МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ЗАСТОСУВАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ДО РОЗВ'ЯЗАННЯ ЗАДАЧ ОПТИМІЗАЦІЇ»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
М.С. Музиченко
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра прикладної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Музиченку Миколі Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Застосування генетичних алгоритмів до розв’язання задач оптимізації

керівник роботи Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Оптимізація підбору гіперпараметрів нейромережі за допомогою генетичного алгоритму.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.04.2024	
2.	Збір вихідних даних.	06.05.2024	
3.	Обробка методичних та теоретичних	24.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	12.09.2024	
5.	Розробка третього розділу.	25.10.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ М.С. Музиченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Застосування генетичних алгоритмів до розв’язання задач оптимізації»: 61 с., 27 рис., 13 джерел, 2 додатки.

ГЕНЕТИЧНИЙ АЛГОРИТМ, МУТАЦІЯ, ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ, СЕГМЕНТАЦІЯ, СХРЕЩУВАННЯ, PYTHON, YOLOV8.

Мета роботи: дослідження та аналіз сучасного застосування генетичних алгоритмів у задачах оптимізації, впровадження генетичних алгоритмів для оптимізації гіперпараметрів моделі YOLOv8 у задачі виявлення контурів будівель на супутникових зображеннях.

Об’єкт дослідження – генетичні алгоритми та моделі глибокого навчання у задачах оптимізації.

Предмет дослідження – процес оптимізації гіперпараметрів моделі YOLOv8 із використанням генетичних алгоритмів для підвищення точності виявлення контурів будівель.

Метод дослідження – аналіз, моделювання, експеримент.

Під час виконання роботи досліджено предметну область – використання генетичних алгоритмів для розв’язання задач оптимізації у складних і багатовимірних просторах рішень. У роботі розроблено інтегрований підхід, що поєднує генетичні алгоритми та сучасні моделі глибокого навчання для розв’язання задачі виявлення контурів будівель. Запропонований підхід дозволив автоматизувати налаштування гіперпараметрів моделі, що забезпечило підвищення середньої точності порівняно з базовими налаштуваннями.

Розроблений підхід забезпечує високоточне виявлення контурів будівель і оптимізацію процесу обробки великих обсягів даних.

SUMMARY

Master’s qualifying paper “Application of Genetic Algorithms to Solving Optimization Problems”: 61 p., 27 figures, 13 references, 2 supplements.

GENETIC ALGORITHM, MUTATION, HYPERPARAMETER OPTIMIZATION, SEGMENTATION, CROSSING, PYTHON, YOLOV8.

The object of the study is genetic algorithms and deep learning models in optimization tasks.

The aim of the study is to investigate and analyze the modern applications of genetic algorithms in optimization tasks and implement genetic algorithms for hyperparameter optimization of the YOLOv8 model in the task of detecting building contours on satellite images.

Research methods: analysis, simulation, experiment.

The work investigates the subject area – the use of genetic algorithms for solving optimization problems in complex and multidimensional solution spaces. The study develops an integrated approach that combines genetic algorithms with modern deep learning models to solve the problem of detecting building contours. The proposed approach automates the tuning of model hyperparameters, leading to improved average accuracy compared to baseline settings.

The developed approach ensures high-precision detection of building contours and optimizes the process of handling large data volumes.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Генетичні алгоритми та аналіз їх застосування 10
1.1 Теорія генетичних алгоритмів 10
1.2 Обмеження збіжності у генетичних алгоритмах 16
1.3 Оптимізація збіжності генетичних алгоритмів 17
1.4 Аналіз сучасного застосування генетичних алгоритмів 18
2 Проблема виявлення контуру будівлі 24
та оцінки її висоти 24
2.1 Постановка задачі 24
2.2 Дані для тренування моделі 25
2.3 Дані для тестування моделі 26
2.4 Подання заявок 27
2.5 Формат Labelme 27
2.6 Оцінювання 29
2.7 Невалідні файли 31
2.8 Скрипт для оцінювання 31
2.9 Додаткові примітки 32
3 Програмна реалізація розв’язання задачі виявлення контуру будівлі та оцінки її висоти 33
3.1 Технології розпізнавання образів 33
3.2 Використання генетичних алгоритмів для оптимізації гіперпараметрів нейронної мережі 37
3.3 Розробка програми 39
3.3.1 Підбір найкращих гіперпараметрів під час тренування 39

3.3.2 Оптимізація процесу виявлення будинків та оцінки їх висоти42
3.4 Тестування та оцінка точності роботи моделі 46
Висновки 50
Перелік посилань..... 51
Додаток А Програмна реалізація тренування моделі YOLOv8 з застосуванням генетичного алгоритму 53
Додаток Б Програмна реалізація генетичного алгоритму для оптимізації виявлення об’єктів за допомогою YOLOv8 55

ВСТУП

У сучасному світі задачі оптимізації відіграють ключову роль у багатьох галузях науки, техніки та економіки. Від ефективного розподілу ресурсів до оптимізації логістичних процесів, від проєктування інженерних конструкцій до передбачення фінансових ринків – потреба у знаходженні оптимальних рішень є надзвичайно важливою. Однак, класичні методи оптимізації не завжди здатні впоратися з цими задачами через їхню складність, великі розміри та нерегулярність простору рішень. Саме в таких умовах генетичні алгоритми (ГА) показують свою ефективність і переваги.

Моделі глибокого навчання, засновані на згорткових нейронних мережах, зарекомендували себе як потужні інструменти для задач сегментації та виявлення об’єктів на зображеннях. Задачі виявлення контурів будівель на супутникових зображеннях і визначення їхніх геометричних характеристик є важливою складовою таких застосувань. Ці задачі характеризуються високою складністю через різноманітність форм будівель, наявність тіней, перекриттів об’єктів і неоднорідність даних. Для їх ефективного вирішення необхідні точні та адаптивні підходи, здатні забезпечити високу якість сегментації в умовах великого обсягу даних і складних просторових залежностей.

Точність роботи даних моделей значною мірою залежить від правильно налаштованих гіперпараметрів. Налаштування цих параметрів вручну є трудомістким і не завжди забезпечує оптимальний результат. У таких умовах використання генетичних алгоритмів як сучасного методу оптимізації відкриває нові можливості для автоматизації цього процесу, оскільки вони дозволяють ефективно знаходити оптимальні параметри шляхом імітації природного відбору.

Метою цієї роботи є розробка та впровадження інтегрованого підходу, який поєднує можливості генетичних алгоритмів і моделей глибокого

навчання, зокрема YOLOv8, для задачі виявлення контурів будівель на супутникових зображеннях. Особливу увагу приділено оптимізації гіперпараметрів моделі за допомогою генетичних алгоритмів та постобробці результатів, що включає апроксимацію багатокутників та подавлення перекриттів будівель.

Об’єктом дослідження є генетичні алгоритми та моделі глибокого навчання у задачах оптимізації.

Предметом дослідження є застосування генетичних алгоритмів для налаштування гіперпараметрів моделі YOLOv8 та оптимізація її роботи для виявлення контурів будівель.

Актуальність теми полягає у необхідності розробки ефективних методів автоматизації аналізу супутникових зображень, що враховують високу складність і різноманітність даних. Інтеграція генетичних алгоритмів із моделями глибокого навчання дозволяє досягти вищої точності та адаптивності підходу, що робить цю технологію перспективною для реальних застосувань у багатьох сферах.

Новизна роботи полягає у поєднанні генетичних алгоритмів із сучасними моделями глибокого навчання для розв’язання специфічної задачі виявлення будівель. У рамках роботи реалізовано генетичний алгоритм для оптимізації підбору гіперпараметрів моделі YOLOv8, протестовано ефективність підходу на реальних даних і запропоновано методи постобробки для підвищення точності результатів.

У першому розділі роботи розглядаються теоретичні основи генетичних алгоритмів та сучасні підходи до їх застосування. Другий розділ присвячено задачі виявлення контурів будівель: її постановці, форматі даних та методам оцінювання результатів. У третьому розділі представлено програмну реалізацію запропонованого підходу, описано процес тренування моделі YOLOv8 з використанням генетичних алгоритмів, а також наведено результати тестування та оцінки точності роботи моделі.

1 ГЕНЕТИЧНІ АЛГОРИТМИ ТА АНАЛІЗ ЇХ ЗАСТОСУВАННЯ

1.1 Теорія генетичних алгоритмів

Задачі оптимізації займають важливе місце в багатьох галузях науки та техніки. Для їхнього розв’язання часто виникає необхідність дослідження великих і складних просторів можливих рішень, що ускладнює застосування класичних алгоритмів. У таких випадках перспективним підходом є використання генетичних алгоритмів, які ґрунтуються на принципах біологічної еволюції. Завдяки застосуванню механізмів природного відбору, мутацій та схрещування, ці алгоритми дозволяють ефективно знаходити оптимальні або близькі до оптимальних розв’язки навіть для задач із високою обчислювальною складністю.

Генетичні алгоритми належать до класу пошукових методів, які засновані на принципах природного відбору, характерних для біологічної еволюції. Їхня основна концепція полягає в поєднанні механізмів збереження найперспективніших рішень з організованим обміном інформацією, збагаченим елементами стохастичності. Цей підхід входить до категорії евристичних методів і широко використовується для задач оптимізації та моделювання. Основним інструментом у таких алгоритмах є методи випадкового генерування, комбінування та модифікації параметрів, що імітують природні процеси, такі як спадковість, мутація, природний відбір і схрещування.

Як один із різновидів еволюційних обчислень, генетичні алгоритми демонструють високу ефективність у розв’язанні складних оптимізаційних задач. Їхньою ключовою відмінністю є застосування оператора схрещування, який відповідає за рекомбінацію кандидатних рішень. Цей процес відтворює принцип біологічного схрещування, сприяючи обміну та комбінуванню генетичної інформації між елементами [2].

Представлення потенційних рішень у таких алгоритмах здійснюється у вигляді векторів – генотипів, що складаються з окремих генів. Залежно від задачі, гени можуть бути представлені у вигляді бінарних значень, чисел чи інших параметрів. У класичних реалізаціях генотипи мають фіксовану довжину, однак у деяких модифікаціях допускається її зміна.

Розрахунки починаються зі створення початкової популяції генотипів, що зазвичай генеруються випадково. Кожен генотип оцінюється за допомогою спеціальної функції пристосованості (фітнес-функції). Ця функція визначає, наскільки добре рішення, представлене генотипом, відповідає поставленій задачі. Отримані числові показники дозволяють порівнювати рішення між собою, відбираючи найкращі для подальшої еволюції.

Процес реалізації генетичного алгоритму передбачає ітеративну обробку набору можливих рішень, який у контексті цього методу називають поколіннями. На основі показника пристосованості здійснюється відбір найбільш перспективних особин. Особини з вищими значеннями пристосованості мають більші шанси бути обраними для участі в наступних етапах алгоритму.

До відібраних рішень застосовуються генетичні оператори, серед яких основними є схрещування та мутація. Схрещування забезпечує рекомбінацію ознак двох чи більше батьківських особин, створюючи нові рішення, тоді як мутація вносить випадкові зміни, сприяючи генетичному різноманіттю популяції. Для новостворених рішень також розраховується показник пристосованості. На основі цих даних проводиться повторний відбір найкращих особин, які переходять до наступного покоління [3].

Цей процес циклічно повторюється, імітуючи природний еволюційний розвиток протягом певної кількості поколінь. Алгоритм завершується, коли досягається заданий критерій зупинки. Таким критерієм може бути:

– завершення визначеної кількості поколінь;
– досягнення глобального або субоптимального рішення;
– вичерпання часу, виділеного для виконання алгоритму.

Реалізація генетичного алгоритму передбачає кілька основних етапів. Спершу визначається цільова функція пристосованості, яка оцінює якість кожного рішення у популяції. Після цього створюється початкова популяція, зазвичай сформована випадковим чином. Наступним кроком є селекція, під час якої відбираються найбільш пристосовані особини. До них застосовуються операції репродукції, такі як схрещування або мутація. Потім розраховуються нові значення функції пристосованості для всіх створених рішень.

У разі виконання критеріїв зупинки алгоритм завершується, інакше відбувається перехід до наступного покоління і весь процес повторюється.

Схематичне представлення цього циклу представлено на рисунку 1.1.

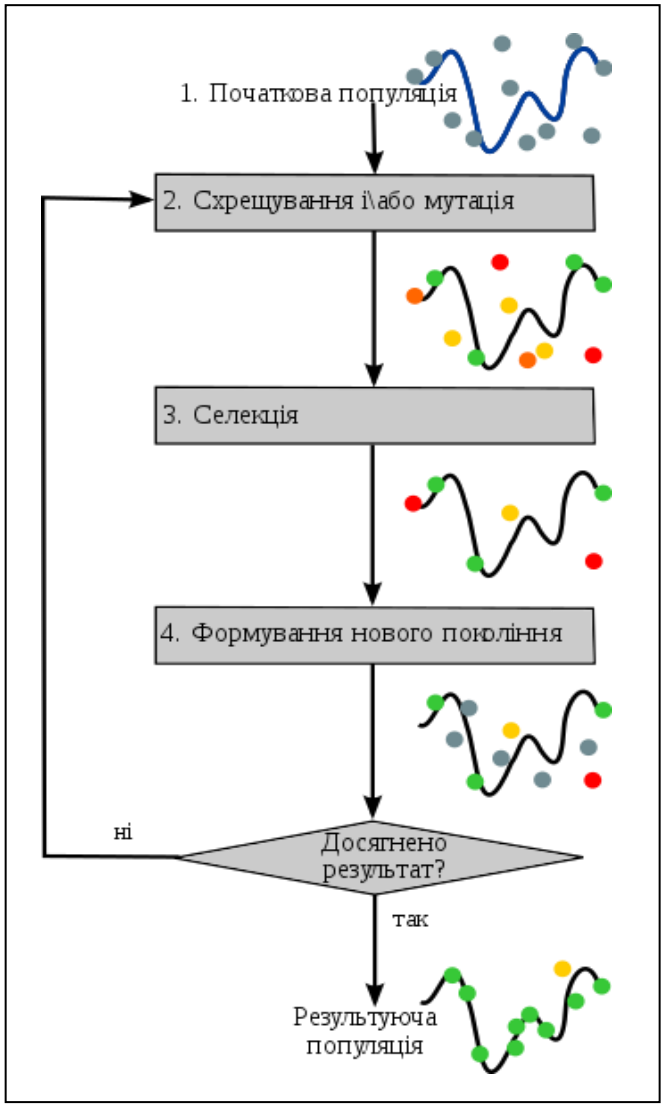


Рисунок 1.1 – Схематичне представлення роботи генетичного алгоритму

Адекватність цієї функції є критично важливою для успішного виконання алгоритму. Це означає, що її значення повинні корелювати з реальними показниками якості рішень. У разі невідповідності цільова функція може спотворювати результати пошуку, що значно ускладнює досягнення оптимального рішення. Наприклад, якщо розподіл значень функції фітнесу не відповідає реальному розподілу якості рішень, алгоритм може некоректно відбирати перспективні особини або зациклюватися на локальних екстремумах.

Правильне визначення функції фітнесу є фундаментальним етапом, який визначає успіх всього алгоритму. Вона повинна точно відображати цілі задачі та забезпечувати реалістичну оцінку рішень, стимулюючи еволюцію популяції в напрямку глобального оптимуму.

Необхідно, щоб функція фітнесу забезпечувала диференційованість оцінок значень, уникаючи обширних «рівнинних» зон. Це передбачає, що для варіативних рішень мають бути притаманні відмінні оцінки, незалежно від їх мінімальних розбіжностей. У ситуації, коли функція фітнесу призначає ідентичні оцінки значному спектру рішень, алгоритм не має змоги ідентифікувати оптимальний напрямок для подальшого розвитку. Такий стан ускладнює процес еволюції та може спричинити застій у роботі алгоритму.

Однією з проблем, що випливає з цього недоліку, є так звана «проблема поля для гольфу». У даному контексті простір пошуку є майже однорідним, окрім однієї оптимальної точки, яка відповідає глобальному розв’язку. В таких умовах генетичний алгоритм не в змозі ефективно визначити напрямок руху і, замість досягнення прогресу, або зупиняється, або переміщується хаотично, витрачаючи ресурси без досягнення значущих результатів.

Також функція фітнесу має бути оптимізована з точки зору використання ресурсів. Це критично важливо, оскільки вона є ключовим компонентом, що використовується на кожному кроці алгоритму та для кожного індивідуума у популяції. Занадто складна функція, яка потребує великих обчислювальних зусиль, може істотно знизити швидкість роботи

алгоритму, що є критичним у ситуаціях з великою кількістю популяції або ітерацій [4].

Початкова популяція. Ініціація генетичного алгоритму відбувається з формуванням первинної популяції. На цій стадії індивіди створюються за допомогою випадкового процесу так, щоб вони відповідають визначеному формату популяції та могли бути оцінені за допомогою функції пристосованості. Цей етап результує у формуванні первинної популяції, яка складається з деякої кількості індивідів N, що будуть залучені до подальшого еволюційного процесу.

Селекція. Селекція є наступним критичним етап, який визначає, які індивіди продовжать участь в еволюційному процесі. Основною метою цього етапу є забезпечення залежності імовірності виживання кожного індивіда від його функції пристосованості. Індивіди з вищими показниками пристосованості мають більшу ймовірність бути обраними для наступного покоління. Процент виживання є сталою величиною алгоритму, яка, як правило, встановлюється наперед.

Для здійснення селекції застосовуються різноманітні методи. Турнірна селекція, де випадково вибирається невелика кількість індивідів і до наступного етапу проходить той, хто має кращий показник функції пристосованості, є одним із поширених підходів. Інший відомий метод – метод рулетки, де імовірність відбору кожного індивіда пропорційна до значення його функції фітнесу, що дозволяє врахувати переваги сильніших індивідів та забезпечити можливість виживання менш пристосованих. Третій метод – це метод ранжування, де імовірність вибору залежить від позиції індивіда у відсортованому за значенням функції пристосованості списку.

Схрещування. Розмноження є ключовим етапом генетичних алгоритмів, під час якого створюються нові особини шляхом комбінування генетичного матеріалу батьків. У більшості випадків для створення нащадків залучаються двоє батьків, обраних із популяції відповідно до визначеного оператора. Основною вимогою до процесу розмноження є можливість

передачі спадкових рис обох батьків, які змішуються у певний спосіб для формування нового генотипу.

Різні оператори вибору батьків забезпечують варіативність підходів до розмноження. Один із найпростіших методів – панміксія, коли батьки обираються випадково, а всі особини мають однакові шанси стати батьками. У методі інбридингу перший батько обирається випадково, а другий – на основі максимальної схожості з першим, що сприяє закріпленню генетичних рис. Протилежний підхід, аутбридинг, передбачає, що другим батьком стає особина, яка має найменшу схожість із першим, що сприяє генетичному різноманіттю. Обидва підходи можуть бути реалізовані у фенотипній або генотипній формах: у першому випадку схожість оцінюється на основі значення функції фітнесу, а в другому – за структурою генотипу [5].

Ефективність генетичного алгоритму значною мірою залежить від збереження генетичної різноманітності у популяції. Проблема зменшення різноманітності, коли вся популяція швидко наповнюється копіями одного генотипу, є серйозною проблемою застосування ГА. Це відбувається, коли домінантний генотип, який досягає локального максимуму, витісняє всі інші варіанти. У таких випадках алгоритм втрачає здатність до подальшої еволюції.

Для боротьби з цією проблемою використовуються різні підходи. Один із них полягає у виборі для розмноження не лише найпристосованіших особин, але й менш пристосованих. Цей підхід сприяє збереженню генетичного різноманіття, однак може збільшити обчислювальну складність, оскільки вимагає підтримки великої популяції. Іншим ефективним методом є поєднання селекції з механізмом мутацій. Мутації вносять випадкові зміни до генотипу частини особин, які обираються на основі заздалегідь визначеного параметра алгоритму.

Мутації також відіграють важливу роль у підтриманні генетичного різноманіття. Вони дозволяють популяції уникати застою на локальних екстремумах, забезпечуючи додатковий простір для пошуку глобальних

оптимумів. Частка особин, до яких застосовуються мутації, а також їхній характер визначаються параметрами алгоритму, що налаштовуються залежно від конкретної задачі.

Розмноження та мутації є взаємодоповнюючими механізмами, які не лише забезпечують генерацію нових особин, але й підтримують необхідний рівень різноманітності популяції, дозволяючи алгоритму ефективно досліджувати простір рішень.

1.2 Обмеження збіжності у генетичних алгоритмах

Генетичні алгоритми, хоч і є потужним інструментом для розв’язання складних задач оптимізації, часто стикаються з проблемою збіжності до локального, а не глобального максимуму. Це означає, що алгоритм може «застрягнути» в локально оптимальному розв’язку, не досягаючи глобального екстремуму, навіть якщо такий існує.

Основна причина такої поведінки полягає у невмінні алгоритму відмовитися від короткочасних переваг, пов’язаних із високими значеннями функції фітнесу, для пошуку кращих довгострокових рішень. Іншими словами, генетичні алгоритми часто віддають перевагу стабільним локальним максимумам, замість того щоб продовжувати дослідження простору можливих розв’язків.

Ймовірність збіжності до локального максимуму значною мірою визначається формою ландшафту фітнес-функції. Деякі задачі мають чітко виражений напрямок, що веде до глобального максимуму, тоді як інші характеризуються множинними локальними максимумами, які вводять алгоритм в оману. У таких випадках фітнес-функція може «вказувати» на локальний максимум, і алгоритм не здатен вийти за межі цього обмеженого простору рішень. На рисунку 1.2 ілюструється ситуація, коли всі можливі мутації навколо локального максимуму формують область, яка стає бар’єром для подальшого прогресу алгоритму.

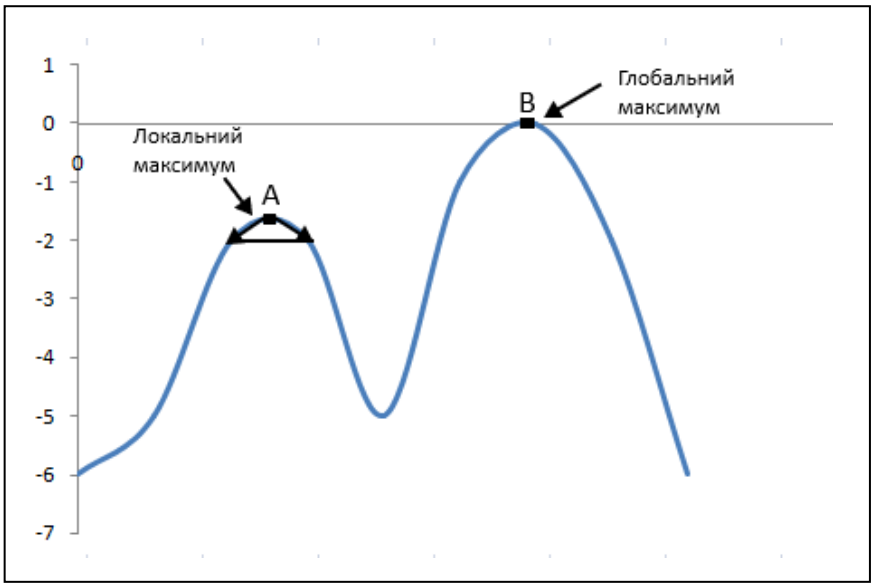


Рисунок 1.2 – Ілюстрація проблеми локального екстремуму

У таких умовах вихід за межі цієї області або вимагає значної кількості ітерацій, що суттєво знижує ефективність алгоритму, або взагалі стає неможливим за обраної конфігурації параметрів генетичного алгоритму. Це підкреслює важливість правильного налаштування механізмів мутацій і рекомбінацій для подолання подібних обмежень.

1.3 Оптимізація збіжності генетичних алгоритмів

Проблему збіжності генетичного алгоритму до локального максимуму можна спробувати вирішити за допомогою кількох підходів. Одним із ключових рішень є модифікація функції фітнесу. Використання іншої фітнес-функції, яка краще відображає глобальні властивості простору рішень, дозволяє уникати ситуацій, коли алгоритм «зациклюється» на локальних екстремумах. Наприклад, це може бути досягнуто шляхом додавання штрафів за надмірну однорідність популяції або за концентрацію рішень у певних областях простору пошуку.

Ще одним ефективним методом є збільшення ймовірності мутацій. Мутації сприяють введенню випадкових змін у генотипи, що дозволяє

популяції виходити за межі локальних максимумів. Збільшення частоти мутацій створює більше можливостей для дослідження нових напрямків пошуку, особливо коли популяція стає занадто однорідною. Адаптивні підходи, за яких ймовірність мутацій змінюється залежно від стану алгоритму, дозволяють динамічно регулювати цей процес і підтримувати баланс між дослідженням і експлуатацією.

Підтримання різноманітності рішень у популяції також є критично важливим для подолання проблеми локального максимуму. Якщо популяція стає занадто гомогенною, схрещування генів перестає приносити нові рішення, оскільки генетичний матеріал особин майже не відрізняється. У таких випадках алгоритм швидко втрачає здатність до еволюційного розвитку. Методи селекції, які спрямовані на збереження різноманітності, зокрема ранжування чи вибір менш пристосованих особин для розмноження, допомагають уникнути цієї проблеми.

Генетичні алгоритми демонструють кращу здатність долати локальні екстремуми, якщо кількість можливих переходів між станами популяції є значною. Це означає, що більше рішень доступні для дослідження на кожному етапі еволюції. Велика кількість можливих переходів дозволяє популяції уникати застою у локальних максимумах і продовжувати пошук оптимальних рішень навіть у складних задачах.

Модифікація фітнес-функції, підвищення ймовірності мутацій та селекція з підтримкою різноманітності є методами, що ефективно вирішують проблему збіжності генетичних алгоритмів.

1.4 Аналіз сучасного застосування генетичних алгоритмів

Генетичні алгоритми (ГА) залишаються важливим інструментом у сфері оптимізації завдяки своїй здатності знаходити наближені рішення для складних задач, які важко розв’язати традиційними методами. Вони можуть

досліджувати великий простір рішень, що робить їх ефективними для задач з великою кількістю локальних екстремумів, і можуть бути адаптовані до різних типів задач та обмежень, що робить їх універсальним інструментом для оптимізації. Крім того, генетичні алгоритми легко паралелізуються, що дозволяє ефективно використовувати сучасні багатоядерні процесори та розподілені обчислювальні системи.

Вибір генетичних алгоритмів для розв'язання задач оптимізації обґрунтовується їх унікальними характеристиками, які роблять їх ефективними для широкого спектра складних проблем. Розглянемо основні причини, чому генетичні алгоритми є кращими для розв'язання задач оптимізації.

Глобальна пошукова здатність. Генетичні алгоритми здійснюють глобальний пошук в просторі рішень, що дозволяє уникати проблеми застою в локальних мінімумах. Вони ефективно досліджують різні ділянки простору рішень завдяки використанню операцій кросовера та мутації, що сприяє знаходженню глобально оптимальних рішень.

Робота з великою кількістю змінних та складних функцій. ГА можуть ефективно працювати з задачами, які мають велику кількість змінних та складні, нелінійні цільові функції. Вони не вимагають знання похідних функцій і можуть оптимізувати функції, які є нерегулярними, розривними або мають багато локальних екстремумів.

Гнучкість та адаптивність. Генетичні алгоритми легко адаптуються до різних задач оптимізації, включаючи обмежені та багатокритеріальні задачі, шляхом налаштування операторів відбору, кросоверу та мутації під специфічні вимоги.

Паралельні обчислення. ГА природно підходять для паралельних обчислень, оскільки обробка окремих індивідуумів в популяції може виконуватися незалежно. Це дозволяє значно прискорити процес оптимізації, використовуючи сучасні багатоядерні процесори та розподілені обчислювальні системи.

Відсутність необхідності в детермінованій моделі. Генетичні алгоритми не вимагають детермінованої математичної моделі задачі. Вони можуть працювати з «чорною скринькою» – тобто функцією, яка повертає значення цільової функції, не розкриваючи внутрішню структуру або механізм обчислення.

Здатність до комбінування з іншими методами. ГА легко комбінуються з іншими методами оптимізації, що дозволяє створювати гібридні алгоритми. Наприклад, поєднання ГА з локальними пошуковими алгоритмами або методами машинного навчання дозволяє підвищити ефективність та швидкість конвергенції.

Прикладні аспекти використання ГА охоплюють різні галузі. У інженерії та проєктуванні вони широко використовуються для оптимізації конструкцій, включаючи аеродинамічні форми, конструкції будівель та промислових компонентів. У логістиці та транспортуванні генетичні алгоритми допомагають оптимізувати маршрути, розклади транспорту та управління запасами. В економіці та фінансах ГА використовуються для оптимізації інвестиційних портфелів, прогнозування фінансових ринків та управління ризиками. У біоінформатиці їх застосовують у задачах секвенування ДНК, прогнозування структури білків та оптимізації фармакологічних з’єднань [1].

Генетичні алгоритми також знайшли застосування у численних програмних рішеннях. Одним із популярних інструментів є програмний пакет DEAP (Distributed Evolutionary Algorithms in Python), який дозволяє створювати індивідуальні еволюційні алгоритми для широкого спектра задач. Цей пакет підтримує паралельні обчислення, що значно підвищує продуктивність. Інший приклад – бібліотека PyGAD, яка також реалізована на Python і використовується для реалізації генетичних алгоритмів у машинному навчанні, обробці зображень, оптимізації маршрутів та інших областях. PyGAD відрізняється легкістю у використанні та гнучкістю, дозволяючи користувачам швидко налаштовувати алгоритми під свої потреби.

MATLAB також пропонує вбудовані інструменти для використання генетичних алгоритмів, що робить його популярним вибором серед інженерів та дослідників. Генетичний алгоритм у MATLAB дозволяє розв’язувати задачі оптимізації з обмеженнями та без них, використовуючи готові функції та інтерфейси для налаштування параметрів алгоритму. В області промислового дизайну та виробництва інструменти як OptiStruct від Altair використовують генетичні алгоритми для топологічної оптимізації конструкцій. Цей інструмент допомагає інженерам створювати легші та більш ефективні конструкції, зменшуючи матеріальні витрати та підвищуючи міцність.

Таким чином, генетичні алгоритми залишаються потужним інструментом для розв’язання складних задач оптимізації. Сучасні дослідження спрямовані на підвищення їх ефективності та розширення областей застосування. Використання гібридних методів, квантових обчислень та інтеграції з машинним навчанням відкриває нові перспективи для їх подальшого розвитку та успішного застосування в різних галузях.

Серед додаткових прикладів програмних рішень, де використовуються генетичні алгоритми, варто згадати пакет jMetal, написаний на Java. jMetal надає багатий набір інструментів для багатокритеріальної оптимізації та підтримує різні еволюційні алгоритми, включаючи генетичні. Він широко використовується в академічних дослідженнях і практичних застосуваннях, де потрібна оптимізація з декількома цілями. Бібліотека HeuristicLab є ще одним потужним інструментом для дослідження та реалізації генетичних алгоритмів. Вона надає графічний інтерфейс, що дозволяє користувачам без глибоких знань у програмуванні налаштовувати та експериментувати з еволюційними алгоритмами. HeuristicLab також підтримує різноманітні евристичні методи, включаючи генетичні алгоритми, і може бути використана для задач у різних галузях, таких як логістика, машинне навчання, та оптимізація виробничих процесів.

У практичних додатках, таких як планування ресурсів підприємства (ERP) та управління ланцюгами поставок (SCM), генетичні алгоритми

допомагають оптимізувати процеси та зменшити витрати. В медицині, зокрема у персоналізованій медицині та геноміці, ГА застосовуються для аналізу великих обсягів біологічних даних, що допомагає у визначенні ефективних стратегій лікування [1].

Щодо інтеграції генетичних алгоритмів з сучасними технологіями, варто зазначити, що їх поєднання з методами глибокого навчання дозволяє оптимізувати архітектуру нейронних мереж, покращувати якість прогнозів і підвищувати продуктивність систем штучного інтелекту. Зокрема, такі підходи використовуються в автономних системах, де необхідно швидко та ефективно адаптувати моделі до змін у навколишньому середовищі.

Одним із головних аспектів ефективного використання ГА є правильний вибір параметрів. Вибір параметрів, таких як розмір популяції, ймовірність кросовера та мутації, стратегія відбору, значно впливає на ефективність алгоритму. Неправильний вибір параметрів може призвести до повільної конвергенції або застою в локальних мінімумах. Хоча ГА мають здатність знаходити глобальні оптимуми, вони можуть повільно сходитися, що є проблемою для задач, які потребують швидкого рішення. Для дуже великих задач кількість можливих рішень може стати занадто великою навіть для ГА, що потребує додаткових методів для зменшення обчислювальної складності [2].

Сучасні тенденції та дослідження включають поєднання генетичних алгоритмів з іншими методами оптимізації, такими як локальні пошукові алгоритми, методи рою часток або машинне навчання. Це дозволяє покращити ефективність та швидкість конвергенції. ГА використовуються як частина більш складних метаевристичних методів, таких як алгоритми пошуку на основі розподілу та еволюційні стратегії. Ведуться дослідження щодо використання квантових обчислень для реалізації генетичних алгоритмів, що може значно прискорити процес оптимізації. Інтеграція глибокого навчання з генетичними алгоритмами для задач оптимізації, наприклад, для оптимізації архітектури нейронних мереж, також є перспективним напрямком.

Отже, генетичні алгоритми, завдяки своїй гнучкості та ефективності, знаходять застосування у найрізноманітніших галузях, від інженерії та виробництва до біоінформатики та фінансів. Сучасні тенденції та дослідження, спрямовані на покращення їхніх можливостей, забезпечують нові шляхи для вирішення складних задач оптимізації, роблячи генетичні алгоритми незамінним інструментом у багатьох сферах науки і техніки.

2 ПРОБЛЕМА ВИЯВЛЕННЯ КОНТУРУ БУДІВЛІ ТА ОЦІНКИ ЇЇ ВИСОТИ

2.1 Постановка задачі

Завданням даної роботи є розробка й оптимізація моделі для виявлення контуру будівель та оцінки їх висоти для завдання з контесту з машинного навчання, представленого на платформі Algotester [6].

Algotester – сучасна платформа для організації змагальних заходів з програмування. Сайт регулярно проводить конкурси з алгоритмізації як для студентів, так і для професіоналів. На сайті міститься великий архів завдань із завданнями різної складності, які можна використовувати для вивчення алгоритмів та практики кодування. Algotester підтримує змагання у стилі ICPC, а також IOI та завдання з оптимізації, схожі на марафон.

Algotester спільно з Huawei провели контест з машинного навчання, що проходив з 25 травня по 10 червня 2024 року. Індивідуальним учасникам було запропоновано розв’язати одну задачу машинного навчання протягом 16 днів змагання. Сторінку контесту представлено на рисунку 2.1.

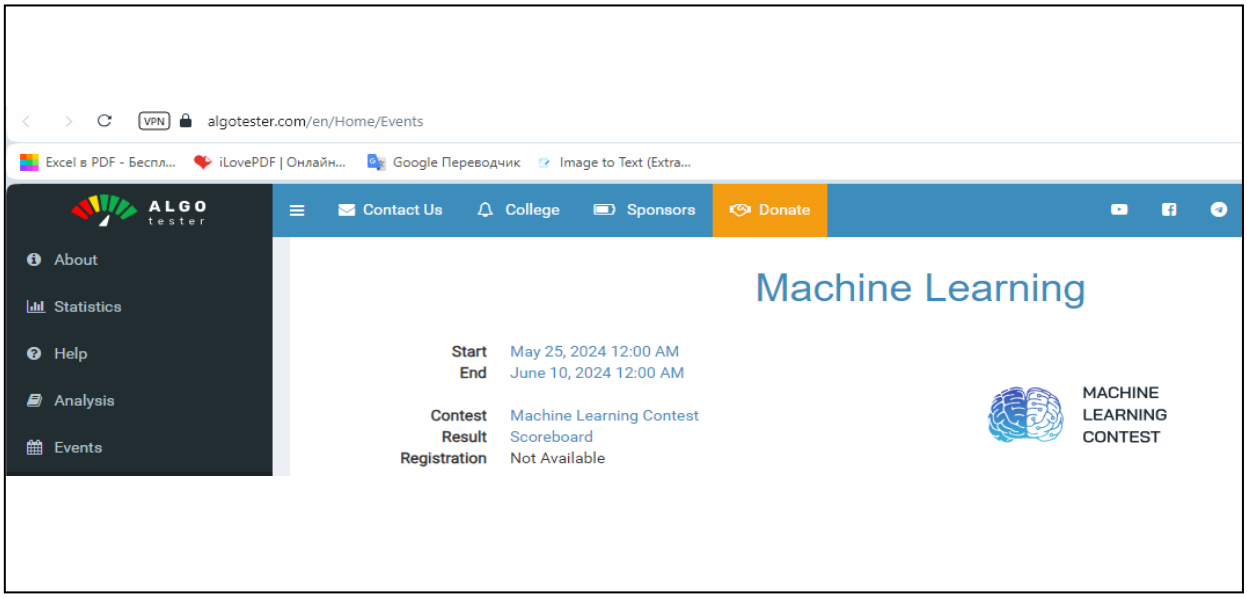


Рисунок 2.1 – Сторінка контесту

Найкращі виконавці були нагороджені грошовими винагородами й отримали від 200 доларів США для місць з 6 по 10 за загальним заліком до 3000 доларів США переможцю контесту.

Будинки є найпомітнішими рукотворними структурами на Землі. Точне визначення контурів будівель для виявлення та прогнозування їх висоти має вирішальне значення для багатьох застосувань, таких як міське планування, бездротова інфраструктура, оцінка чисельності населення тощо.

Наразі найточніша інформація про контур та висоту будівлі отримується за допомогою даних LIDAR або багаторакурсних зображень з використанням щільного зіставлення. Однак отримати такі дані важко.

Визначення контуру будівель та оцінка їх висоти за допомогою монокулярного супутникового зображення є гарячою темою в галузі дистанційного зондування останніми роками. У цьому контесті учасникам надаються монокулярні супутникові зображення та дані міток з інформацією про контури та висоту будівель, щоб знайти хороший алгоритм виділення контурів будівель та оцінки їх висоти.

2.2 Дані для тренування моделі

Датасет для тренування містить набір із 2200 монокулярних супутникових зображень Землі, на кожному з яких зображено принаймні одну будівлю.

Для кожного зображення в наборі навчальних даних надається файл JSON у форматі LabelMe, який містить контури та висоту для всіх будівель на зображенні.

Архів тренувальних даних містить дві папки: *images* і *ground_truth_files*. Перший містить зображення у форматі *png*, а другий містить файли даних у форматі *json*.

Зображення та відповідний файл даних мають однакову назву. Кожне зображення має точний розмір 512×512 пікселів.

2.3 Дані для тестування моделі

Тестові дані містять набір із 4444 монокулярних супутникових зображень Землі, на кожному з яких є принаймні одна будівля.

Архів тестових даних містить зображення у форматі png. Кожне зображення має точний розмір 512×512 пікселів.

Приклад правильно розміченого зображення представлено на рисунку 2.2.



Рисунок 2.2 – Приклад зображення з тренувальних даних

Дані для тестування внутрішньо розділені на дві групи: попередні та остаточні. Під час змагання учасники можуть побачити свій рахунок лише для попереднього набору. Після завершення конкурсу будуть оголошені результати остаточного набору даних.

Учасникам невідомо, яке зображення відповідає попередньому чи остаточному набору, тому, щоб подати заявку, потрібно оцінити весь тестовий набір даних, який містить як попередній, так і остаточний набори.

2.4 Подання заявок

Заявка учасника є архівом *zip* із спрогнозованими даними для набору тестових даних.

Кожен файл в архіві повинен мати таку саму назву, як одне із зображень із тестового набору даних, розширення json і містити спрогнозовані за допомогою моделі дані для будівель у відповідному зображенні у форматі *LabelMe*. Наприклад, дані для зображення *aaaeexkyyo.png* мають називатися *aaaeexkyyo.json*.

Архів повинен містити лише описані вище файли. Учасники не повинні створювати папки або включати будь-які зображення в архів.

2.5 Формат LabeLme

LabelMe – це кросплатформне програмне забезпечення з відкритим вихідним кодом, яке використовується для візуалізації та створення даних полігональних міток на мові *Python*. Більш докладні відомості про формат даних і інсталяцію програмного забезпечення можна отримати на сторінці *GitHub* [7].

Приклад роботи графічного інструменту анотації зображень *LabelMe* представлено на рисунку 2.3.

Щоб ваше подання було правильно оцінено, поле *shapes* кореневого об’єкта *JSON* має містити список усіх будівель. Поле точок кожного будинку має містити список вершин багатокутника, які представляють контур будівлі. Поле *group_id* має містити висоту будівлі в метрах.

На рисунку 2.4 наведено приклад допустимого файлу з двома будівлями: перша має форму трикутника та висоту дев’ять метрів; другий має чотири вершини та висоту сім метрів.

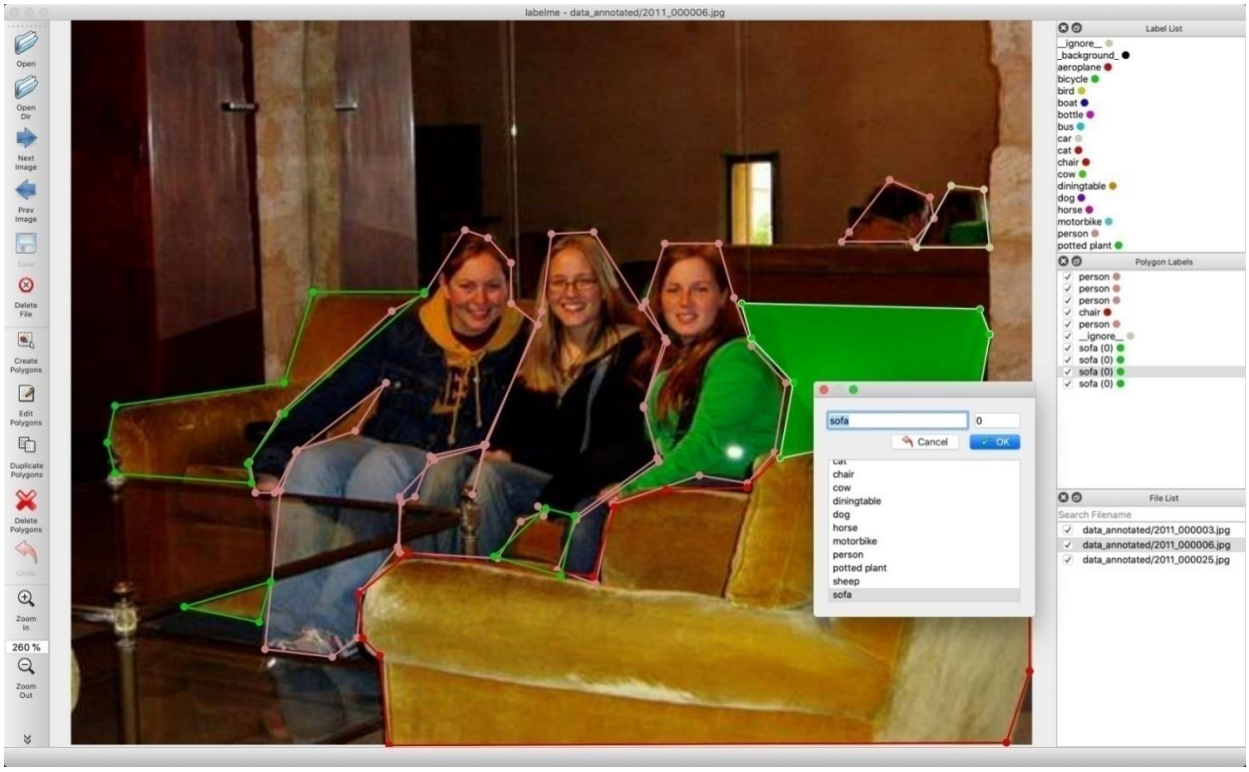


Рисунок 2.3 – Приклад роботи з LabelMe


```
{
  "shapes": [
    {
      "points": [
        [ 316, 486 ],
        [ 307, 510 ],
        [ 312, 512 ]
      ],
      "group_id": 9
    },
    {
      "points": [
        [ 416, 457 ],
        [ 435, 446 ],
        [ 421, 423 ],
        [ 402, 434 ]
      ],
      "group_id": 7
    }
  ]
}
```


Рисунок 2.4 – Приклад допустимого файлу з двома будівлями

2.6 Оцінювання

Оцінка для подання учасників контесту розраховується як поєднання трьох метрик: точності (Precision Score), повноти (Recall Score) та оцінки висоти (Height Score):

$$Score = max(0, \lfloor (Precision\ Score + Recall\ Score - 4 \cdot Height\ Score) \times \times 5 \cdot 10^4 \rfloor). \quad (2.1)$$

Задача учасників – максимізувати цей бал.

Оцінка точності (Precision Score). Точність визначається як частка площі будівель у вихідних файлах учасника, які правильно розміщені, до загальної площі знайдених будівель:

$$Precision\ Score = \frac{Correctly\ Placed\ Area}{Total\ Area\ In\ Output} \cdot 100. \quad (2.2)$$

Будівля A з ваших вихідних файлів вважається правильно розміщеною, якщо загальна площа перетину полігону будівлі A з усіма будівлями з еталонних даних для відповідного зображення перевищує половину площі полігону будівлі A.

Якщо у вихідних файлах немає будівель, тобто

$$Total\ Area\ In\ Output = 0, \quad (2.3)$$

то оцінка точності буде 0.

Оцінка повноти (Recall Score). Повнота визначається як частка площі будівель в еталонних даних, які правильно ідентифіковані, до загальної площі будівель в еталонних даних:

$$Recall\ Score = \frac{Correctly\ Identified\ Area}{Total\ Area\ In\ Ground\ Truth} \cdot 100. \quad (2.4)$$

Будівля *B* з еталонного файлу вважається правильно ідентифікованою, якщо загальна площа перетину полігону будівлі *B* з усіма будівлями з відповідного вихідного файлу перевищує половину площі полігону будівлі *B*.

Оцінка висоти (Height Score). Оцінка висоти визначається як квадратний корінь з середньозваженого середнього квадрата різниць висот усіх пар будівель, що збіглися:

$$Height\ Score = \sqrt{\frac{\sum_{a;b:matched}\ (height(a) - height(b))^2 \cdot weight(a,b)}{Total\ Weight}}. \qquad (2.5)$$

Парні будівлі діляться на **три групи**.

Зіставлені пари. Будівля A з ваших вихідних даних вважається зіставленою з будівлею B з еталонних даних, якщо площа перетину будівель A і B становить щонайменше половину площі будівлі A або щонайменше половину площі будівлі B:

$$area(intersection(A, B)) \geq \frac{\min\ (area(a), area(B))}{2}. \qquad (2.6)$$

Вага такої пари вважається рівною площі перетину двох полігонів.

Зверніть увагу, що будівля може бути зіставлена з більш ніж однією іншою будівлею.

Незіставлені будівлі з вихідних даних. Кожна будівля з ваших вихідних даних, яка не зіставлена з жодною будівлею з еталонних файлів, вважається зіставленою з будівлею висотою 0. Вага такої пари вважається рівною площі будівлі.

Незіставлені будівлі з еталонних даних. Кожна будівля з еталонних даних, яка не зіставлена з жодною будівлею з ваших вихідних даних, вважається зіставленою з будівлею висотою 0. Вага такої пари вважається рівною площі будівлі.

2.7 Невалідні файли

Файл в архіві, що відповідає одному із зображень з тестового набору даних, вважається невалідним, якщо:

– файл не є валідним JSON-файлом;
– файл не відповідає формату LabelMe;
– хоча б один із полігонів має самоперетини;
– кількість будівель перевищує 1000;
– кількість вершин у полігоні будь-якої будівлі перевищує 300;
– загальна кількість вершин у файлі перевищує 5000;
– висота будь-якої будівлі є від'ємною або перевищує 1000;
– координата будь-якої точки полігону будівлі є від'ємною або перевищує 512;
– існують дві будівлі A і B, такі що площа їхнього перетину перевищує 10% від площі меншої з будівель A або B:

$$area(intersection(A, B)) \geq 0.1 \cdot min(area(A), area(B)); \qquad (2.7)$$

– скрипт оцінювання не може коректно зчитати або обробити файл з будь-якої іншої причини.

Кожен невалідний файл вважається скриптом оцінювання порожнім, тобто таким, що не містить будівель. Для додаткових уточнень звертайтеся до скрипту оцінювання. Посилання на скрипт наведене на сайті [8].

2.8 Скрипт для оцінювання

Скрипт, який використовується для оцінювання, доступний для завантаження за посиланням [8]. Учасники можуть використовувати його для локального тестування своїх результатів на навчальному наборі даних.

2.9 Додаткові примітки

Розглянемо додаткові примітки умов контесту:

- зображення для навчального, попереднього та фінального наборів даних обираються випадковим чином без будь-яких упереджень;
- джерело даних із мітками є однаковим для навчального та тестового наборів;
- мітки створені вручну сторонньою компанією та можуть містити невелику частину некоректних даних;
- контур будівлі визначається як відбиток будівлі на землі, однак, у випадку малих будівель контур може іноді включати дах будівлі;
- у деяких файлах з еталонними даними контури будівель можуть мати невеликі зміщення порівняно з їхнім розташуванням на зображеннях – це зміщення може бути не однаковим для всіх будівель на зображенні;
- невелика частина еталонних файлів може містити суттєве зміщення контурів будівель;
- деякі еталонні файли можуть містити накладені або некоректні мітки;
- розмір подання учасників не повинен перевищувати 15 МБ;
- розмір розпакованих даних не повинен перевищувати 256 МБ;
- учасники можуть подавати свої результати раз на 6 годин та отримувати зворотний зв’язок із балом для попереднього набору даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ ВИЯВЛЕННЯ КОНТУРУ БУДІВЛІ ТА ОЦІНКИ ЇЇ ВИСОТИ

3.1 Технології розпізнавання образів

Комп'ютерний зір – це область штучного інтелекту, яка спрямована на отримання, аналіз і розуміння візуальної інформації з реального світу, представленої у вигляді зображень або відео. Основна мета комп'ютерного зору – автоматизація процесів, які люди виконують за допомогою зору, таких як розпізнавання об'єктів, виявлення руху, аналіз сцени та інші.

Розпізнавання образів є важливою частиною комп'ютерного зору і зосереджується на ідентифікації певних об'єктів або шаблонів у зображеннях. Задачі розпізнавання образів включають:

– класифікацію зображень;
– виявлення об'єктів (визначення положення і класу об'єкта);
– сегментацію об'єктів (розподіл зображення на області, які належать конкретним об'єктам) [9].

Розпізнавання об'єктів – це процес виявлення та ідентифікації об'єктів на зображенні. Методи розпізнавання поділяються на класичні алгоритми та підходи, що базуються на глибокому навчанні.

Розв'язання задачі, сформованої у попередньому розділі, полягає у виявленні будівель і поданні їх меж у вигляді багатокутників, це можна розглядати як задачу векторизації об'єктів на основі детекції та контурного аналізу. У цьому випадку задача належить до кількох пов'язаних напрямків у комп'ютерному зорі та машинному навчанні.

Основна задача – виявлення багатокутників, потрібно визначити межі будівель у вигляді багатокутників, які відображають точні контури будівель. Це спрощена версія сегментації з додатковим етапом постобробки.

Дана задача не є повною мірою задачею сегментації, оскільки на відміну від повної семантичної сегментації, де кожен піксель класифікується,

тут акцент робиться на детекції об’єкта (визначенні рамок будівель) та спрощенні контурів (перетворенні складних форм у багатокутники).

Етапи формування багатокутників контуру будинку:

– детекція об’єктів: будівлі спочатку виявляються як рамки за допомогою детекторів образів на зображенні;
– контурний аналіз (спрощена сегментація): в межах рамки виконується локальна сегментація, щоб визначити точну форму будівлі – використовуються методи виділення;
– апроксимація контурів: виділений контур перетворюється в багатокутник зі зменшеною кількістю вершин – цей етап допомагає перетворити нерівні або «зашумлені» контури в чіткі багатокутники;
– побудова геометричних моделей: визначені багатокутники можуть бути представлені у векторному форматі (наприклад, GeoJSON, Shapefile) для подальшого використання в ГІС або інших системах.

Можливі підходи до розв’язання задачі. Дану задачу можна вирішити за допомогою класичних методів комп’ютерного зору, таких як використання алгоритмів Canny або Sobel для виділення контурів та, наприклад, Алгоритм Рамера-Дугласа-Пекера для апроксимації кривих. Дані підходи добре працюють для висококонтрастних зображень, але не підходять для складних випадків (наприклад, тіні, низька якість) [10].

Найбільш сучасним й ефективним методом розв’язання представленої проблеми є використання методів на основі глибокого навчання. В задачах розпізнавання та сегментації зображень зазвичай використовуються згорткові нейронні мережі (Convolutional Neural Networks, CNN).

CNN – це тип штучних нейронних мереж, спеціально розроблений для аналізу зображень. Їх основною характеристикою є здатність автоматично вивчати просторові й локальні ознаки з даних.

Основними компонентами CNN є згортковий шар (Convolutional Layer), шар підвибірки (Pooling Layer), шар активації (Activation Layer) та повнозв’язний шар (Fully Connected Layer).

Згортковий шар використовує фільтри (ядра) для обчислення локальних ознак, таких як контури, текстури, форми. Цей шар ідентифікує патерни в невеликих областях зображення.

Шар підвибірки виконує зменшення розмірності, обираючи найбільш значущі ознаки (наприклад, max-pooling або average-pooling), зменшуючи кількість необхідних обчислень і запобігаючи перенавчанню.

Шар активації використовує нелінійні функції активації, такі як ReLU, Sigmoid або Tanh, для врахування складних залежностей.

Повнозв’язний шар виконує класифікацію або інші завдання на основі виділених ознак [11].

Перевагами CNN є автоматичне виділення ознак, локальність і просторові зв’язки та ієрархія ознак, що значно спрощують процес обробки зображень, покращують точність розпізнавання та дозволяють ефективно працювати з великими наборами даних.

CNN використовуються для задач сегментації та детекції через їх гнучкість у масштабах, універсальність та сучасні архітектури для детекції та сегментації, що поєднують швидкість і точність завдяки вдосконаленим шарам і оптимізаціям.

Для розв’язання поставленої задачі логічно використати сучасні алгоритми комп’ютерного зору, зокрема:

– детектори об’єктів (YOLO, Faster R-CNN) для швидкого та точного визначення об’єктів у зображеннях;
– Instance Segmentation (Mask R-CNN, YOLOv8) для сегментації окремих об’єктів з їх точним контуром;
– Polygonal Prediction Models (Poly-YOLO) для прогнозування багатокутних контурів, що особливо корисно для задач, де важлива геометрія об’єктів.

Можливі труднощі. При проєктуванні засобів виявлення будинків на зображеннях виникають труднощі, пов’язані з точністю отриманих контурів, оскільки багатокутники часто мають «шумні» або нерівні лінії, особливо у

випадках складних будівель. Перетворення складних контурів у багатокутники може призводити до втрати важливих деталей, що негативно впливає на якість результату. Додаткову складність створюють зовнішні фактори, такі як тіні, перекриття об’єктів і низька роздільна здатність зображень, які ускладнюють формування чітких і точних багатокутників.

YOLOv8. Для розв’язання поставленої задачі будемо використовувати модель з відкритим кодом YOLO (You Only Look Once) від Ultralytics, що поширюється за ліцензією AGPL-3.0 (GNU Affero General Public License) [12].

YOLOv8 є сучасною моделлю комп’ютерного зору, розробленою для вирішення задач виявлення, сегментації та класифікації об’єктів на зображеннях. Завдяки використанню удосконалених згорткових нейронних мереж, модель забезпечує високу точність і швидкість обробки.

YOLOv8 підтримує інстансну сегментацію, яка дозволяє виділяти межі будівель із високою точністю. Модель адаптована до роботи з різними масштабами об’єктів, що робить її ефективною навіть для будівель складної геометрії. Завдяки anchor-free підходу YOLOv8 здатна коректно обробляти об’єкти з нерегулярними формами, а її гнучкість дозволяє адаптуватися до специфічних задач, наприклад, навчатися на наборах даних, створених із супутникових зображень. Це дає змогу виявляти будівлі різного розміру та форми, включно з тими, які мають значні варіації через освітлення або тіні.

Ще однією перевагою YOLOv8 є її оптимізація для роботи в реальному часі. Вона швидше за більші моделі та потребує менше ресурсів, що особливо актуально для використання в умовах обмежених обчислювальних потужностей.

У поєднанні з сучасними алгоритмами CNN, YOLOv8 забезпечує точну локалізацію об’єктів навіть у складних умовах, таких як перекриття будівель, тіні або низька роздільна здатність. Її одностадійна архітектура дозволяє уникати тривалих обчислювальних процесів, забезпечуючи швидке й ефективне рішення задач виявлення. Таким чином, YOLOv8 є ідеальним

вибором для задач аналізу супутникових зображень, де важливі точність, швидкість і гнучкість підходу.

3.2 Використання генетичних алгоритмів для оптимізації гіперпараметрів нейронної мережі

Генетичні алгоритми (ГА) є еволюційним методом оптимізації, який моделює процес природного відбору (див. розділ 1.1).

У розділі 1.4 розглянуто сучасні підходи до використання генетичних алгоритмів, зокрема інтеграцію з методами глибокого навчання. Такий симбіоз демонструє значний потенціал для розв’язання задач оптимізації, наприклад, у налаштуванні архітектури нейронних мереж. Цей напрямок стає одним із ключових у сучасному машинному навчанні завдяки своїй здатності автоматизувати процес вибору оптимальних параметрів, що забезпечує підвищення ефективності моделювання та адаптації до специфічних задач [13].

ГА можна ефективно застосовувати для налаштування гіперпараметрів нейронних мереж, таких як YOLOv8, зокрема для поставленої нами задачі виявлення будівель на супутникових зображеннях. Враховуючи складність налаштування параметрів моделі вручну, ГА забезпечують автоматизований підхід до пошуку оптимальних налаштувань, які можуть значно підвищити точність і продуктивність моделі.

У даній роботі генетичний алгоритм застосовується для налаштування гіперпараметрів моделі YOLOv8 для задачі сегментації, що дозволяє автоматизувати процес пошуку оптимальних налаштувань, таких як швидкість навчання (learning rate), імпульс (momentum) та регуляризація (weight decay), завдяки ітераційному еволюційному підходу.

Під час тренування нейромережі будемо створювати покоління «рішень» (наборів гіперпараметрів), які поступово покращуються завдяки

операціям мутації та відбору. Процес розпочинається з одного початкового набору гіперпараметрів, який у наступних поколіннях видозмінюється, породжуючи нові комбінації параметрів.

Ключовим моментом є функція клонування та мутації, яка створює «дитячі» набори гіперпараметрів із «батьківських» шляхом внесення невеликих випадкових змін у межах заданого діапазону. Це забезпечує різноманітність у популяції. Особливу увагу приділено параметру “momentum”, значення якого обмежується від 0 до 1, щоб уникнути нестабільності під час навчання.

На кожному поколінні всі активні рішення (набори гіперпараметрів) використовуються для тренування моделі YOLOv8. Для цього застосовується заздалегідь підготовлений датасет (див. розділ 2.2). Навчання виконується з використанням певної кількості епох, а також фіксованих параметрів, таких як розмір батчу, розмір зображення та тип оптимізатора. Після тренування для кожного набору параметрів обчислюється метрика точності mAP50-95(M), яка відображає якість сегментації моделі.

По завершенню тренування покоління відбувається сортування наборів гіперпараметрів за значенням метрики точності, після чого зберігаються лише найкращі. Це гарантує, що наступне покоління будується на основі найкращих «батьківських» рішень, поступово покращуючи якість параметрів.

Повний код програми для тренування YOLO з застосуванням генетичного алгоритму представлено у додатку А. Результати роботи програми на тестових даних проаналізовані у розділі 3.4.

Загалом, цей підхід ілюструє, як ГА можуть бути інтегровані у процес тренування YOLOv8 для автоматичної оптимізації параметрів. Метод дозволяє мінімізувати ручну працю, забезпечуючи ефективний і гнучкий пошук оптимальних налаштувань для складних задач комп’ютерного зору, таких як виявлення контурів будівель на супутникових зображеннях.

3.3 Розробка програми

3.3.1 Підбір найкращих гіперпараметрів під час тренування

Відкриваємо середовище розробки та імпортуємо потрібні бібліотеки (рис. 3.1). Deepcopy використовується для створення копії батьківського рішення при мутації, random застосовується для внесення випадкових змін (мутацій) до параметрів, а ultralytics – бібліотека для роботи з моделями YOLO.

```
from copy import deepcopy
import random

from ultralytics import YOLO
```

Рисунок 3.1 – Імпорт бібліотек

Задаємо глобальні параметри генетичного алгоритму (рис. 3.2):

– *MUTATION_RANGE* – визначає діапазон випадкових змін (мутацій) для гіперпараметрів;
– *CHILDREN_COUNT* – кількість «нащадків», які створюються від кожного рішення в поточному поколінні;
– *SOLUTIONS_TO_KEEP* – кількість найкращих рішень, що зберігаються для наступного покоління;
– *GENERATIONS* – кількість ітерацій (поколінь), які виконує алгоритм;
– *EPOCHS* – кількість епох тренування моделі.

```
MUTATION_RANGE = [0.9, 1.1]
CHILDREN_COUNT = 1
SOLUTIONS_TO_KEEP = 3
GENERATIONS = 10
EPOCHS = 10
```

Рисунок 3.2 – Глобальні параметри генетичного алгоритму

Задаємо функцію мутації (рис. 3.3), що створює «нащадка» на основі «батьківського» рішення, змінюючи його параметри у випадкових межах, та початкову популяцію (рис. 3.4).

```python
def clone_and_mutate(parent):
    child = deepcopy(parent)

    for key in child:
        if key == "map":
            continue
        mutation = random.uniform(MUTATION_RANGE[0], MUTATION_RANGE[1])
        child[key] *= mutation

    child["map"] = None

    # Clamp momentum to ensure that parameter updates are not going to explode
    child["momentum"] = min(1.0, max(0.0, child["momentum"]))

    return child
```

Рисунок 3.3 – Функція мутації

```python
starting_point = {
    "lr": 0.001,
    "momentum": 0.9,
    "weight_decay": 0.0005,
    "map": None
}

solutions = [starting_point]
```

Рисунок 3.4 – Початкова популяція

Для кожного наявного рішення створюємо нових нащадків шляхом мутації. Повторюємо задану кількість поколінь (рис. 3.5).

```python
for generation in range(GENERATIONS):
    solutions_count = len(solutions)
    for i in range(solutions_count):
        for j in range(CHILDREN_COUNT):
            child = clone_and_mutate(solutions[i])
            solutions.append(child)
```

Рисунок 3.5 – Створення нових нащадків шляхом мутації

Тренуємо модель для кожного рішення (рис. 3.6). Якщо для рішення ще не обчислена метрика точності моделі, виконується тренування моделі.

YOLOv8 завантажується з попередньо натренованими вагами. Використовується заданий датасет *(/content/config.yaml*), кількість епох (*EPOCHS*) та такі гіперпараметри, як швидкість навчання (*learning rate*), імпульс (*momentum*) та регуляризація (*weight decay*). Після тренування обчислюється точність (*mAP*) для масок на валідаційному наборі, і значення зберігається в рішенні.

```
for solution in solutions:
    if solution["map"] is not None:
        continue

    model = YOLO('yolov8m-seg.pt')  # load a pretrained model (recommended for training)
    model.train(
        data='/content/config.yaml',
        batch=64,
        epochs=EPOCHS,
        imgsz=512,
        optimizer="AdamW",
        lr0=solution["lr"],
        momentum=solution["momentum"],
        weight_decay=solution["weight_decay"],
    )

    metrics = model.val()
    solution["map"] = metrics.results_dict["metrics/mAP50-95(M)"]
```

Рисунок 3.6 – Тренування моделі для кожного рішення

Проводимо відбір, сортуючи рішення від найвищого до найнижчого за значенням map і зберігаючи найкращі для наступного покоління (рис. 3.7).

```
solutions = sorted(solutions, key=lambda x: x["map"], reverse=True)
solutions = solutions[:SOLUTIONS_TO_KEEP]
```

Рисунок 3.7 – Відбір найкращих індивідів з популяції

Для наочності процесу роботи програми виводимо поточний стан найкращих рішень (рис. 3.8).

```
print(f"Generation {generation + 1}:")
for solution in solutions:
    print(solution)
```

Рисунок 3.8 – Виведення найкращих індивідів у даному поколінні

Загалом, даний код реалізує тренування моделі YOLOv8 з використанням генетичного алгоритму для оптимізації гіперпараметрів. Основна мета – автоматизувати налаштування таких параметрів, як швидкість навчання (*lr*), імпульс (*momentum*) і коефіцієнт регуляризації (*weight_decay*), щоб досягти максимальної точності роботи моделі (*mAP*). Повний код програми представлено у Додатку А.

Експериментально визначивши найкращі гіперпараметри роботи моделі, переходимо до етапу оптимізації процесу виявлення будинків та оцінки їх висоти.

3.3.2 Оптимізація процесу виявлення будинків та оцінки їх висоти

Відкриваємо середовище розробки та імпортуємо потрібні бібліотеки (рис. 3.9). Імпортуються бібліотеки для роботи з зображеннями (*cv2*), масивами чисел (*numpy*), багатокутниками (*shapely)*, моделі *YOLOv8*, прогрес-барами (*tqdm*) та скрипт для оцінки результатів за умовами контесту (*scorer*).

```
from copy import deepcopy
import json
import os
import random

import cv2
import numpy as np
from shapely.geometry import Polygon
from tqdm import tqdm
from ultralytics import YOLO

from scorer import main as scorer_main
```

Рисунок 3.9 – Імпорт бібліотек

Задаємо параметри генетичного алгоритму (рис. 3.10): *MUTATION_RANGE* визначає, як сильно змінюватимуться гіперпараметри, *CHILDREN_COUNT*, *SOLUTIONS_TO_KEEP*, *GENERATIONS* задають кількість нових нащадків, найкращих збережених рішень і поколінь генетичного алгоритму відповідно.

```
MUTATION_RANGE = [0.9, 1.1]
CHILDREN_COUNT = 2
SOLUTIONS_TO_KEEP = 10
GENERATIONS = 20
```

Рисунок 3.10 – Глобальні параметри генетичного алгоритму

Завантажуємо попередньо натреновану YOLOv8 та шляхи до датасетів, створюємо список зображень для аналізу, створюємо папку для збереження результатів (рис. 3.11).

```
# Load a model
model = YOLO("./best.pt")  # load a custom model

target_dir = "./mlc_eval_images"
ground_truth_dir = "./mlc_eval_ground_truth"
results_dir = "./results_eval"

if not os.path.exists(results_dir):
    os.makedirs(results_dir)

images = os.listdir(target_dir)
images = [str(os.path.join(target_dir, image)) for image in images]
```

Рисунок 3.11 – Завантаження моделі та датасетів зображень

Задаємо функцію апроксимації контурів (рис. 3.12), що зменшує кількість вершин багатокутника для спрощення подання та відповідності обмеженням. Для апроксимації контурів використовується метод *cv2.approxPolyDP*. Задаємо функцію фільтрації *non_max_suppression*. Фільтруємо багатокутники, які перекриваються, і видаляємо нерелевантні чи некоректні полігони.

```python
# Function to approximate contours and reduce the number of vertices
def approximate_contour(contour, epsilon_factor=0.01):
    epsilon = epsilon_factor * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)

    if approx is None:
        return None

    return approx.squeeze(1)
```

Рисунок 3.12 – Функція апроксимації контурів

Видаляємо перекриття між багатокутниками методом, модифікованим на основі IoU (Intersection over Union). Додаємо нові полігони, якщо розділення багатокутників створює додаткові, що задовольняють критерії. Пропускаємо дуже малі чи невалідні багатокутники.

В основному блоці виконуємо сегментацію контурів будівель на кожному зображенні (рис. 3.13).

```python
def run(target_conf, min_area_ratio):

    # Predict with the model
for image_path in tqdm(images):
    results = model.predict(image_path, imgsz=512, conf=target_conf, verbose=False)  # predict on an image

    result = results[0]
    output_name = os.path.join(results_dir, os.path.basename(image_path).replace(".png", ".json"))

    if result.masks is None or len(result.masks) == 0:
        # Skip if no buildings found
        output_data = {
            "shapes": []
        }

        with open(output_name, "w") as f:
            json.dump(output_data, f)
        continue
```

Рисунок 3.13 – Основний блок сегментації зображень та збереження результатів

Видаляємо зайві вершини та полігони, які не задовольняють обмеженням. Багатокутники конвертуються у JSON-формат для подальшого використання чи оцінки.

Виконуємо оцінку якості роботи моделі при даних параметрах, викликаючи зовнішній модуль для оцінки точності передбачення на основі порівняння з еталонною розміткою (рис. 3.14).

```
return scorer_main(ground_truth_dir, results_dir)
```

Рисунок 3.14 – Оцінки точності передбачення

Для оптимізації гіперпараметрів будемо використовувати генетичний алгоритм. Генетичний алгоритм оптимізує такі гіперпараметри:

– target_conf – поріг довіри для виявлення об’єктів;

– min_area_ratio – мінімальна площа багатокутника відносно початкової площі.

Задаємо початкову популяцію (рис. 3.15). Початкова популяція складається з одного набору гіперпараметрів.

```
starting_point = {
    "target_conf": 0.05,
    "min_area_ratio": 0.5,
    "score": None,
}

solutions = [starting_point]
```

Рисунок 3.15 – Початкова популяція

У кожному поколінні створюємо нові набори гіперпараметрів шляхом мутації (рис. 3.16). Мутація випадковим чином змінює параметри у межах заданого діапазону *MUTATION_RANGE*.

```
def clone_and_mutate(parent):
    child = deepcopy(parent)

    for key in child:
        if key == "score":
            continue
        mutation = random.uniform(MUTATION_RANGE[0], MUTATION_RANGE[1])
        child[key] *= mutation

    child["score"] = None

    return child
```

Рисунок 3.16 – Функція клонування та мутації

Для кожного набору параметрів запускаємо функцію *run*, яка виконує передбачення та оцінку роботи моделі. Сортуємо рішення за якістю роботи, найкращі рішення зберігаємо для наступного покоління (рис. 3.17).

```
for generation in range(GENERATIONS):
    solutions_count = len(solutions)
    for i in range(solutions_count):
        for j in range(CHILDREN_COUNT):
            child = clone_and_mutate(solutions[i])
            solutions.append(child)

    for solution in solutions:
        if solution["score"] is not None:
            continue

        solution["score"] = run(solution["target_conf"], solution["min_area_ratio"])

    solutions = sorted(solutions, key=lambda x: x["score"], reverse=True)
    solutions = solutions[:SOLUTIONS_TO_KEEP]

    print(f"Generation {generation + 1}:")
    for solution in solutions:
        print(solution)
```

Рисунок 3.17 – Реалізація основного циклу генетичного алгоритму

Загалом дана програма автоматизує оптимізацію гіперпараметрів для виявлення будівель на зображеннях. Вона знаходить контури будівель, оптимізує поріг довіри та мінімальне співвідношення площі полігонів через генетичний алгоритм, усуває помилкові контури та оцінює точність результатів порівняно з еталоном. В результаті роботи отримуються найкращі параметри для точного виявлення будівель.

3.4 Тестування та оцінка точності роботи моделі

Проведемо тестування моделі YOLOv8 для оцінки її точності у задачі виявлення будівель на супутникових зображеннях. У ході тестування використовується навчена в ході роботи модель, яка була отримана на етапі

тренування із застосуванням генетичного алгоритму. Мета тестування полягає у вимірюванні продуктивності моделі, аналізі впливу оптимізованих гіперпараметрів на точність та оцінці результатів постобробки. Для оцінки використовувався тестовий набір супутникових зображень, який включає розмічені контури будівель. Датасет складався із зображень для оцінки і відповідної еталонної розмітки, що була надана авторами завдання. Набір включає будівлі різних розмірів, форм і розташувань, зокрема випадки з перекриттям об’єктів і тінями.

Передбачення виконувалися на кожному зображенні тестового набору з використанням оптимізованих генетичним алгоритмом гіперпараметрів.

Постобробка результатів включала видалення дрібних і невалідних багатокутників, подавлення перекриттів та зменшення кількості вершин контурів для спрощення подання.

Для оцінки точності використовувалися стандартні метрики, такі як IoU (перекриття між передбаченим і еталонним багатокутником) та середнє значення mAP50-95 для масок (усереднена точність за різними порогами IoU від 0.5 до 0.95), розраховано значення оцінки точності Precision та оцінки повноти Recall та отримано наступні значення:

а) без використання генетичного алгоритму для оптимізації:

Precision Score = 93.954,

Recall Score = 81.756,

Final Score = 4 633 892;

б) з використанням генетичного алгоритму:

Precision Score = 87.435,

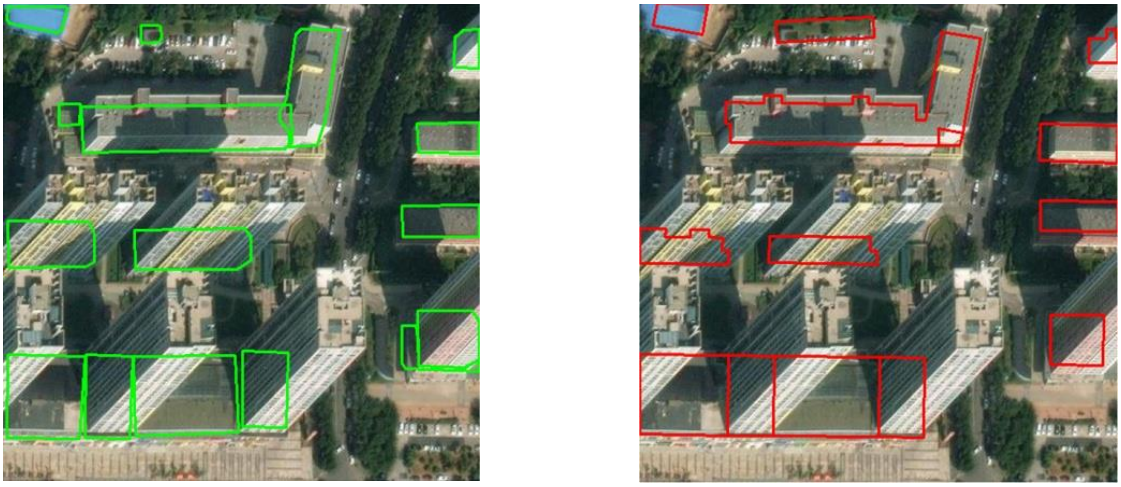
Recall Score = 93.413,

Final Score = 5 418 447.

Результати оцінки показують, що використання генетичного алгоритму у даному випадку покращило підсумковий результат на 17%, що свідчить про значну його роль як у рамках даної задачі, так і загалом для оптимізації параметрів моделей глибокого навчання.

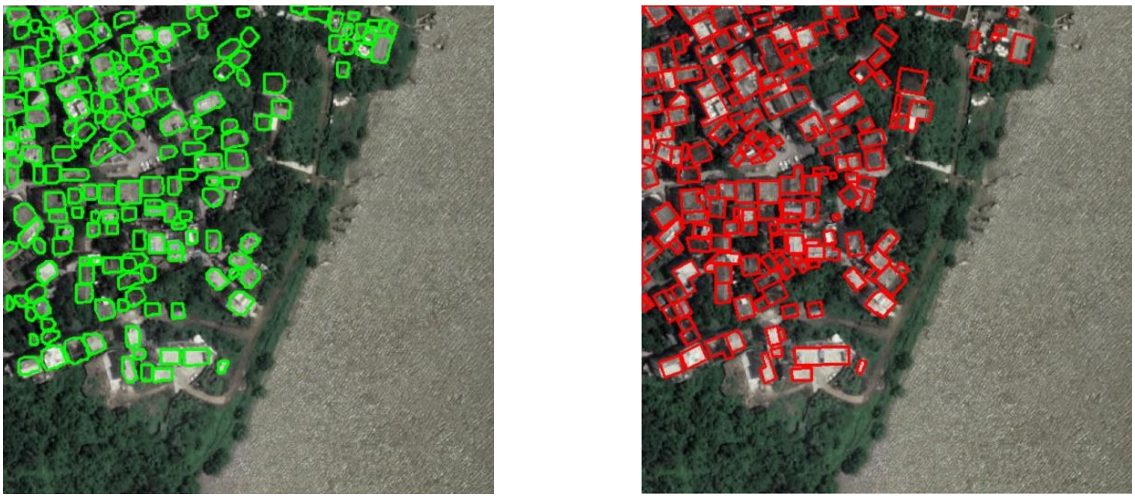
За результатами тестування модель демонструвала високу точність для об’єктів із чіткими межами та стабільну продуктивність на більшості тестових зображень. Застосування постобробки, включно з подавленням перекриттів і зменшенням складності багатокутників, дозволило відповідати обмеженням щодо форми контурів, при цьому зберігаючи загальну точність.

На рисунках 3.18 та 3.19 показано результат виявлення та виділення будинків навченою в ході роботи моделлю у порівнянні з еталонною розміткою на простих і більш складних супутникових зображеннях.



а) Результат роботи моделі б) Еталонна розмітка

Рисунок 3.18 – Оцінка роботи моделі на тестовому зображенні порівняно з еталонною розміткою



а) Результат роботи моделі б) Еталонна розмітка

Рисунок 3.19 – Оцінка роботи моделі на складному зображенні

Точність розробленої моделі є високою, а разом з оптимізацією генетичним алгоритмом вона дозволила б потрапити до списку призерів наведеного контесту.

Тестування підтвердило, що оптимізована модель YOLOv8 є ефективним інструментом для виявлення будівель на супутникових зображеннях, демонструючи високу точність і практичну придатність для складних задач аналізу зображень.

ВИСНОВКИ

У ході проведеного дослідження було детально розглянуто використання генетичних алгоритмів у задачах оптимізації на прикладі оптимізації гіперпараметрів моделі YOLOv8 у задачах виявлення будівель на супутникових зображеннях.

Генетичні алгоритми продемонстрували високу ефективність у підборі оптимальних для задачі гіперпараметрів моделі, дозволяючи автоматизувати процес їх пошуку. Це суттєво зменшує потребу в ручному налаштуванні та забезпечує стабільність результатів.

Проведене тестування показало, що оптимізація за допомогою генетичних алгоритмів забезпечила значне підвищення (на 17%) підсумкової оцінки роботи моделі порівняно з базовими налаштуваннями.

У роботі реалізовано гнучкий підхід до роботи з неоднорідними наборами даних, такими як супутникові зображення. Було розроблено алгоритм для постобробки результатів виявлення будівель, який включає апроксимацію контурів з подавленням перекриттів між будівлями на зображеннях.

Інтеграція ГА в процес оптимізації моделі YOLOv8 спростила процес навчання і зменшила час, необхідний для досягнення якісних результатів, забезпечивши стабільну продуктивність і точність моделі.

Таким чином, використання генетичних алгоритмів дозволяє покращити точність і стабільність роботи моделі YOLOv8 у складних задачах.

Результати роботи демонструють, що інтеграція генетичних алгоритмів у процес навчання та оптимізації моделей глибокого навчання має значний потенціал. Поєднання теоретичних аспектів та практичних розробок у даній роботі дозволяє зрозуміти можливості та переваги генетичних алгоритмів у розв’язанні сучасних задач оптимізації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Троцько В. В. Методи штучного інтелекту : навчально-методичний і практичний посібник. Київ : Університет економіки та права «КРОК», 2020. 86 с.

2. Poli R., Langdon W. B., McPhee N. F. A Field Guide to Genetic Programming. Raleigh : Lulu Enterprises, 2008. 233 p.

3. Holland J. H. Adaptation in Natural and Artificial Systems. Ann Arbor : University of Michigan Press, 1992. 232 p.

4. Ішин С. Самопрограмування : еволюціонуючі алгоритми, самопрограмування та самомодифікація програм. Київ : Техніка, 1997. 28 с.

5. Srinivas N., Deb K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*. 1994. № 12. P. 221–248.

6. Building Contour Detection and Height Estimation Problem. *Algotester*. URL: https://algotester.com/en/ContestProblem/DisplayWithFile/135254 (дата звернення: 08.07.2024).

7. LabelMe (Image Polygonal Annotation with Python). *GitHub*. URL: https://github.com/wkentaro/labelme (дата звернення: 12.07.2024).

8. Скрипт оцінювання задачі конкурсу : електрон. ресурс. URL: https://drive.google.com/file/d/1TRI344ScJMCRTM8v4qJ62xGG7FoMJMKT/view?pli=1 (дата звернення: 12.07.2024).

9. Кромкач В. Роль комп’ютерного зору в сучасному світі: досягнення, виклики та перспективи. *Information Technology: Computer Science, Software Engineering and Cyber Security*. 2024. № 2. С. 79–87.

10. Гонсалес Р., Вудс Р. Цифрова обробка зображень / пер. з англ. П. А. Чочіа, Л. І. Рубанов: 3-тє вид., випр. і допов. Київ : Техносфера, 2019. 1104 с.

11. Glorot X., Bordes A., Bengio Y. Deep Sparse Rectifier Neural Networks. *Fourteenth International Conference on Artificial Intelligence and Statistics* :

Proceedings (Ft. Lauderdale, Florida, USA, April 11–13, 2011). Fort Lauderdale, 2011.

12. YOLOv8. *Ultralytics*. URL: https://docs.ultralytics.com/models/yolov8/ (дата звернення: 25.09.2024).

13. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989. 412 p.

ДОДАТОК А

Програмна реалізація тренування моделі YOLOv8 з застосуванням генетичного алгоритму

```
### 4. Train model ###

from copy import deepcopy
import random

from ultralytics import YOLO

MUTATION_RANGE = [0.9, 1.1]
CHILDREN_COUNT = 1
SOLUTIONS_TO_KEEP = 3
GENERATIONS = 10
EPOCHS = 10

def clone_and_mutate(parent):
    child = deepcopy(parent)

    for key in child:
        if key == "map":
            continue
        mutation = random.uniform(MUTATION_RANGE[0], MUTATION_RANGE[1])
        child[key] *= mutation

    child["map"] = None

    # Clamp momentum to ensure that parameter updates are not going to
explode
    child["momentum"] = min(1.0, max(0.0, child["momentum"]))

    return child

starting_point = {
    "lr": 0.001,
    "momentum": 0.9,
    "weight_decay": 0.0005,
    "map": None
}

solutions = [starting_point]
```

```
for generation in range(GENERATIONS):
    solutions_count = len(solutions)
    for i in range(solutions_count):
        for j in range(CHILDREN_COUNT):
            child = clone_and_mutate(solutions[i])
            solutions.append(child)

    for solution in solutions:
        if solution["map"] is not None:
            continue

        model = YOLO('yolov8m-seg.pt')  # load a pretrained model
(recommended for training)
        model.train(
            data='/content/config.yaml',
            batch=64,
            epochs=EPOCHS,
            imgsz=512,
            optimizer="AdamW",
            lr0=solution["lr"],
            momentum=solution["momentum"],
            weight_decay=solution["weight_decay"],
        )

        metrics = model.val()
        solution["map"] = metrics.results_dict["metrics/mAP50-95(M)"]

    solutions = sorted(solutions, key=lambda x: x["map"], reverse=True)
    solutions = solutions[:SOLUTIONS_TO_KEEP]

    print(f"Generation {generation + 1}:")
    for solution in solutions:
        print(solution)
```

ДОДАТОК Б

Програмна реалізація генетичного алгоритму для оптимізації виявлення об’єктів за допомогою YOLOv8

```
from copy import deepcopy
import json
import os
import random

import cv2
import numpy as np
from shapely.geometry import Polygon
from tqdm import tqdm
from ultralytics import YOLO

from scorer import main as scorer_main

MUTATION_RANGE = [0.9, 1.1]
CHILDREN_COUNT = 2
SOLUTIONS_TO_KEEP = 10
GENERATIONS = 20

# Load a model
model = YOLO("./best.pt")  # load a custom model

target_dir = "./mlc_eval_images"
ground_truth_dir = "./mlc_eval_ground_truth"
results_dir = "./results_eval"

if not os.path.exists(results_dir):
    os.makedirs(results_dir)

images = os.listdir(target_dir)
images = [str(os.path.join(target_dir, image)) for image in images]

# Warmup
model.predict(images[0], imgsz=512, conf=0.05, verbose=False)

# Function to approximate contours and reduce the number of vertices
def approximate_contour(contour, epsilon_factor=0.01):
    epsilon = epsilon_factor * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)
```

```
    if approx is None:
        return None

    return approx.squeeze(1)

def check_bounding_box_intersection(building_a, building_b):
    a_min_x, a_min_y, a_max_x, a_max_y = building_a.bounds
    b_min_x, b_min_y, b_max_x, b_max_y = building_b.bounds
    if a_max_x < b_min_x or a_min_x > b_max_x or a_max_y < b_min_y or
a_min_y > b_max_y:
        return False
    else:
        return True

def non_max_suppression(polygons, classes, iou_threshold=0.08,
min_area_ratio=0.5):
    # This function is slightly modified version of the original IoU, as
it divides by min area of two polygons, not
    # the area of the union of two polygons

    # If polygons overlap - subtract the one with higher confidence
(comes earlier in the array) from the one with the lower conf.
    # This sometimes leads to polygon splitting into multiple, so we add
them back to the array
    # But not more than 2x the original size.
    max_polygons = len(polygons) * 2

    orig_area = [polygon.area for polygon in polygons]

    supressed = np.zeros(max_polygons, dtype=bool)

    # for(int j = 0; j < len(polygons); j++)
    j = -1
    while j < len(polygons) - 1:
        j += 1
        if supressed[j]:
            continue

        # for(int i = j - 1; i >= 0; i--)
        i = j
        while i > 0:
            i -= 1
            if supressed[i]:
                continue
```

```
            if not check_bounding_box_intersection(polygons[i],
polygons[j]):
                continue

            intersection = polygons[i].intersection(polygons[j]).area
            if intersection / min(polygons[i].area, polygons[j].area) <=
iou_threshold:
                continue

            # Subtract the polygon[i] from the polygon[j]
            polygons[j] = polygons[j].difference(polygons[i])

            if polygons[j].geom_type == "MultiPolygon":
                supressed[j] = True

                class_id = classes[j]
                orig_area_j = orig_area[j]

                for polygon in polygons[j].geoms:
                    if len(polygons) >= max_polygons:
                        break

                    # Skip the polygon if it's too small
                    if polygon.area <= 10:
                        continue

                    # Skip the polygon if it's area is below half of the
area of the original polygon that produced this one
                    if polygon.area / orig_area_j < min_area_ratio:
                        continue

                    polygons.append(polygon)
                    classes.append(class_id)
                    orig_area.append(orig_area[j])

                break

            # Skip the polygon if it's too small
            if polygons[j].area <= 10:
                supressed[j] = True
                break

    # Prune invalid polygons
    for i in range(len(polygons)):
```

```
        if not polygons[i].is_valid or not polygons[i].is_simple or
polygons[i].is_empty or polygons[i].area <= 10:
            supressed[i] = True
            continue
        if polygons[i].geom_type != "Polygon":
            supressed[i] = True
            continue
        if polygons[i].length < 10:
            supressed[i] = True
            continue
        if polygons[i].area / orig_area[i] < min_area_ratio:
            supressed[i] = True
            continue
        # Prune polygons that are too thin
        if polygons[i].area / polygons[i].length < 2.0:
            supressed[i] = True
            continue

    # Make all coords int
    for i in range(len(polygons)):
        if not supressed[i]:
            polygons[i] = Polygon([[int(x), int(y)] for x, y in
polygons[i].exterior.coords])

    # Prune again, never enough pruning
    for i in range(len(polygons)):
        if not polygons[i].is_valid or not polygons[i].is_simple or
polygons[i].is_empty or polygons[i].area <= 10:
            supressed[i] = True
            continue
        if polygons[i].geom_type != "Polygon":
            supressed[i] = True
            continue

    # Second IOU pass
    for i in range(len(polygons)):
        if supressed[i]:
            continue
        for j in range(i + 1, len(polygons)):
            if supressed[j]:
                continue
            if not check_bounding_box_intersection(polygons[i],
polygons[j]):
                continue
```

```
            intersection = polygons[i].intersection(polygons[j]).area
            if intersection / min(polygons[i].area, polygons[j].area) >
iou_threshold:
                supressed[j] = True

    return [polygons[i] for i in range(len(polygons)) if not
supressed[i]], [classes[i] for i in range(len(polygons)) if not
supressed[i]]

def run(target_conf, min_area_ratio):
    # target_conf is minimum confidence for bounding boxes, this value
showed best balance between precision and recall on validation set

    # Predict with the model
    for image_path in tqdm(images):
        results = model.predict(image_path, imgsz=512, conf=target_conf,
verbose=False)  # predict on an image

        result = results[0]
        output_name = os.path.join(results_dir,
os.path.basename(image_path).replace(".png", ".json"))

        if result.masks is None or len(result.masks) == 0:
            # Skip if no buildings found
            output_data = {
                "shapes": []
            }

            with open(output_name, "w") as f:
                json.dump(output_data, f)
            continue

        masks = result.masks.cpu().xy
        masks = tuple(mask.astype(np.int32) for mask in masks)
        classes = result.boxes.cls.cpu().numpy()

        # Approximate the contours, because original ones have too many
verices and don't fit into the contest constraints
        contours = [approximate_contour(contour) for contour in masks]

        # Convert contours into shapely Polygons, prune invalid ones
        polygons_classes = [(Polygon(contour), class_id) for contour,
class_id in zip(contours, classes) if contour is not None and
len(contour) >= 3]
```

```
        polygons_classes = [(polygon, class_id) for (polygon, class_id)
in polygons_classes if polygon.is_valid and polygon.is_simple and not
polygon.is_empty]

        polygons = [polygon for polygon, class_id in polygons_classes]
        classes = [class_id for polygon, class_id in polygons_classes]

        polygons, classes = non_max_suppression(polygons, classes,
min_area_ratio=min_area_ratio)

        shapes = []
        for polygon, class_id in zip(polygons, classes):
            shape = {}
            shape["group_id"] = int(class_id * 3 + 3)
            shape["points"] = [[int(x), int(y)] for x, y in
polygon.exterior.coords]
            shapes.append(shape)

        output_data = {
            "shapes": shapes
        }

        with open(output_name, "w") as f:
            json.dump(output_data, f, indent=4)

    return scorer_main(ground_truth_dir, results_dir)


starting_point = {
    "target_conf": 0.05,
    "min_area_ratio": 0.5,
    "score": None,
}

solutions = [starting_point]

def clone_and_mutate(parent):
    child = deepcopy(parent)

    for key in child:
        if key == "score":
            continue
        mutation = random.uniform(MUTATION_RANGE[0], MUTATION_RANGE[1])
        child[key] *= mutation
```

```
    child["score"] = None

    return child

for generation in range(GENERATIONS):
    solutions_count = len(solutions)
    for i in range(solutions_count):
        for j in range(CHILDREN_COUNT):
            child = clone_and_mutate(solutions[i])
            solutions.append(child)

    for solution in solutions:
        if solution["score"] is not None:
            continue

        solution["score"] = run(solution["target_conf"],
solution["min_area_ratio"])

    solutions = sorted(solutions, key=lambda x: x["score"], reverse=True)
    solutions = solutions[:SOLUTIONS_TO_KEEP]

    print(f"Generation {generation + 1}:")
    for solution in solutions:
        print(solution)
```