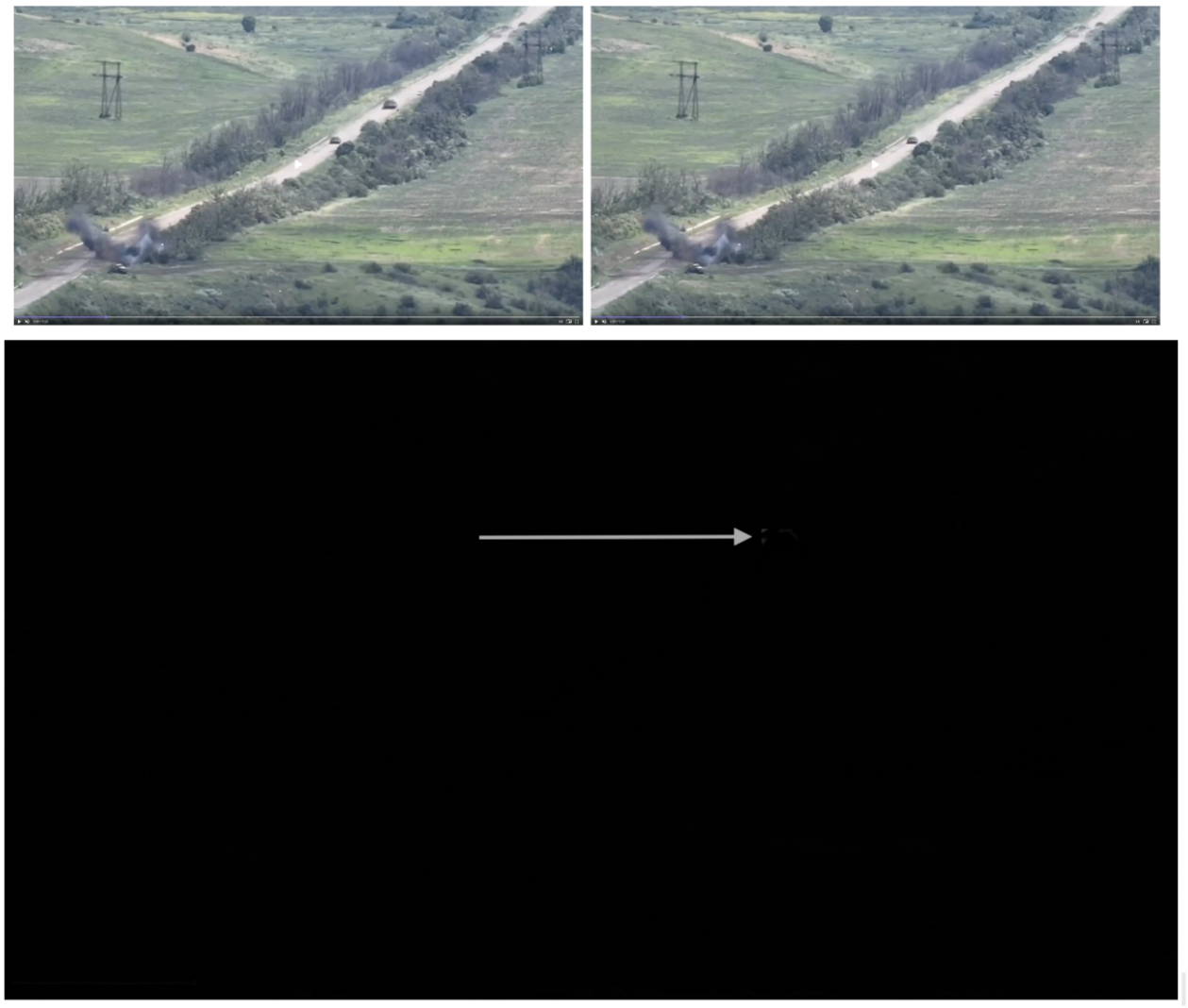


Рис 2.2 Порівняння двох BGR2GRAY фотографій зі штучно доданим об’єктом

Як ми можемо побачити, порівняння двох BGR2GRAY фотографій, які були зроблені стаціонарно, способом накладання, показали гірший ефект, ніж порівняння за допомогою cv2.absdiff().

Розглянемо приклад даних, які у випадку з cv2.absdiff() показали неточний результат. Тобто, чи можемо ми за допомогою приведення до монохромного елімінувати вплив зміни кута зйомки на порівняння (Рис 2.3).



Рис 2.3 Порівняння двох BGR2GRAY фотографій зі зміною кута зйомки

Як ми можемо побачити, приведення до монохромного формату прибрало вплив зміни кута зйомки і, до певної міри, прибере вплив освітлення та погодних умов на час зйомки.

Отже, накладання монохромних зображень допомагає комбінувати, аналізувати та створювати нові візуальні композиції. З використанням таких бібліотек, як OpenCV, процес стає простішим і легкодоступним.

Порівняння за допомогою модулю PIL

PIL (Python Imaging Library) є відомою бібліотекою для обробки зображень у Python. Вона надає засоби для завантаження, обробки та

збереження різних форматів зображень. Однією з її ключових можливостей є здатність порівнювати зображення на різних рівнях (Рис 2.4) [15].



Рис 2.4 Порівняння двох фотографій зі штучно доданим об’єктом за допомогою бібліотеки PIL

Основні методи порівняння:

- піксель-до-пікселя: цей простий метод порівняння вимагає перевірки кожного пікселя зображення на відповідність пікселю іншого зображення. Хоча цей метод є прямолінійним, він може бути не завжди ефективним, оскільки незначні зміни у зображенні можуть призвести до великої кількості різниці пікселів;

- гістограми: порівняння зображень за допомогою їх гістограм дозволяє оцінити розподіл яскравості та кольорових компонентів у зображенні. Це може бути корисно, коли потрібно знайти загальні відмінності між двома зображеннями, а не конкретні піксельні розбіжності;

- форма та особливості: замість прямого порівняння пікселів або гістограм, можна використовувати більш складні алгоритми, що виявляють

форми або особливості на зображенні, а потім порівнюють ці особливості між зображеннями.

PIL добре підходить для простого порівняння зображень. Наприклад, воно може бути корисним для виявлення дублікатів або перевірки наявності змін на веб-сайті.

Однак для більш складних завдань, таких як порівняння облич, рекомендується використовувати спеціалізовані бібліотеки, такі як OpenCV.

PIL є потужним інструментом для обробки зображень у Python і може служити відмінною відправною точкою для тих, хто хоче порівнювати зображення.

З його допомогою можна легко виявляти зміни, але для детального аналізу або специфічних завдань може знадобитися використання додаткових інструментів та бібліотек.

2.2. Трансформація зображено з подальшим порівнянням

Трансформація перспективи.

OpenCV, або Open Source Computer Vision Library, є однією з найбільш популярних бібліотек для комп'ютерного зору. Вона надає ряд інструментів для обробки зображень, включаючи трансформацію перспективи. Метод cv2.warpPerspective дозволяє змінювати перспективу зображення в OpenCV, що може бути корисним для виправлення спотворень або зміни точки зору.

Основні концепції [16]:

1. Визначення точок: для виконання трансформації перспективи необхідно визначити чотири точки на вхідному зображенні та чотири відповідні точки на вихідному зображенні. Ці точки дозволять створити матрицю перспективи.

2. Створення матриці перспективи: функція cv2.getPerspectiveTransform приймає два масиви з чотирьох точок для вхідного та вихідного зображень і повертає матрицю перспективи.

3. Застосування трансформації: функція cv2.warpPerspective використовує матрицю перспективи для зміни перспективи зображення.

Трансформація перспективи може застосовуватись у наступних випадках:

- виправлення спотворень: якщо фотографія була зроблена під кутом до плоскої поверхні, можна використовувати трансформацію перспективи для отримання виду «зверху вниз»;
- зміна точки зору: трансформація перспективи може використовуватися для імітації зміни позиції спостерігача;
- аугментована реальність: інтеграція об'єктів віртуального світу в реальний світ потребує зміни перспективи, щоб вони виглядали природно.

Ми можемо розглянути трансформацію перспективи на зображенні з дрону. До прикладу, нам потрібно сформувати карту місцевості чи більш точно визначити координати точки на місцевості за допомогою прив’язки до орієнтирів. Для цього нам потрібно трансформувати зображення, яке було відзнято під кутом, в зображення з видом «згори» (Рис. 2.5) [17].

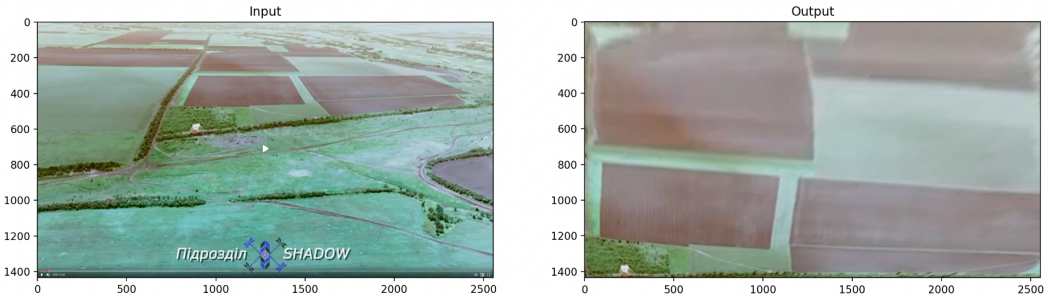


Рис 2.5 Трансформування перспективи зображення з дрону

До обмежень трансформації перспективи належать вибору ключових точок, втрата даних та обчислювальна складність.

Правильний вибір точок відповідності є важливим для отримання точної матриці перспективи. Невірний вибір може призвести до неправильної трансформації.

Великі зміни перспективи можуть призвести до втрати частини зображення або низької якості результуючого зображення.

Хоча OpenCV є оптимізованим для обробки зображень, трансформація перспективи у великих роздільних здатностях може бути вимогливою до ресурсів.

Отже, трансформація перспективи в OpenCV є потужним інструментом для роботи з зображеннями. З її допомогою можна виправляти спотворення, змінювати перспективу та інтегрувати віртуальні об'єкти в реальний світ. Однак, як і з будь-яким іншим інструментом, необхідно розуміти її обмеження і використовувати відповідно до конкретних потреб і ситуацій.

Порогова зміна зображення. Глобальне порогове значення (V=127).

Порогова зміна, або бінаризація зображення, є одним з найпоширеніших методів обробки зображень. Ця техніка дозволяє виділити певні області зображення, конвертуючи його в двокольорове (чорно-біле) представлення. Глобальне порогове значення означає встановлення одного значення порогу для всього зображення [18].

Коли говорять про глобальне порогове значення, це означає, що для всього зображення використовується одне й те ж порогове значення. У даному випадку V = 127.

Це означає, що усі пікселі зі значенням яскравості нижчим за 127 будуть конвертовані в чорний колір, а всі інші – в білий (Рис. 2.6).

Процес порогової зміни складається з:

1. Завантаження зображення: завантажте вхідне зображення, яке потребує бінаризації.

2. Конвертація у сіре: зазвичай зображення перетворюють в сіре (градації сірого), щоб працювати тільки з яскравістю, а не з трьома каналами кольору.

3. Застосування порогового значення: усі пікселі з яскравістю нижче 127 конвертуються в чорний колір, а інші – в білий.

До переваг глобальної порогової зміни відносять простоту виконання - глобальна порогова зміна є простим методом, що не вимагає складних обчислень або параметрів.

Іншою перевагою є швидкість. Через свою простоту, цей метод є дуже швидким та ефективним для обробки великих зображень [19].

До обмежень можна віднести неідеальні зображення – глобальна порогова зміна може бути неефективною для зображень з різним освітленням або низьким контрастом.

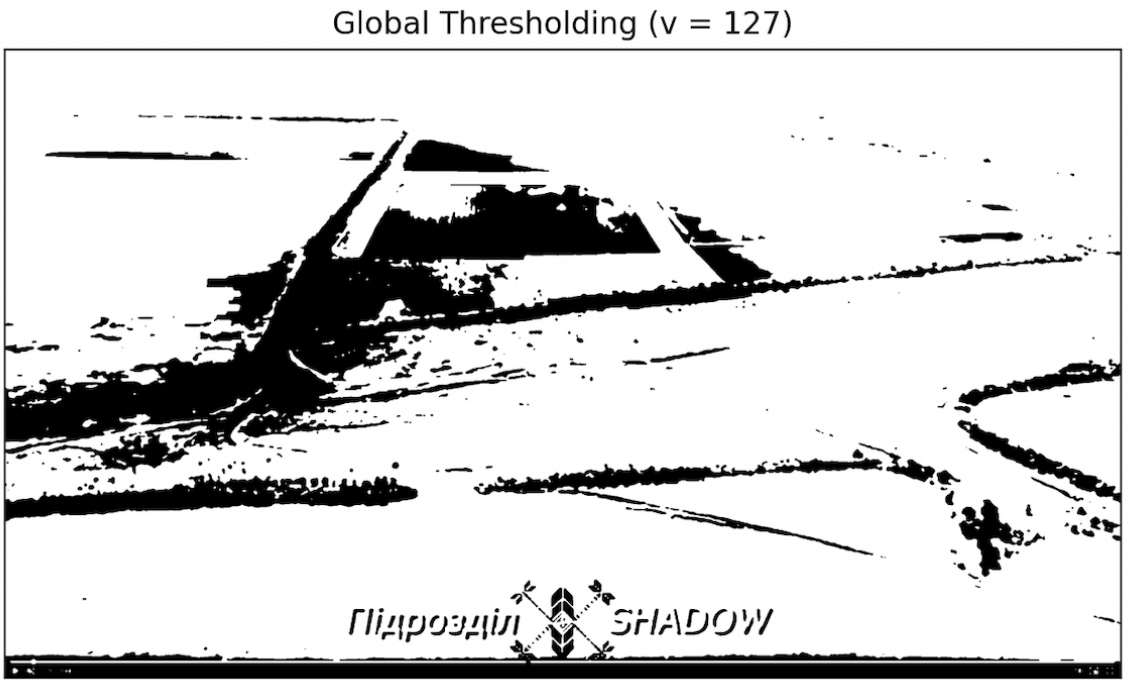


Рис 2.6 Зміна зображення з застосуванням глобального порогового значення

Ще одним обмеженням є втрата деталей – деякі деталі можуть бути втрачені після бінаризації, особливо у зображеннях з низьким контрастом між об'єктом і фоном.

Отже порогова зміна зображення є основним інструментом у сфері обробки зображень, який може бути використаний у різних застосуваннях, від виділення об'єктів до підготовки даних для машинного навчання. Глобальне порогове значення (V = 127) є простим та швидким методом для отримання бінарних зображень, але його ефективність може залежати від якості вхідного зображення та специфіки завдань.

Порогова зміна зображення. Адаптивне середнє порогове значення.

Порогова зміна зображення дозволяє виділити важливі області зображення, конвертуючи його в двокольорове представлення. Але що робити, якщо освітлення зображення нерівномірне? Адаптивне порогове значення намагається вирішити цю проблему, розраховуючи порог для різних областей зображення.

Адаптивне порогове значення використовує місцеві дані з кожного регіону зображення для визначення відповідного порогу (Рис. 2.7). У випадку адаптивного середнього порогового значення порог для кожного пікселя визначається середнім значенням яскравості в його сусідніх пікселях [20].

Процес адаптивного середнього порогового значення:

1. Завантаження та підготовка зображення: завантажте зображення і конвертуйте його в градації сірого.

2. Вибір розміру вікна: виберіть розмір місцевого вікна (наприклад, 11x11), яке буде використовуватися для визначення порогу кожного пікселя.

3. Розрахунок порогу: для кожного пікселя зображення розрахуйте середнє значення яскравості від усіх пікселів у вибраному вікні і використовуйте це середнє значення як поріг.

4. Бінаризація: за допомогою розрахованого порогу конвертуйте кожен піксель в чорний або білий, залежно від його яскравості.

До переваг алгоритму відносяться обробка нерівномірного освітлення та збереження деталей.

Завдяки місцевому розрахунку порогу цей метод ефективно працює з зображеннями з нерівномірним освітленням. Та цей метод часто зберігає деталі, які можуть бути втрачені при глобальній пороговій зміні.

З обмежень можна виділити обчислювальну складність - адаптивна порогова зміна вимагає більше обчислень, ніж глобальна порогова зміна. Та вибір розміру вікна - неправильний вибір розміру вікна може призвести до втрати деталей або небажаних шумів на зображенні.

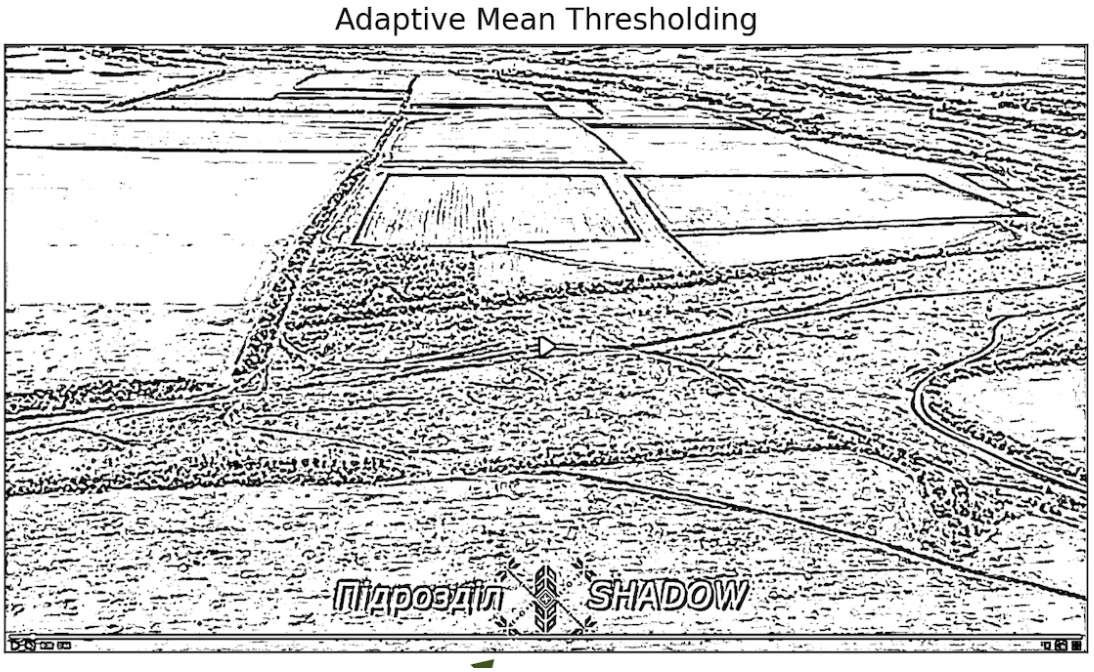


Рис 2.7 Приклад застосування адаптивного середнього порогового значення

Отже, адаптивне середнє порогове значення є ефективним методом для порогової зміни зображень із нерівномірним освітленням чи контрастом. Визначення порогу допомагає досягти більш точної бінаризації в порівнянні з глобальними методами. Однак правильний вибір параметрів, таких як розмір вікна, є ключовим для отримання найкращих результатів.

Порогова зміна зображення. Адаптивне порогове значення Гауса.

Порогова зміна зображення використовується для конвертації зображень в двокольорове (чорно-біле) представлення. Однак не всі зображення мають однорідний фон чи освітлення. У таких випадках загальне (глобальне) порогове значення може не дати оптимальних результатів. Тут на допомогу приходить адаптивне порогове значення, і одним з його варіантів є адаптивне порогове значення Гауса (Рис. 2.8).

При адаптивному пороговому значенні Гауса, порог для кожного пікселя зображення розраховується на основі вагової суми яскравості пікселів у вікні навколо цього пікселя. Ваги даються гаусовою функцією, тому ближчі пікселі матимуть більший вплив на порогове значення, ніж ті, що далі [21].

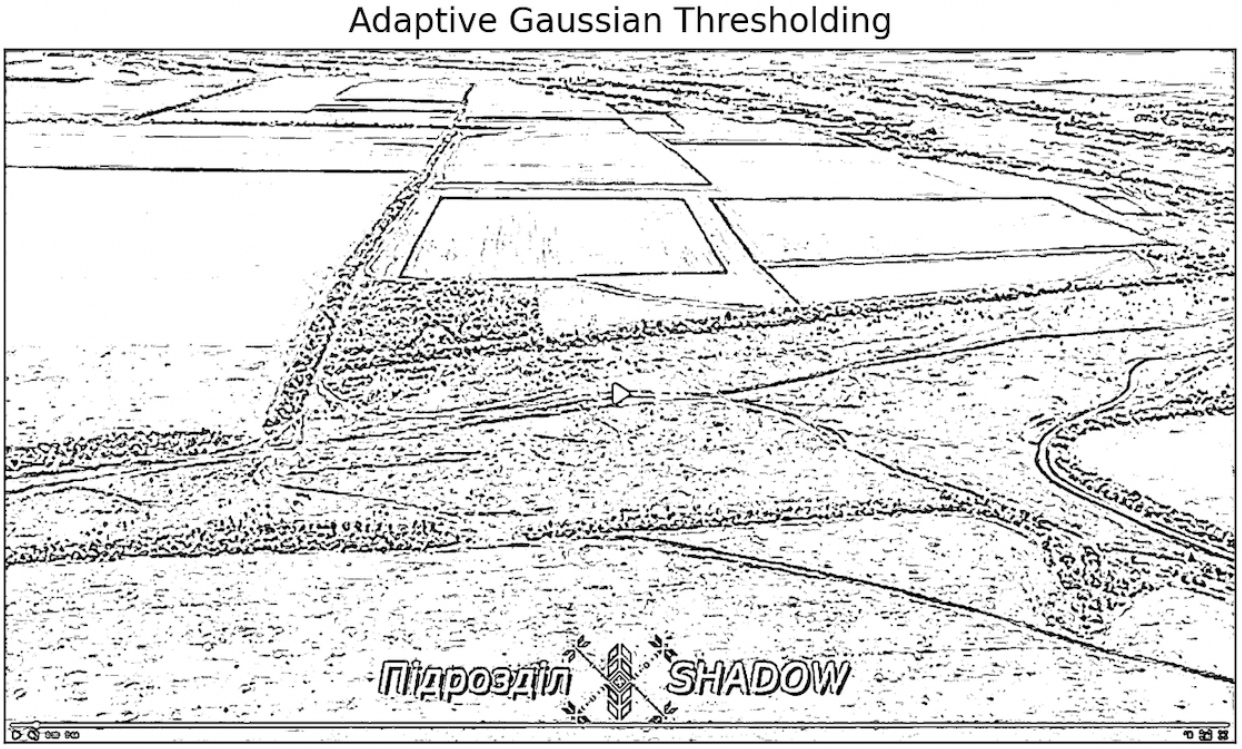


Рис 2.8 Приклад застосування адаптивного порогового значення Гауса

Процес адаптивного порогового значення Гауса:

1. Завантаження та підготовка зображення: завантажте зображення і конвертуйте його в градації сірого.

2. Вибір розміру вікна: виберіть розмір місцевого вікна, що буде використовуватися для розрахунку порогового значення.

3. Розрахунок порогу: для кожного пікселя зображення розрахуйте поріг на основі вагової суми яскравості пікселів в відповідному вікні, використовуючи гаусову функцію.

4. Бінаризація: застосовуйте розрахований порог для кожного пікселя, конвертуючи його в чорний або білий, в залежності від його яскравості.

До переваг адаптивного порогового значення Гауса відносять:

- точність в нерівномірно освітлених областях: завдяки використанню вагових коефіцієнтів метод більш чутливий до локальних особливостей зображення, таких як тіні або відблиски;

- збереження деталей: зображення часто втрачає деталі при використанні простої бінаризації, але метод Гауса може зберегти більше деталей завдяки його адаптивності.

Обмеженнями даного алгоритму є вища обчислювальна складність та вибір параметрів.

Адаптивна порогова зміна Гауса є обчислювально витратнішою, ніж інші методи порогової зміни.

Отже, адаптивне порогове значення Гауса є потужним інструментом для порогової зміни зображень з нерівномірним фоном або освітленням. Він враховує локальну інформацію зображення і застосовує гауссову вагову схему для отримання більш точних порогових значень. Якщо правильно налаштувати параметри, цей метод може надати чіткі, деталізовані бінарні зображення.

Виявлення країв.

Виявлення країв є однією з основних операцій обробки зображень, що використовується для визначення місця зміни інтенсивності яскравості або кольору на зображенні. Ця зміна інтенсивності може вказувати на присутність краю, що, в свою чергу, може вказувати на границю між двома об'єктами або частинами об'єкта.

Методи виявлення країв:

1. Метод градієнтів: використовуючи градієнтні операції, можна визначити напрямок, в якому інтенсивність зображення змінюється найшвидше. Основні методи:

- метод Собеля: використовує два 3x3 ядра для визначення горизонтальних і вертикальних змін інтенсивності;
- метод Превітта: схожий на Собеля, але має інше ядро;
- метод Шарра: використовує більші ядра для більшої точності.

2. Метод Лапласа: застосовує другу похідну для виявлення країв, що може виявити краї, де градієнт зображення проходить через нуль.

3. Метод Кенні: цей метод є одним з найбільш популярних і включає декілька кроків, зокрема згладжування, виявлення градієнтів, невелике зростання і подвійне порогове значення (Рис. 2.9).

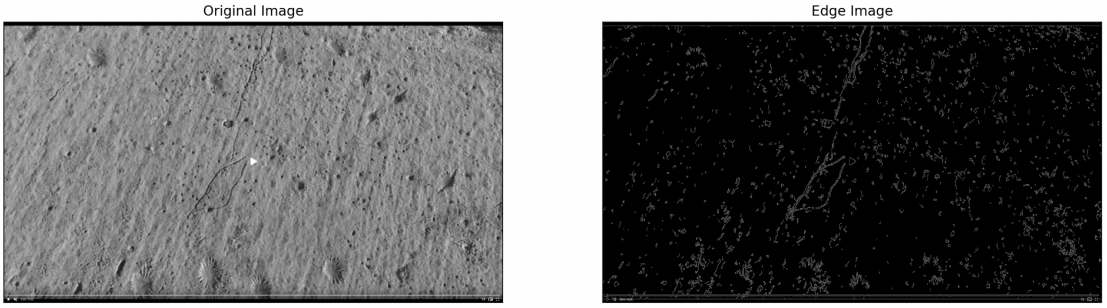


Рис 2.9 Приклад застосування алгоритму виявлення країв

До викликів даного алгоритму можемо віднести шум. Наявність шуму на зображенні може призвести до виявлення невірних країв. Часто перед виявленням країв зображення піддаються згладжуванню для зменшення впливу шуму.

Інший виклик - переривчасті краї. В деяких випадках можливе виявлення неповних або розривчастих країв.

Також викликом є товщина краю. Ідеально, край повинен бути однопіксельною лінією. Однак деякі методи можуть дати широкі або розмиті краї.

Алгоритм застосовується в таких сферах:

- медичне зображення: для виявлення патологічних змін, таких як пухлини;
- автоматичне визначення об'єктів: для визначення границь об'єктів на зображенні;
- відеонагляд: для визначення рухомих об'єктів.

Виявлення країв є важливою складовою в області обробки зображень. Він допомагає виділити важливу інформацію на зображенні і може бути використаний у різноманітних застосуваннях, від медичної діагностики до розпізнавання об'єктів. З розвитком комп'ютерного зору і глибокого навчання

методи виявлення країв продовжують розвиватися, набуваючи більшої точності та надійності.

2.3. Пошук об’єктів за допомогою моделей ML

Послідовна модель: зразок з літаками.

Послідовна модель в Keras - це лінійний стек шарів нейронних мереж. Вона дозволяє легко та швидко створювати моделі, додаючи до них різні шари. Один з застосунків цієї моделі - класифікація зображень, як, наприклад, описано далі, де ми класифікуємо зображення літаків (Рис. 2.10).

Для початку потрібно імпортувати необхідні бібліотеки та модулі. Після цього можна почати створення моделі за допомогою Sequential(). Зазвичай для класифікації зображень використовуються конволюційні нейронні мережі (CNN), які добре підходять для аналізу візуальних даних.

Спочатку додаємо конволюційний шар, використовуючи Conv2D, де визначаємо кількість фільтрів, розмір ядра та функцію активації. Далі, можна додати шар макспулінгу за допомогою MaxPooling2D для зменшення розміру вхідного зображення. Після декількох таких шарів, зображення «втратить» свої просторові характеристики, але «запам'ятає» ключові особливості.

Коли основна структура моделі готова, зображення пропускаються через повністю з'єднаний шар (Dense) для класифікації. Перед цим використовують шар Flatten(), щоб перетворити двовимірний масив у вектор. Останній шар Dense має вказану кількість нейронів, яка відповідає кількості класів, та функцію активації softmax.

Для тренування моделі потрібно визначити функцію втрат та оптимізатор. Після компіляції моделі вона готова до тренування за допомогою метода fit, де вказуються тренувальні дані, кількість епох та розмір партії.

Після тренування можна використовувати метод evaluate для оцінки точності моделі на тестових даних. Також можна використовувати метод predict для класифікації нових зображень [22].

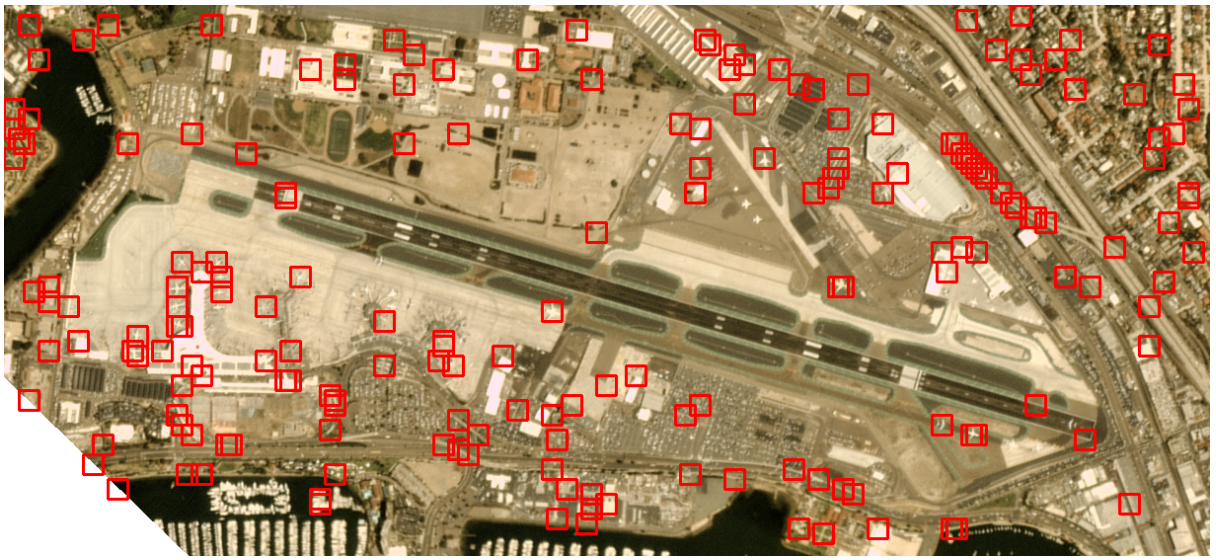


Рис 2.10 Результат пошуку літаків послідовною моделлю

Отже, послідовна модель в Keras - це потужний інструмент для створення нейронних мереж. Використовуючи її для класифікації зображень літаків, можна досягти високої точності та розпізнавання. Це підтверджує потенціал глибокого навчання в задачах обробки зображень та комп'ютерного зору.

Знаходження об’єктів за допомогою ImageAI та YOLOv3 моделі.

Сучасні задачі комп'ютерного зору, такі як виявлення об'єктів на зображеннях, вимагають високої точності та швидкодії. Однією з передових технік в цій області є модель YOLO (You Only Look Once), зокрема її третя версія - YOLOv3. ImageAI - це бібліотека Python, яка робить роботу з YOLOv3 простішою та зручнішою.

ImageAI - це бібліотека глибокого навчання для Python, яка пропонує простий та зручний інтерфейс для роботи з моделями глибокого навчання. Вона підтримує декілька моделей, включаючи YOLOv3, для виявлення, класифікації, прогнозування та навіть для тренування моделей на ваших даних.

YOLOv3 - це модель глибокого навчання для виявлення об'єктів, яка відрізняється своєю швидкодією та точністю (Рис. 2.11). Основна ідея YOLO полягає в тому, що замість традиційного підходу до виявлення об'єктів (декілька разів аналізувати різні частини зображення), YOLO аналізує зображення один раз, швидко визначаючи розташування та класифікацію об'єктів [23].



Рис 2.11 Знаходження літака моделлю YOLOv3

Третя версія YOLO має додаткові функції для більш точного виявлення об'єктів різного розміру та в декілька разів покращує точність в порівнянні з попередніми версіями.

Щоб почати роботу з ImageAI та YOLOv3, потрібно встановити бібліотеку ImageAI та завантажити натреновані ваги моделі YOLOv3 чи натренувати їх самостійно (Рис. 2.12).



Рис 2.12 Знаходження літаків моделлю YOLOv3

Після цього можна використовувати ImageAI для завантаження моделі, встановлення шляху до ваг та виконання виявлення об'єктів на зображеннях. Бібліотека дозволяє налаштовувати різні параметри, такі як порогове значення

впевненості для виявлення об'єктів, кастомізувати класи об'єктів, які потрібно виявляти, та інше.

Виявлення об'єктів є ключовою задачею в галузі комп'ютерного зору. Завдяки бібліотекам та моделям, таким як ImageAI та YOLOv3, розробники та дослідники можуть швидко та ефективно реалізовувати системи виявлення об'єктів високої точності. Зручний інтерфейс ImageAI, у поєднанні з потужною архітектурою YOLOv3, робить процес роботи з глибоким навчанням доступним та продуктивним.

Висновки до розділу 2

Отже, у цьому розділі було ретельно досліджено та проаналізовано різні методи трансформації зображень. Велика увага приділялася вивченню технік трансформації перспективи, що дозволяють змінювати оглядовий кут та розмір об'єктів на зображеннях. Також розглядалася важлива тема порогової зміни зображень, і вивчались як глобальні підходи до встановлення порогового значення, так і адаптивні методи, які дозволяють змінювати поріг в залежності від локальних особливостей зображення. Окремо досліджувались методи виявлення країв, які є важливим етапом у підготовці зображень до подальшого аналізу.

Результати цього дослідження мають велике практичне значення, оскільки вони вказують на те, як можна покращити якість вхідних зображень перед подальшим аналізом. Оптимізовані методи трансформації та порогової зміни зображень можуть забезпечити більш точні результати аналізу та розпізнавання об'єктів на зображеннях. Крім того, використання методів виявлення країв допомагає виділити важливі деталі та контури об'єктів, що також сприяє підвищенню точності обробки візуальних даних.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПОРІВНЯННЯ, ТРАНСФОРМАЦІЇ ЗОБРАЖЕНЬ. МОДЕЛЕЙ ML

3.1 Попіксельне порівняння за допомогою Computer Vision

Накладання двох BGR2GRAY фотографій та послідуюче порівняння.

Розробка серверних додатків для обробки запитів є невід'ємною частиною багатьох інформаційних технологій, і в цьому контексті однією з популярних бібліотек для створення веб серверів на мові програмування Python є Flask.

Flask – це легкий та гнучкий фреймворк, призначений для швидкої розробки веб застосунків. Використовуючи Flask, можна створити сервер, який обробляє HTTP-запити та надсилає HTTP-відповіді [24].

Flask базується на принципах розробки веб додатків за моделлю «маршрутизація-функція». Значенням «маршруту» є URL-адреса, на яку можна відправляти HTTP-запити, асоційовані з певною функцією обробки. Функції обробки визначають логіку відповіді на запити та можуть включати операції, такі як обробка даних, взаємодія з базою даних та інші.

Створення сервера на Flask передбачає імпортування класу Flask та визначення маршрутів за допомогою декораторів [25].

У багатьох випадках, важливим завданням є обробка зображень на сервері. Використовуючи Flask, ми можемо створити сервер, який приймає зображення у вигляді запитів та повертає результати їх обробки [26].

У лістингу, наведеному нижче, розроблено ендпоінт «compare_gray», який порівнюватиме два зображення в режимі відтінків сірого.

Одним із підходів до реалізації цього є створення функції «compare_gray_image_pixels», яка приймає два зображення у форматі PNG та порівнює їх за допомогою пікселів. Після цього, ми створили ендпоінт «compare_gray», який оброблятиме POST-запити, приймаючи файли та викликаючи зазначену функцію.

Лістинг 3.1

Реалізація серверу з «compare_gray» ендпоінтом

```
from flask import Flask, request, send_file
from compare_gray_image_pixels import compare_gray_image_pixels
app = Flask(__name__)

@app.route('/compare_gray', methods=['POST'])
def compare_gray():
  if 'file1' not in request.files or 'file2' not in request.files:
    return 'Files are not uploaded', 400

  file1 = request.files['file1']
  file2 = request.files['file2']
  fileMimeType = file1.mimetype
  if file1.mimetype.startswith('image/') and
file2.mimetype.startswith('image/'):
    # Save the files to disk or process them in memory
    file1.save('./compare/pixel_compare_img_1.png')
    file2.save('./compare/pixel_compare_img_2.png')
    # Compare the images
    compare_gray_image_pixels('./compare/pixel_compare_img_1.png',
'./compare/pixel_compare_img_2.png')
    result = open('./compare/pixel_compare_result.png', 'rb')
    return send_file(result, mimetype=fileMimeType)
  else:
    return 'Invalid file type', 400
```

Давайте розглянемо функцію «compare_gray_image_pixels» детальніше:

1. Функція має містити імпорт бібліотек cv2 (OpenCV) і imutils. OpenCV використовується для операцій з обробки зображень [11], а imutils надає декілька зручних функцій для простоти роботи з зображеннями [27].

2. Аргументами приймає шляхи до оригінального і нового зображень.

3. Оригінальне та нове зображення зчитуються за допомогою OpenCV, і їм присвоюється новий розмір за допомогою функції imutils.resize.

4. Створюється копія оригінального зображення, і потім обчислюється абсолютна різниця між оригінальним і новим зображеннями [28].

5. Результат порівняння перетворюється в відтінки сірого [13], а потім виконується дилатація для виокремлення відмінностей [29].

6. Використовується бінаризація для визначення контурів, які подалі використовуються для виділення відмінностей на новому зображенні [28].

7. Для кожного контуру обводиться прямокутник на новому зображенні, і результат зберігається у файлі «pixel_compare_result.png» [14].

8. Додавання експорту функції за допомогою __all__ [29].

Лістинг 3.2

Реалізація «compare_gray_image_pixels»

```
import cv2
import imutils

def compare_gray_image_pixels(original_path: str, new_path: str):
  original = cv2.imread(original_path)
  new = cv2.imread(new_path)

  original = imutils.resize(original, height = 600)
  new = imutils.resize(new, height = 600)

  diff = original.copy()
  cv2.absdiff(original, new, diff)

  gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

  for i in range(0, 3):
    dilated = cv2.dilate(gray.copy(), None, iterations= i + 1)

  (T, thresh) = cv2.threshold(dilated, 3, 255, cv2.THRESH_BINARY)

  cnts = cv2.findContours(thresh, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)

  for c in cnts:
    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(new, (x, y), (x + w, y + h), (0, 255, 0), 2)

  cv2.imwrite("./compare_gray/pixel_compare_result.png", new)

__all__ = ['compare_gray_image_pixels']
```

Для проведення тестування ендпоінту веб сервера використовується програма Postman. Postman - це універсальний інструмент для розробників, який надає зручний інтерфейс для взаємодії з веб серверами та тестування їхніх API [30].

Завдяки Postman можна легко створювати HTTP-запити, відправляти їх до визначених ендпоінтів, отримувати відповіді в реальному часі та аналізувати результати. Використання Postman спрощує процес тестування, дозволяючи

ефективно перевіряти правильність роботи ендпоінтів та впевнено впроваджувати внесені зміни у веб додатки.

Вибір та використання Postman у проєкті було обумовлено його високою функціональністю та зручністю для тестування та взаємодії з API. Postman надавав зручний інтерфейс для створення та відправлення HTTP-запитів до сервера, що визначено в проєкті (Рис. 3.1).

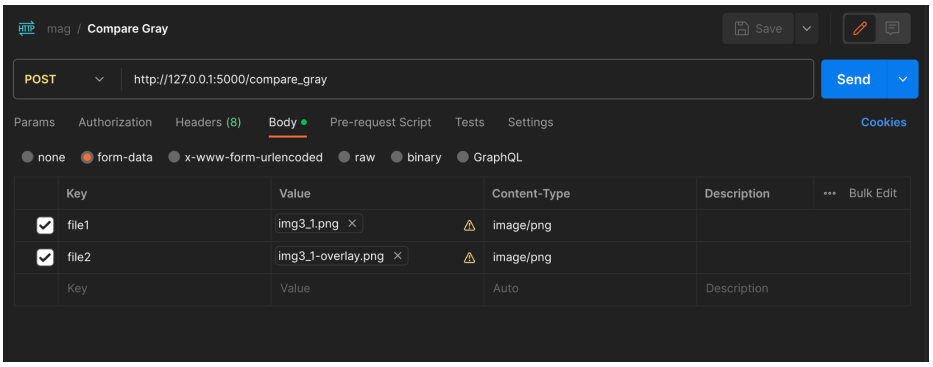


Рис 3.1 Тестування «compare_gray» ендпоінту

Нижче наведені зображення для виконання порівняння (Рис. 3.2).



Рис 3.2 Зображення для порівняння

Отриманий результат відображено нижче (Рис. 3.3).



Рис 3.3 Результат порівняння «compare_gray»

Порівняння за допомогою модулю PIL.

У продовження функціоналу, який вже реалізований у «compare_gray» ендпоінті, був доданий новий ендпоінт «compare_pillow». Цей новий ендпоінт служить для подальшого розширення можливостей порівняння зображень у веб застосунку, основаному на Flask.

«compare_pillow» ендпоінт приймає два файли у форматі PNG для порівняння та викликає функцію «compare_pillow», яка реалізує алгоритм порівняння фотографій, використовуючи бібліотеку Pillow [4]. Цей алгоритм дозволяє проводити більш детальне порівняння, а також виявляти різницю у різних аспектах зображень.

Лістинг 3.3

Реалізація функції «compare_pillow»

```
from PIL import Image, ImageDraw
import numpy as np

def compare_pillow(image1, image2):
  #Opening images and converting to B&W
  original = Image.open(image1)
  new = Image.open(image2)

  original_converted = original.convert("L") # I and L have the same
result
  new_converted = new.convert("L") # I and L have the same result
  original_data = original_converted.getdata()
  new_data = new_converted.getdata()

  #Subtracting pixels
  diff_pix = np.subtract(original_data, new_data)

  #Creating a new image with only the different pixels
  img_final = Image.new("L", original.size)
  img_final.putdata(diff_pix)

  #Calculating box coordinates
  threshold = 25
  box_width = 50
  Drawer = ImageDraw.Draw(new)
  for y in range(original.size[1]):
    for x in range(original.size[0]):
      i = y * original.size[0] + x
      if abs(diff_pix[i]) > threshold:
        Drawer.rectangle((x, y, x+box_width, y+box_width),
outline="red", width=3)
```

```
    #Saving the image with box
    new.save('./compare_pil/pillow_compare_result.png')

__all__ = ['compare_pillow']
```

Основна ідея «compare_pillow» полягає в наступному:

1. Зчитуються та конвертуються два зображення в форматі чорно-білого [31].

2. Відбувається віднімання значень пікселів одного зображення від іншого.

3. Створюється нове зображення, де кожен піксель відображає різницю між відповідними пікселями оригінальних зображень [33].

4. Визначаються координати прямокутників (боксів) для виділення відмінностей, та ці відмінності виділяються червоним контуром на новому зображенні.

5. Зберігається зображення з виділеними відмінностями у вказаному каталозі [13].

Перевіримо ендпоінт за допомогою Postman (Рис. 3.4):

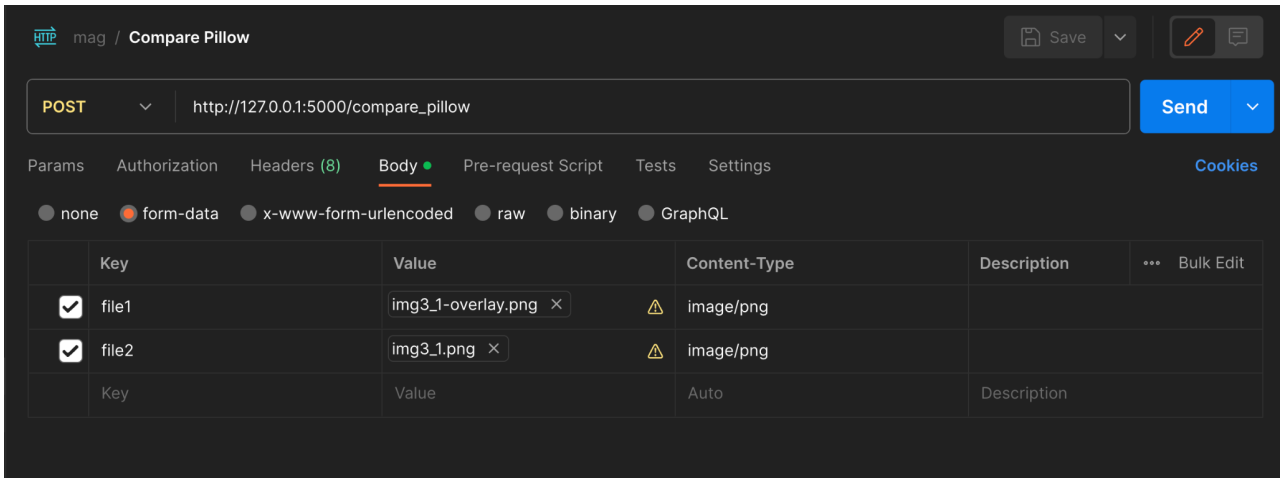


Рис 3.4 Тестування «compare_pillow» ендпоінту

Для порівняння використовувались зображення, наведені на рисунку 1.2. Отриманий результат зображено на рисунку нижче (Рис. 3.5).



Рис 3.5 Результат порівняння «compare_pillow»

Ендпоінти «compare_gray» та «compare_pillow» представляють собою два різних підходи до порівняння зображень у веб застосунку на основі Flask.

Ендпоінт «compare_gray» використовує бібліотеку OpenCV для порівняння зображень в градаціях сірого. Цей метод дозволяє швидко виявляти відмінності, але він чутливий до змін в освітленні та кольорі.

З іншого боку, ендпоінт «compare_pillow» використовує бібліотеку Pillow та представляє собою більш гнучкий підхід. Він використовує чорно-білі версії зображень та визначає відмінності в інтенсивності пікселів. Цей метод менш чутливий до колірних відмінностей та дозволяє точніше виділяти різниці в зображеннях.

Обираючи між цими ендпоінтами, розробник може враховувати конкретні потреби свого проекту, враховуючи особливості порівняння, які найкраще відповідають вимогам та характеристикам зображень у конкретному випадку.

3.2 Трансформація зображень з послідуючим порівнянням

Трансформація перспективи.

Трансформація перспективи у бібліотеці OpenCV – це важливий аспект обробки зображень, який дозволяє змінювати та адаптувати перспективу зображення, враховуючи його геометричні особливості.

Створюємо ендпоінт «perspective_transform», який приймає файл у форматі PNG та відомі координати для здійснення перспективної трансформації. Координати для зручності вказуються у відсотках.

Після цього викликається функція «perspective_transform», яка виконує трансформацію зображення, враховуючи вказані параметри та параметри трансформації [16].

Лістинг ендпоінту «perspective_transform» вказувати не будемо. Код створення нового ендпоінту алалогічний до наведених вище.

Розглянемо детальніше функцію «perspective_transform», яка викликається в ендпоінті.

Лістинг 3.4

Реалізація функції «perspective_transform»

```
import cv2
import numpy as np

def perspective_transform(img_path, inputPoints):
  img = cv2.imread(img_path)
  assert img is not None, "file could not be read, check with
os.path.exists()"
  height, width, channel = img.shape
  pixelCoords = [((point[0] / 100) * width, (point[1] / 100) * height)
for point in inputPoints]
  resultPoints = np.float32([[0, 0],[width, 0],[0, height],[width,
height]])
  M = cv2.getPerspectiveTransform(np.float32(pixelCoords),
resultPoints)
  dst = cv2.warpPerspective(img, M, (width, height))
  cv2.imwrite('./transformation/transform_result.png', dst)
```

```
__all__ = ['perspective_transform']
```

Давайте детальніше розглянемо, як вона працює:

1. img_path - це шлях до вхідного зображення, яке буде трансформовано.

2. inputPoints - це список координат точок на вхідному зображенні, які визначають область для перспективної трансформації. Координати вказуються у відсотках від ширини та висоти зображення [17].

3. cv2.imread(img_path) - зчитує вхідне зображення за вказаним шляхом.

4. assert img is not None - перевіряє, чи вдалося зчитати зображення.

5. pixelCoords - переводить відсоткові координати в піксельні координати для визначення точок перспективної трансформації [34].

6. resultPoints - це кінцеві координати для трансформації, вказані у вигляді чотирикутника.

7. cv2.getPerspectiveTransform(np.float32(pixelCoords), resultPoints) - отримує матрицю перспективної трансформації на основі вхідних та вихідних координат.

8. cv2.warpPerspective(img, M, (width, height)) - застосовує трансформацію до вхідного зображення.

9. cv2.imwrite('./transformation/transform_result.png', dst) - зберігає трансформоване зображення за вказаним шляхом (Рис. 3.6) [35].

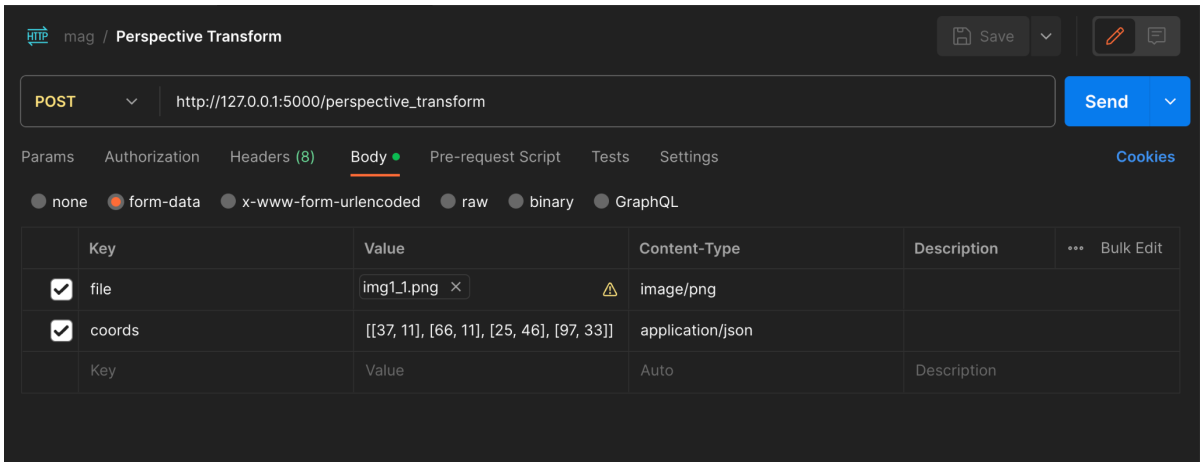


Рис 3.6 Тестування ендпоінту трансформації перспективи

Фото для трансформації та результат наведено нижче (Рис. 3.7).



Рис 3.7 Результат трансформації перспективи

Порогова зміна зображення. Глобальне порогове значення (V=127).

У наступних ендпоінтах основний фокус буде приділено пороговій зміні зображень та їх подальшому порівнянню. Зокрема, «global_thresholding» ендпоінт буде відповідальний за отримання двох зображень у форматі PNG та виклик функції «global_thresholding».

У цій функції буде застосовано глобальний поріг для кожного пікселя на зображенні, змінюючи його значення в залежності від порогового значення. Змінені фотографії, таким чином, будуть представляти собою результат порогової обробки [18].

Після отримання змінених зображень ендпоінт «global_thresholding» буде викликати метод «compare_thresholded_image», який порівняє відповідні пікселі на двох зображеннях та виділить області, де відбулися зміни внаслідок порогової обробки [36].

Цей підхід дозволить ефективно визначити відмінності між двома фотографіями, які виникли внаслідок глобальної порогової зміни значень пікселів. Такий підхід до порівняння зображень має широкий спектр застосувань, включаючи виявлення різниці в освітленні, контрасті чи інших характеристиках зображень [19].

Наведемо код реалізації застосування глобального порогового значення.

Лістинг 3.5

Реалізація функції «global_thresholding»

```
import cv2

def global_thresholding(image_path, result_path):
  img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
  assert img is not None, "file could not be read, check with
os.path.exists()"
  img = cv2.medianBlur(img, 5)
  thresholdedImage = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1]
  cv2.imwrite(result_path, thresholdedImage)

__all__ = ['global_thresholding']
```

Далі ретельно розглянемо код функції «compare_thresholded_image». Ця функція відіграє ключову роль у порівнянні зображень, які були піддані глобальній пороговій зміні. Проаналізуємо кожен крок цього коду, розкриваючи, як саме відбувається порівняння та визначення різниць між зображеннями.

Лістинг 3.6

Реалізація функції «compare_thresholded_image»

```
import cv2

def compare_thresholded_image(original_img_path: str, new_img_path:
str, result_img_path: str, min_contour_area: int = 100):
  original = cv2.imread(original_img_path)
  new = cv2.imread(new_img_path)

  result = new.copy()
  diff = cv2.absdiff(original, new)

  diff_gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)

  _, thresh = cv2.threshold(diff_gray, 25, 255, cv2.THRESH_BINARY)

  contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

  contours = [cnt for cnt in contours if cv2.contourArea(cnt) >
min_contour_area]
  for contour in contours:
    (x, y, w, h) = cv2.boundingRect(contour)
    cv2.rectangle(result, (x, y), (x + w, y + h), (0, 255, 0), 2)

  cv2.imwrite(result_img_path, result)

__all__ = ['compare_thresholded_image']
```

Спочатку, функція завантажує вихідне та нове зображення. Після цього вона обчислює різницю між ними за допомогою функції cv2.absdiff(), яка віднімає значення кожного пікселя одного зображення від відповідного пікселя іншого [12].

Отримана різниця перетворюється в чорно-біле зображення, і тут використовується порогова обробка для виділення областей, де відбулися зміни. Знайдені контури на цьому зображенні дозволяють визначити границі змінених областей [29].

Функція використовує параметр min_contour_area для відфільтрування невеликих контурів, що може бути корисно при ігноруванні дрібних змін або шуму на зображенні.

Нарешті, знайдені контури відображаються на новому зображенні зеленими прямокутниками, які відзначають змінені області. Результат порівняння зберігається у вказаному шляху result_img_path.

Варто відзначити, що функція також зберігає вихідне та нове зображення у властивостях compare_thresholded_image.original та compare_thresholded_image.new відповідно. Це може бути використано у подальших операціях чи аналізі різниці між зображеннями.

На рисунку 3.8 зображено тестовий запит в Postman.

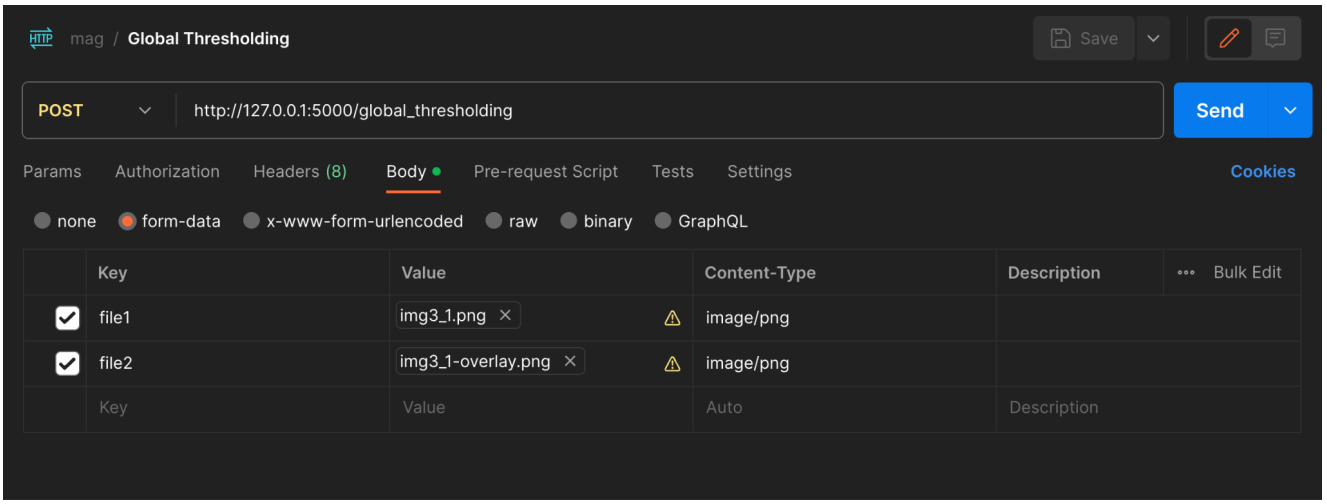


Рис 3.8 Тестовий запит глобального порогового значення

Фото для порівняння наведені в рисунку 2.2.

Нижче наведено результат застосування глобального порогового значення, порівняння двох фото та підсвічування різниці (Рис. 3.9-3.10).

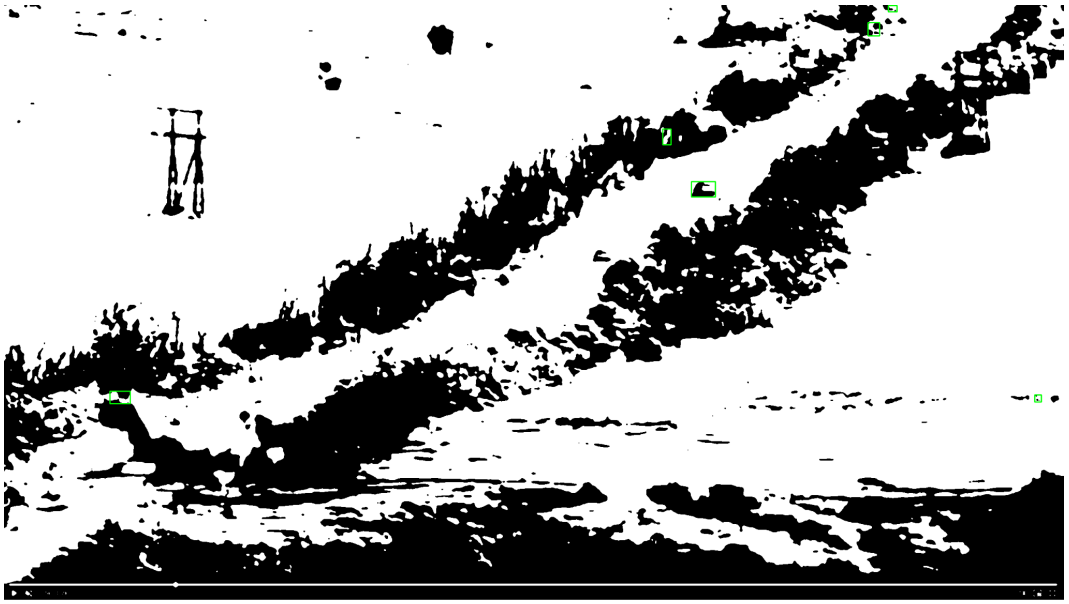


Рис 3.9 Результат порівняння фото з застосованим глобальним пороговим значенням та фільтрацією шуму

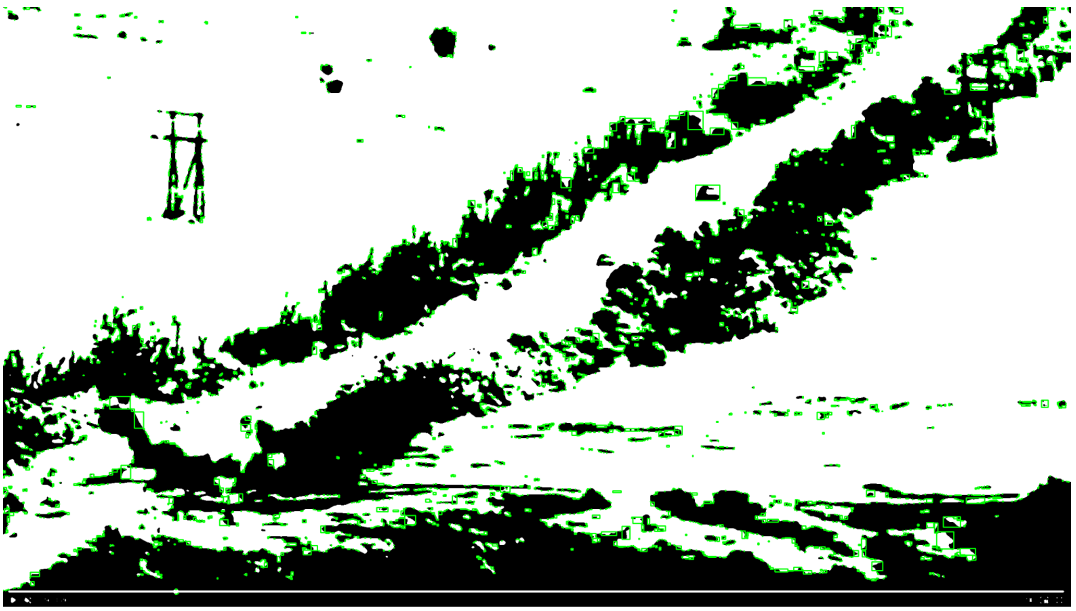


Рис 3.10 Результат порівняння фото з застосованим глобальним пороговим значенням без фільтрації шуму

Порогова зміна зображення. Адаптивне середнє порогове значення.

По аналогії із порівнянням зображень за допомогою глобального порогового значення, ми впроваджуємо ендпоінт «mean_thresholding». Цей ендпоінт визначає змінені області на фотографіях шляхом застосування методу середнього порогу.

Лістинг 3.7

Реалізація функції «mean_thresholding»

```
import cv2
def mean_thresholding(image_path, result_path):
 img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
 assert img is not None, "file could not be read, check with
os.path.exists()"
 img = cv2.medianBlur(img,5)
 thresholdedImage = cv2.adaptiveThreshold(img, 255 ,
cv2.ADAPTIVE_THRESH_MEAN_C,\
   cv2.THRESH_BINARY, 11, 2)
 cv2.imwrite(result_path, thresholdedImage)

__all__ = ['mean_thresholding']
```

Після отримання змінених зображень за допомогою цього методу, проводиться їх порівняння за допомогою функції «compare_thresholded_image» [15]. Тестовий запит описаний нижче (Рис. 3.11).

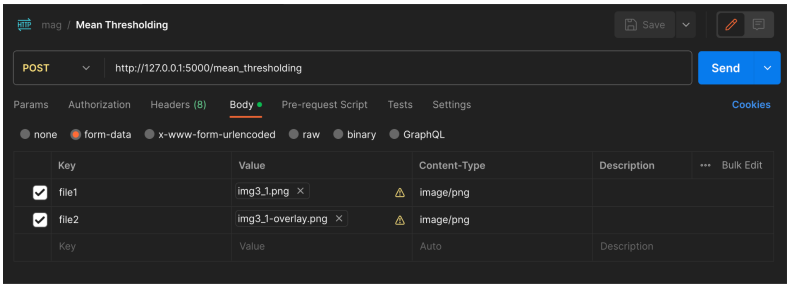


Рис 3.11 Тестовий запит на порівняння зображень із застосуванням адаптивного порогового значення

Із застосуванням зображень з рисунку 2.2 отримаємо результат зображений на рисунку нижче (Рис. 3.12).



Рис 3.12 Результат порівняння зображень із застосуванням адаптивного порогового значення

Порогова зміна зображення. Адаптивне порогове значення Гауса.

Останній приклад порогової обробки зображень, представлений у даній роботі, - це використання адаптивного порогового значення Гауса. Ендпоінт «gaussian_thresholding» запускає відповідну функцію для виконання адаптивної порогової обробки, а потім проводить порівняння отриманих трансформованих зображень [21].

Лістинг 3.8

Реалізація функції «gaussian_thresholding»

```
import cv2

def gaussian_thresholding(image_path, result_path):
 img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
  assert img is not None, "file could not be read, check with
os.path.exists()"
 img = cv2.medianBlur(img,5)
                          thresholdedImage                                =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
   cv2.THRESH_BINARY,11,2)
 cv2.imwrite(result_path, thresholdedImage)

__all__ = ['gaussian_thresholding']
```

Запит в Postman аналогічний попереднім ендпоінтам і містить два зображення.

На рисунку нижче зображений результат порівняння (Рис. 3.13) [30].



Рис 3.13 Результат порівняння зображень із застосуванням адаптивного порогового значення Гауса

Виявлення країв.

Метод виявлення країв, представлений ендпоінтом «edge_detection», виявився цікавим для подальшого порівняння зображень. Цей метод відокремлює виразні краї об'єктів на зображенні, надаючи можливість визначити їхні форми та розташування. Однак його використання не обмежується лише виявленням контурів об'єктів.

Ендпоінт «edge_detection» може бути ефективним інструментом для виявлення змін на зображеннях, оскільки зміни часто супроводжуються змінами в контурах об'єктів. Такий підхід дозволяє виділяти області, де відбулися зміни, що може бути корисно при аналізі динаміки змін на зображенні [37].

Лістинг 3.9

Реалізація функції «detect_edges»

```
import cv2
import numpy as np

def detect_edges(image_path, result_path):
  img = cv2.imread(image_path)
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  edges = cv2.Canny(gray, 50, 150, apertureSize = 3)

  cv2.imwrite(result_path, edges)

__all__ = ['detect_edges']
```

Результат виявлення країв та порівняння наведений на рисунку нижче Рис. 3.14).



Рис 3.14 Результат порівняння зображень із виявленими краями

3.3 Пошук об’єктів за допомогою моделі ML

Послідовна модель: зразок з літаками.

Проводимо навчання моделі на основі інформації, яку отримуємо з наборів даних, доступних на ресурсі Kaggle [38]. Наша мета - розробити модель, яка здатна ефективно визначати об'єкти, пов'язані з літаками, з цих наборів даних [22].

У рамках цього процесу ми створюємо ендпоінт «predict_planes», який дозволяє користувачам передавати файли у форматі PNG для подальшого аналізу. Цей ендпоінт викликає метод «predict_planes», вбудований у нашу модель для детекції літаків на зображеннях. Результат роботи моделі передається у відповідь на запит користувача.

Такий підхід дозволяє нам використовувати навчену модель для передбачення присутності літаків на зображеннях. Розгортаючи цей ендпоінт, ми надаємо можливість користувачам ефективно взаємодіяти з нашою моделлю та отримувати результати аналізу їхніх власних зображень [39].

Лістинг 3.10

Реалізація ендпоінту «predict_planes»

```
from predict_planes import predict_planes

@app.route('/predict_planes', methods=['POST'])
def plane_prediction():
  if 'file' not in request.files:
    return 'File is not uploaded', 400

  file = request.files['file']
  fileMimeType = file.mimetype
  if file.mimetype.startswith('image/'):
    file.save('./sequentical/image.png')
    predict_planes('./sequentical/image.png', 20, 20, 10)
    result = open('./sequentical/result.png', 'rb')
    return send_file(result, mimetype=fileMimeType)
  else:
    return 'Invalid file type', 400
```

У наступному лістингу ми розглянемо функцію «predict_planes», яка виконує важливий етап аналізу зображень. Ця функція використовує метод

розділення зображення на менші частини з перекриттям, і кожен з них перевіряється на наявність літака. Якщо в процесі аналізу виявлено літак, то цей об'єкт чітко виділяється на оригінальному зображенні.

Алгоритм розділення зображень відіграв ключову роль у нашому проєкті, дозволяючи нам шукати об'єкти на зображеннях більших розмірів, ніж ті, що використовувалися для навчання. Цей алгоритм орив дозволив нам розширити обсяг нашого аналізу та виявлення, роблячи можливим ефективну обробку високороздільних повітряних знімків і виявлення потенційних цілей на великих ландшафтах. Шляхом ефективного розбиття і обробки великих зображень алгоритм значно покращив нашу можливість виявляти об'єкти інтересу.

Алгоритм розпізнавання літаків важливий для визначення їхньої присутності та локалізації на великих зображеннях. Він дозволяє ефективно визначати області, що містять літаки, та підкреслює їх на оригінальному зображенні для подальшого аналізу [40].

Такий підхід є ключовим для використання моделі в реальних умовах, де може бути необхідно визначати літаки на великих панорамних зображеннях чи фотографіях. Функція «predict_planes» робить цей процес автоматизованим та відповідає на запитання про наявність літаків у зазначених областях зображення.

Лістинг 3.11

Реалізація функції «predict_planes»

```
import cv2
import numpy as np
from keras.models import load_model
import sys
import os

def crop_image(image_path, crop_size, stride):
  # Load the image
  img = cv2.imread(image_path)

  # Slide a window over the image and crop each sub-image
  for y in range(0, img.shape[0] - crop_size[1], stride):
    for x in range(0, img.shape[1] - crop_size[0], stride):
      crop = img[y:y+crop_size[1], x:x+crop_size[0]]
```

```
      # Do something with the cropped image, such as saving it to disk
      cv2.imwrite(f'./sequentical/cropped/crop_{x}_{y}.jpg', crop)

def predict_planes(image_path, w=20, h=20, stride=10):
  # Remove all files from the cropped folder
  fileNamesArray = os.listdir('./sequentical/cropped')
  for file in fileNamesArray:
    os.remove('./sequentical/cropped/' + file)

  crop_image(image_path, (w, h), stride)

  # Load the saved model
  model = load_model('./models/sequentical.h5')
  fileNamesArray = os.listdir('./sequentical/cropped')

  template = cv2.imread(image_path)
  template_output = template.copy()

  # print(sorted(fileNamesArray))
  for element in sorted(fileNamesArray):
    # Load the image
    img = cv2.imread('./sequentical/cropped/' + element)

    if img is None:
      sys.exit("Could not read the image.")

    # Preprocess the image
    img = cv2.resize(img, (20, 20))
    img = img.astype('float32') / 255.0

    # Use the model to predict the probabilities for each class
    probs = model.predict(np.expand_dims(img, axis=0))[0]

    # Threshold the probabilities to obtain a binary mask of the planes
    plane_mask = (probs > 0.5).astype('uint8')

    if plane_mask[0]:
      x = int(element.split(".")[0].split("_")[1])
      y = int(element.split(".")[0].split("_")[2])
      cv2.rectangle(template_output, (x, y), (x + w, y + h), (0, 0,
255), 2)

  # Remove all files from the cropped folder
  fileNamesArray = os.listdir('./sequentical/cropped')
  for file in fileNamesArray:
    os.remove('./sequentical/cropped/' + file)

  # Write result to image
  cv2.imwrite('./sequentical/result.png', template_output)

__all__ = ['predict_planes']
```

У наступному рисунку представлені результати виконання функції «predict_planes», яка виявила та локалізувала літаки на заданому зображенні.

Подібне візуальне представлення допомагає легше розуміти роботу алгоритму та його ефективність у виявленні об'єктів. Відзначимо, що виділення літаків на зображенні допомагає полегшити подальший аналіз та ідентифікацію цих об'єктів у контексті конкретної задачі аналізу зображень (Рис. 3.15).



Рис 3.15 Результат виявлення літаків секвенційною моделлю

Знаходження об’єктів за допомогою ImageAI та YOLOv3 моделі.

Ми проводимо навчання моделі на основі доступних даних, взятих з ресурсу Kaggle [41]. Досягаємо цього шляхом створення та оптимізації моделі, яка може ефективно розпізнавати літаки на зображеннях.

Надалі, в рамках розробки, ми впроваджуємо «yolo_detect_planes» ендпоінт, аналогічно до попереднього «predict_planes». Цей ендпоінт викликає метод «yolo_detect_planes» для виявлення літаків на зображеннях,

використовуючи технологію YOLO (You Only Look Once). Результати цієї операції будуть надані у відповіді на відповідний запит [23].

Використання YOLO для детекції літаків забезпечує високу точність та швидкодію процесу, що робить його ефективним засобом для виявлення об'єктів на зображеннях. Цей підхід дозволяє автоматично визначати літаки, надаючи користувачеві зручний та точний інструмент для аналізу фотографій [41].

Лістинг 3.12

Реалізація функції «yolo_detect_planes»

```
import cv2
from imageai.Detection.Custom import CustomObjectDetection

def yolo_detect_planes(image_path: str):
 detector = CustomObjectDetection()
 detector.setModelTypeAsYOLOv3()
 detector.setModelPath("./planes/models/yolov3_planes_last.pt")

detector.setJsonPath("./planes/json/planes_yolov3_detection_config.json")
 detector.loadModel()
 detections = detector.detectObjectsFromImage(image_path,
output_image_path="./planes/result.png")

__all__ = ['yolo_detect_planes']
```

На поданому нижче рисунку відображено зображення, яке піддається процесу детекції за допомогою нашої моделі (Рис. 3.16). Це зображення містить літак, який визначається та виділяється в результаті застосування розробленого методу детекції. Виявлений літак чітко виділяється на оригінальному зображенні, що дозволяє легко ідентифікувати його на фотографії та отримувати точні результати детекції. Цей процес виявлення об'єктів надає корисну інформацію та сприяє у вирішенні завдань, пов'язаних з аналізом та обробкою зображень, зокрема виявлення літаків на аерофотознімках.



Рис 3.16 Результат виявлення літаку моделлю YOLOv3

Висновки до розділу 3

У третьому розділі роботи був розглянутий процес пошуку та ідентифікації об'єктів за допомогою моделей машинного навчання, що включає в себе використання послідовної моделі та моделі YOLOv3 для виявлення об'єктів на зображеннях. В даному розділі проводився аналіз ефективності та точності використаних методів, а також розглядалися практичні аспекти застосування цих моделей у різних областях, зокрема у військовій та безпекових сферах.

Одним із важливих висновків з цього розділу є те, що використання сучасних технік машинного навчання у комп'ютерному зорі має великий потенціал для покращення процесу аналізу зображень. Застосування послідовної моделі та YOLOv3 дозволило досягти високого рівня точності та ефективності при виявленні об'єктів на зображеннях. Ці результати вказують на перспективи використання сучасних моделей машинного навчання у різних галузях, включаючи безпеку, де точний та швидкий аналіз зображень є критично важливим.

Важливим аспектом є інтеграція цих моделей у реальні системи та розробка практичних застосувань для різних завдань. З цією метою рекомендується подальше наукове дослідження та розвиток алгоритмів машинного навчання для забезпечення ще більшої точності та ефективності у сфері комп'ютерного зору.

ВИСНОВКИ

Отже, можна констатувати, що поставлена мета була досягнута завдяки ретельному дослідженню, розробці та оптимізації методів порівняння, трансформації та аналізу зображень у контексті комп'ютерного зору, а також через успішну інтеграцію цих методів у моделі машинного навчання. Основні результати цього дослідження включили в себе ряд ключових досягнень:

- розробка та вдосконалення ефективних алгоритмів порівняння зображень, які дозволили виявляти найменші відмінності та відзначати схожості між ними з високою точністю. Ці алгоритми стали основою для подальших досліджень та забезпечили високу точність виявлення змін на зображеннях;
- оптимізація методів трансформації зображень для покращення якості та підготовки їх до подальшого аналізу. Цей підхід покращив роботу з візуальними даними та дозволив отримувати більш точні результати у різних задачах;
- інтеграція розроблених методів із відомими моделями машинного навчання, зокрема з YOLOv3, щоб покращити процес виявлення та класифікації об'єктів на зображеннях. Ця інтеграція розширила можливості системи та підвищила її здатність розпізнавати об'єкти з високою точністю;
- отримані результати мають значний практичний потенціал і можуть бути використані у різних областях, де необхідний швидкий та точний аналіз зображень.

Для подальших наукових досліджень рекомендується спрямовувати зусилля на покращення алгоритмів з метою зниження обчислювальної складності та збільшення швидкості обробки. Також важливо розглядати можливості інтеграції розроблених методів у інші комп'ютерні системи та програмні комплекси. Зокрема, важливим напрямком може бути подальший розвиток алгоритмів глибокого навчання, що забезпечуватимуть ще більшу точність та ефективність у роботі з візуальними даними.

У підсумку, ця робота відкриває широкі перспективи для використання розроблених методів у практичних завданнях та вносить важливий внесок у розвиток сучасних технологій обробки зображень і комп'ютерного зору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Карапет Б. В. Знаходження відмінностей при порівнянні зображень з дрону, знаходження потенційної цілі / Р. С. Савіцький. // VI Всеукраїнська науково-технічна конференція «Комп’ютерні технології: інновації, проблеми, рішення» 29-30 листопада 2023 р. м. Житомир.
2. Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S. & Ghayvat, H. (2021). CNN variants for computer vision: History, architecture, application, challenges and future scope. Electronics, 10(20), 2470.
3. Davies, E. R. (2017). Computer vision: principles, algorithms, applications, learning. Academic Press.
4. Chityala, R., & Pudipeddi, S. (2020). Image processing and acquisition using Python. CRC Press.
5. Lv, H., Jing, D., & Zhan, W. (2021, September). Research on infrared image target detection technology based on YOLOv3 and computer vision. In Journal of Physics: Conference Series (Vol. 2033, No. 1, p. 012142). IOP Publishing.
6. Lu, Y., & Young, S. (2020). A survey of public datasets for computer vision tasks in precision agriculture. Computers and Electronics in Agriculture, 178, 105760.
7. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".
8. Benjamin, P., & Eliot, A. (2019). Hands-On Computer Vision with TensorFlow2.
9. Janai, J., Güney, F., Behl, A., & Geiger, A. (2020). Computer vision for autonomous vehicles: Problems, datasets and state of the art. Foundations and Trends® in Computer Graphics and Vision, 12(1–3), 1-308.
10. Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018.

11. Gollapudi, S., & Gollapudi, S. (2019). OpenCV with Python. Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs, 31-50.

12. cv2.absDiff в документації OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga6fef31bc8c4071cbc114a758a2b79c14.

13. Yulina, S. (2021). Implementation of Haar Cascade Classifier for Face Detection and Grayscale Image Transformation Using OpenCV. Jurnal Komputer Terapan, 7(1), 100-109.

14. Chadha, A., Kashyap, S., Gupta, M., & Kumar, V. (2020). License plate recognition system using OpenCV & PyTesseract. CSI Journal of Computing, 3(3), 31-35.

15. Lindblad, T., Kinser, J. M., Lindblad, T., & Kinser, J. M. (2013). NumPy, SciPy and Python Image Library. Image Processing using Pulse-Coupled Neural Networks: Applications in Python, 35-56.

16. Zhang, J., Zhang, J., Chen, B., Gao, J., Ji, S., Zhang, X., & Wang, Z. (2020, June). A perspective transformation method based on computer vision. In 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA) (pp. 765-768). IEEE.

17. Shaikh R. OpenCv Perspective Transformation [Електронний ресурс] / Raqueeb Shaikh. – 2606. – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143.

18. Devi, R. (2017). Geometric transformations and thresholding of images using Opencv-Python. GRD Journal for Engineering, 2(11).

19. Image Thresholding [Електронний ресурс] – Режим доступу до ресурсу: https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html.

20. Batubara, M. A., Alam, S., Nisa, K., Utami, R. W., Fitriawan, H., & Ulvan, A. (2021, June). Estimating the Number of Trees in Margasari Mangrove Forests of

Lampung Through Aerial Images Using Adaptive Thresholding and Contour Extraction Methods. In International Conference on Sustainable Biomass (ICSB 2019) (pp. 131-136). Atlantis Press.

21. Rehman, N. A., & Haroon, F. (2023). Adaptive Gaussian and Double Thresholding for Contour Detection and Character Recognition of Two-Dimensional Area Using Computer Vision. Engineering Proceedings, 32(1), 23.

22. O'Brien K. Airplane Image Classification using a Keras CNN [Електронний ресурс] / Kyle O'Brien. – 902. – Режим доступу до ресурсу: https://medium.com/@kylepob61392/airplane-image-classification-using-a-keras-cnn-22be506fdb53.

23. Ju, M., Luo, H., Wang, Z., Hui, B., & Chang, Z. (2019). The application of improved YOLO V3 in multi-scale target detection. Applied Sciences, 9(18), 3775.

24. Grinberg, M. (2018). Flask web development: developing web applications with python. " O'Reilly Media, Inc.".

25. Flask User’s Guide [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/.

26. Chan, J., Chung, R., & Huang, J. (2019). Python API Development Fundamentals: Develop a full-stack web application with Python and Flask. Packt Publishing Ltd.

27. Imutils Python package [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/imutils/.

28. Wegner A. How to track objects with stationary background? [Електронний ресурс] / Andre Wegner. – 606. – Режим доступу до ресурсу: https://www.authentise.com/post/how-to-track-objects-with-stationary-background.

29. BGR2GRAY в документації OpenCV [Електронний ресурс] – Режим доступу до ресурсу: http://www.devdoc.net/linux/OpenCV-doc-4.3.0/dc/d38/group__gapi__colorconvert.html#ga3e8fd8197ab16811caf9b31cb1e08a05

30. ProgramCreek. Python cv2.absdiff() Examples [Електронний ресурс] / programcreek – Режим доступу до ресурсу: https://www.programcreek.com/python/example/89428/cv2.absdiff.

31. Офіційний сайт Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/.

32. Pillow convert method doccumentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-pil-image-convert-method/.

33. Pillow putdata method doccumentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-pil-putdata-method/.

34. Devi, R. (2017). Geometric transformations and thresholding of images using Opencv-Python. GRD Journal for Engineering, 2(11).

35. Howse, J., & Minichino, J. (2020). Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning. Packt Publishing Ltd.

36. Rosebrock A. OpenCV Thresholding (cv2.threshold) [Електронний ресурс] / Adrian Rosebrock – Режим доступу до ресурсу: https://pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/.

37. Canny Edge Detection documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

38. Hammell B. Planes in Satellite Imagery [Електронний ресурс] / Bob Hammell – Режим доступу до ресурсу: https://www.kaggle.com/datasets/rhammell/planesnet.

39. Shrott R. Sequence Models by Andrew Ng — 11 Lessons Learned [Електронний ресурс] / Ryan Shrott. – 1202. – Режим доступу до ресурсу: https://towardsdatascience.com/sequence-models-by-andrew-ng-11-lessons-learned-c62fb1d3485b.

40. Canu S. Build a Sequential model | Computer Vision with Keras [Електронний ресурс] / Sergio Canu – Режим доступу до ресурсу:

https://pysource.com/2022/07/19/build-a-sequential-model-computer-vision-with-keras-p-2/.

41. CAN M. Fixed Wing UAV - Plane [Електронний ресурс] / MELIH CAN – Режим доступу до ресурсу: https://www.kaggle.com/datasets/melihcan09/fixed-wing-uav-plane.