МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИЯВЛЕННЯ РУЙНУВАНЬ НА ДОРОГАХ»**

Виконав: студент 2 курсу, групи 8.1262
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
Є. С. Чернов
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, професор, д.т.н., С. В. Чопоров
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, доцент, к.ф.-м.н, О. В. Кудін
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра інформаційних технологій

Рівень вищої освіти магістр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та штучний інтелект

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, д.т.н., професор

_____ Шило Г.М.
(підпис)

« 5 » травня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Чернов Євгеній Сергійович
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка нейронної мережі для виявлення руйнувань на дорогах

керівник роботи (проекту) С. В. Чопоров, д.т.н., професор,
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 29.11.2023

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.05.23

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ _____
(підпис) (ініціали та прізвище)

Керівник роботи _____ _____
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ _____
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка нейронної мережі для виявлення руйнувань на дорогах»: 50 с., 5 рис., 2 табл., 15 джерел, 3 додатки.

АЛГОРИТМ, НЕЙРОННА МЕРЕЖА, MASK R-CNN, TENSORFLOW, PYTHON, ЗОБРАЖЕННЯ, МЕТОД, СИСТЕМА, МОДЕЛЬ.

Об’єкт дослідження – системи та алгоритми розпізнавання облич.

Мета роботи: Розробка нейронної мережі, що дозволяє розпізнавати руйнування на дорогах на статичних зображеннях з використанням алгоритму Mask R-CNN, який поєднує в собі здатність виявляти об'єкти та генерувати сегментаційні маски для цих об'єктів.

Метод дослідження – аналітичний.

Робота присвячена розробці системи нейронної мережі для виявлення руйнувань на дорогах за статичними фотознімками. В якості методу локалізації руйнування на зображенні в даній роботі був обраний алгоритм Mask R-CNN. Для виділення ознак для виявлення та сегментації об'єктів використовується базова мережа, така як ResNet.

В рамках роботи проведено дослідження ефективності різних алгоритмів для виявлення та розпізнавання об’єктів на зображеннях, яке показало високу ефективність сегментації та деталізації об’єктів стосовно до задачі розпізнавання вибоїн на дорогах в реальному часі.

SUMMARY

Master’s Qualification Theses “Development of a neural network for detecting destructions on roads”: 50 pages, 5 figures, 2 tables, 15 references, 3 attachments.

ALGORITHM, NEURAL NETWORK, MASK R-CNN, TENSORFLOW, PYTHON, IMAGE, METHOD, SYSTEM, MODEL.

Object of research – face recognition systems and algorithms.

Purpose of thesis: Development of a neural network that allows recognition of road damage on static images using the Mask R-CNN algorithm, which combines the ability to detect objects and generate segmentation masks for these objects.

The research method – analytical.

The thesis is concerned with the development of a neural network system for detecting road damage from static photographs. The Mask R-CNN algorithm was chosen as the method of localization of destruction on the image in this work. An underlying network such as ResNet is used to extract features for object detection and segmentation.

The effectiveness study of various modifications of various algorithms for the detection and recognition of objects in images was carried out, which showed the high efficiency of segmentation and detailing of objects in relation to the task of recognizing potholes on roads in real time.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 1
Summary 2
Вступ 6
1 Огляд систем і алгоритмів 8
1.1 Огляд існуючих систем розпізнавання 8
1.2 Сучасні системи розпізнавання руйнувань 8
1.2.1 RoadCrackNet 8
1.2.2 YOLO-based Road Damage Detection 10
1.2.3 Pothole Detection System 12
1.3 Недоліки сучасних систем розпізнавання 14
2 Огляд алгоритмів та методів виявлення та розпізнавання руйнувань на дорогах 16
2.1 Алгоритми виявлення руйнування на зображенні 17
2.1.1 Faster R-CNN 18
2.1.2 YOLO (You Only Look Once) 18
2.1.3 SSD (Single Shot MultiBox Detector) 19
2.1.4 RetinaNet 19
2.2 Алгоритми розпізнавання руйнування на дорозі 21
2.2.1 Convolutional Neural Networks (CNN) 21
2.2.2 U-Net 21
2.2.3 Mask R-CNN 22
2.2.4 DeepLab 22
2.3 Вибір найкращого алгоритму для реалізації 24
3 Етапи обробки статичних зображень 26
3.1 Умови застосування обраного методу 26
3.2 Алгоритмічний опис обраного методу 28

4 Розробка нейронної мережі розпізнавання руйнувань 30
4.1 Інструментарій розробки 30
4.2 Аналіз потоків даних 32
4.3 Підготовка наборів даних 32
4.4 Опис структури 35
4.5 Тестування розробленої мережі 38
Висновки 42
Перелік посилань 43
Додаток А Приклад вигляду JSON файлу зображення 45
Додаток Б Файл pothole.py – основний файл класу PotholeDataset 46
Додаток В Файл config.py – файл конфыгурації 49

ВСТУП

Дорожнє покриття відіграє важливу роль в забезпеченні безпечної та комфортної їзди для автомобілів. Проте, внаслідок зношеності, погіршення якості будівельних матеріалів, недостатнього фінансування для ремонту та інших факторів, дорожнє покриття може піддаватися руйнуванням. Це може призвести до нещасних випадків на дорогах та збитків для транспортних компаній та держави. В Індії щороку близько 1100 життів гинуть внаслідок нещасних випадків, спричинених вибоїнами. Звичайні громадяни не мають можливості повідомити про погані дороги відповідною владою, тоді як влада не знає про ситуацію.

У зв’язку з цим, виявлення руйнувань на дорогах та своєчасне їх усунення є актуальною проблемою. До складних завдань у цьому напрямку належить виявлення найменших дефектів, які не заважають безпечному руху, проте з часом можуть призвести до серйозних проблем.

Одним з можливих рішень для виявлення руйнувань на дорогах є застосування машинного навчання та нейронних мереж. Це дає змогу автоматизувати процес виявлення руйнувань та забезпечити високу точність та швидкість роботи мережі.

Метою цієї дипломної роботи є розробка нейронної мережі для виявлення руйнувань на дорогах. Для досягнення цієї мети будуть використані методи машинного навчання та обробки зображень. При цьому будуть проаналізовані існуючі системи та підходи до виявлення руйнувань на дорогах та їхні переваги та недоліки.

Додатково можна зазначити, що нейронні мережі дозволяють отримати значну точність виявлення руйнувань на дорогах за рахунок їх здатності до автоматичного вивчення характерних ознак на зображеннях. Також слід зазначити, що розробка нейронної мережі для виявлення руйнувань на дорогах

може бути корисною для підтримки безпеки на дорогах та покращення якості ремонту доріг.

Отже, розробка ефективної системи виявлення руйнувань на дорогах є важливим завданням, яке може зменшити кількість нещасних випадків на дорогах та витрати на ремонт дорожнього покриття.

1 ОГЛЯД СИСТЕМ І АЛГОРИТМІВ

1.1 Огляд існуючих систем розпізнавання

Декілька організацій намагалися розробити інструменти (наприклад, веб-додатки), за допомогою яких громадяни могли б повідомляти про вибоїни відповідній владі. Було проведено кілька хакатонів, однією з цілей яких була ця задача [1]. Однак все ж таки зручніше прибрати людський фактор перегляду інформації та оцінки руйнувань, та спростити розпізнавання вибоїн завдяки технології комп’ютерного зору та нейронних мереж.

На сьогоднішній день існує кілька систем розпізнавання руйнувань на дорогах, які використовуються для виявлення та моніторингу пошкоджень на дорожньому покритті, які використовують різні алгоритми, такі як Faster R-CNN, SSD, YOLO і Mask R-CNN [2]. Системи забезпечують високий відсоток розпізнавання і можуть використовуватися спільно з системами контролю і управління доступом (СКУД) для збільшення рівня контролю доступу на об'єктах з підвищеними вимогами до підвищення безпеки. На сьогоднішній день дослідження в цій області продовжуються, зокрема з використанням технологій комп'ютерного зору та нейронних мереж.

1.2 Сучасні системи розпізнавання руйнувань

1.2.1 RoadCrackNet

RoadCrackNet – це система детектування та класифікації тріщин на дорожньому покритті за допомогою глибокого навчання [3]. Вона використовує дві архітектури нейронних мереж: DeepCrack і RoadNet. DeepCrack – це глибока ієрархічна архітектура для сегментації тріщин, яка

використовує конволюційні шари, рекурентні шари та атеншн-механізм для видобутку ознак тріщин на різних масштабах.

RoadNet – це мультитаск-архітектура для одночасного виявлення центральних ліній дороги та сегментації тріщин, яка використовує спільну енкодер-декодер структуру та два паралельних головних шари для кожного завдання.

RoadCrackNet використовує такі фільтри:

– гауссовий фільтр для зменшення шуму на зображеннях, собелевський фільтр для виділення границь тріщин;
– фільтр Канні для отримання бінарних зображень тріщин;
– фільтр Хафа для виявлення прямих ліній на зображеннях.

Мінімальними вимогами до зображень є:

– роздільна здатність 640x480 пікселів;
– формат JPEG;
– кольоровий режим RGB;
– яскраве освітлення;
– чисте дорожнє покриття без інших об'єктів.

Функціональними характеристиками системи є:

– точність сегментації тріщин 93.4%;
– точність детектування центральних ліній дороги 96.7%;
– швидкодія 25 кадр/сек;
– можливість працювати на CPU та GPU;
– можливість адаптації до різних умов освітлення та дорожнього покриття.

Система RoadCrackNet може складатися з декількох етапів для розпізнавання та виявлення руйнувань на дорогах:

– підготовка даних:
  – збір та підготовка датасету, що містить зображення доріг з руйнуваннями;
  – розбиття датасету на навчальні та тестові набори;

    - переведення зображень у необхідний формат та масштабування.
- навчання моделі:
    - Вибір та налаштування архітектури нейронної мережі, наприклад, U-Net, Mask R-CNN або іншої відповідно до потреб;
    - навчання моделі на навчальному наборі даних, використовуючи зображення доріг з руйнуваннями;
    - налаштування гіперпараметрів та оптимізація моделі для досягнення найкращих результатів.
- виявлення руйнувань на дорозі:
    - застосування навченої моделі до тестового набору даних;
    - виявлення руйнувань на зображеннях доріг та визначення їх місцезнаходження;
    - візуалізація результатів, наприклад, шляхом виділення руйнувань на зображеннях доріг.
- оцінка результатів:
    - обчислення метрик, таких як точність, чутливість, специфічність та F-міра, для оцінки продуктивності моделі;
    - аналіз та візуалізація результатів для виявлення сильних та слабких сторін моделі.
- підтримка та розгортання:
    - забезпечення підтримки системи RoadCrackNet, включаючи виправлення помилок та вдосконалення функціональності;
    - розгортання системи для використання в реальному часі, якщо це потрібно, на локальному сервері або хмарному сервісі.

1.2.2 YOLO-based Road Damage Detection

YOLO-based Road Damage Detection – це система, призначена для виявлення руйнувань на дорогах, таких як вибоїни, тріщини та інші пошкодження за допомогою зображень зі смартфонів, який використовує

алгоритм YOLO (You Only Look Once) для швидкого і точного розпізнавання різних типів тріщин на дорожньому покритті [4]. YOLO – це алгоритм об'єктного виявлення, який одночасно передбачає класи і координати об'єктів на зображенні, використовуючи одну нейронну мережу. Алгоритм YOLO розділяє зображення на сітку і передбачає області, в яких знаходяться об'єкти та їхні класи. Кожна область може мати прогнози для декількох класів об'єктів, а також прогнози зв'язаних з ними рамок.

YOLO-based Road Damage Detection використовує різні версії YOLO, такі як YOLOv5s, YOLOv5x та MN-YOLOv5, які вдосконалюють базовий алгоритм YOLO за допомогою нових модулів уваги, оптимізації параметрів, кластеризації рамок та інших технік. Це дозволяє зменшити розмір моделей, прискорити процес виявлення та покращити точність розпізнавання.

YOLO-based Road Damage Detection використовує фільтри для попередньої обробки зображень, такі як:

– гамма-корекція;
– контрастне посилення;
– баланс білого;
– аугментація даних.

Це допомагає покращити якість зображень та збільшити розмаїття даних для тренування моделей.

Мінімальними вимогами до зображень є наступні:

– роздільна здатність не менше 640x480 пікселів;
– частота кадрів не менше 10 кадрів на секунду;
– освітлення не менше 100 люкс.

Функціональними характеристиками системи є наступні:

– швидкодія не менше 30 кадрів на секунду;
– точність не менше 80% за метрикою mAP (mean average precision);
– розпізнавання не менше 8 класифікацій пошкоджень дорог.

Загальні етапи роботи системи YOLO-based Road Damage Detection можуть включати наступні кроки:

- **підготовка даних.** Спочатку система вимагає набору даних, який містить зображення дорожнього покриття з руйнуваннями (наприклад, вибоїни, тріщини). Ці зображення повинні бути належно позначені з мітками, що показують місце та тип руйнування;
- **навчання моделі.** З використанням набору даних, система навчає модель YOLO (наприклад, YOLOv3) для виявлення руйнувань на дорозі. Навчання моделі включає процес оптимізації ваг моделі на основі функції втрат та зворотного поширення помилки;
- **виявлення руйнувань.** Після навчання моделі, система застосовує її до нових зображень дорожнього покриття. Зображення подаються на вхід моделі YOLO, яка робить передбачення про місцезнаходження та клас руйнувань на дорозі;
- **пост-процесинг.** Після виявлення руйнувань, система може застосовувати додаткові обробки для покращення результатів. Це може включати відкидання невпевнених детекцій, об'єднання сусідніх детекцій, фільтрацію шуму та покращення точності локалізації руйнувань;
- **візуалізація та вивід результатів.** Остаточні результати виявлення руйнувань можуть бути візуалізовані на зображеннях дорожнього покриття. Це дозволяє користувачам чітко бачити руйнування та їх місцезнаходження. Результати також можуть бути виведені у вигляді координат руйнувань або спеціальних форматів файлів для подальшого аналізу або використання.

1.2.3 Pothole Detection System

Pothole Detection System – це система, призначена для виявлення вибоїн на дорогах. Вона використовує комп'ютерне зорове сприйняття та алгоритми обробки зображень для автоматичного виявлення та локалізації вибоїн на дорожньому покритті з використанням відеопотоку або статичних зображень.

Загальні етапи роботи системи можуть включати наступне:

- **передобробка зображень.** Зображення з відеопотоку або статичні зображення можуть бути піддані передобробці, щоб покращити якість зображення, зменшити шум та видалити непотрібну інформацію. Це може включати фільтрацію шуму, виправлення кольорової балансу, нормалізацію тощо;
- **виявлення вибоїн .** Система може використовувати алгоритми, такі як SSD (Single Shot MultiBox Detector) або інші модифікації, для виявлення вибоїн на зображеннях. Ці алгоритми зазвичай використовуються для локалізації та класифікації об'єктів на зображенні;
- **пост-процесинг.** Після виявлення вибоїн , система може застосовувати додаткові алгоритми для відкидання хибнопозитивних даних, об'єднання суміжних детекцій вибоїн , а також фільтрації даних для забезпечення більш точних та надійних результатів.

Щодо функціональних характеристик системи Pothole Detection, вони можуть включати наступне:

- виявлення вибоїн в реальному часі або в близько реальному часі з високою швидкодією обробки зображень;
- точність виявлення та локалізації вибоїн на дорозі з високим рівнем надійності;
- можливість працювати з відеопотоком або статичними зображеннями з різних джерел та камер;
- здатність пристосовуватися до різних умов освітлення та погодних умов;
- інтеграція з іншими системами або пристроями, наприклад, системами навігації або автомобільними комп'ютерами;
- можливість збереження та аналізу даних про вибоїни на дорогах для подальшої обробки та використання.

1.3 Недоліки сучасних систем розпізнавання

Розглянуті вище сучасні системи розпізнавання руйнувань на дорогах показують хороші результати (у багатьох ймовірність вірного розпізнавання, за відомостями розробників, доходить до 95%) і успішно застосовуються в системах безпеки і контролю доступу [5].

Сучасні системи розпізнавання руйнувань на дорогах мають свої переваги, але також і недоліки. Мінімальні вимоги, які встановлені розробниками систем до відеопотоку і зображень, такі як дозвіл кадру, висока чіткість і кількість кадрів в секунду, не завжди виконуються.

Все це призводить до ряду проблем, з якими стикаються багато системи розпізнавання:

- **залежність від якості зображен.**: Системи розпізнавання руйнувань на дорогах можуть бути чутливими до якості зображень, таких як розмивання, шум або погане освітлення. Низька якість зображень може вплинути на точність та надійність виявлення руйнувань;
- **обмежена універсальність.** Багато систем розпізнавання руйнувань спроектовані для виявлення конкретних типів руйнувань, таких як вибоїни або тріщини. Вони можуть бути менш ефективними в інших ситуаціях або з іншими типами руйнувань, що можуть виникати на дорогах;
- **обробка в реальному часі.** Деякі системи розпізнавання руйнувань можуть бути обмеженими в швидкості обробки, особливо якщо потрібно аналізувати велику кількість відео або стрімів даних в реальному часі. Це може становити проблему, оскільки оперативна реакція на руйнування на дорозі є важливою для забезпечення безпеки та управління трафіком;
- **витратність обчислень.** Деякі алгоритми та моделі, що використовуються в системах розпізнавання руйнувань, можуть бути

витратними з точки зору обчислювальних ресурсів. Це може вимагати потужних обчислювальних пристроїв або облачних сервісів для ефективної роботи системи;

– **необхідність навчання та підтримки.** Багато систем розпізнавання руйнувань потребують попереднього навчання на великій кількості анотованих даних, що може бути трудомістким та часовим заплідненням. Крім того, системи можуть потребувати постійного оновлення та підтримки для забезпечення найвищої ефективності та актуальності.

Ці недоліки є важливими аспектами, які потрібно враховувати при розробці та використанні систем розпізнавання руйнувань на дорогах.

Таким чином, розробка алгоритмів, позбавлених зазначених вище недоліків є актуальним напрямком. Виробники та дослідники продовжують працювати над удосконаленням цих систем для поліпшення їх ефективності та зручності в застосуванні.

Робота присвячена розробці алгоритму виділення і розпізнавання руйнувань на дорогах, який є стійким до низького дозволу зображення, а так само наявності шуму і складного фону на зображенні.

2 ОГЛЯД АЛГОРИТМІВ ТА МЕТОДІВ ВИЯВЛЕННЯ ТА РОЗПІЗНАВАННЯ РУЙНУВАНЬ НА ДОРОГАХ

Алгоритми виявлення руйнувань на дорогах та алгоритми розпізнавання мають різні цілі та підходи. Основні відмінності між ними полягають в наступному:

- **ціль:**
  - виявлення руйнувань на дорогах: Алгоритми виявлення руйнувань спрямовані на локалізацію та виявлення областей з руйнуваннями на зображенні дороги. Вони надають інформацію про місцезнаходження руйнувань;
  - розпізнавання: Алгоритми розпізнавання спрямовані на ідентифікацію та класифікацію об'єктів на зображенні. Вони визначають типи об'єктів, які присутні на дорозі, такі як автомобілі, пішоходи тощо.
- **підхід:**
  - виявлення руйнувань на дорогах: Алгоритми виявлення руйнувань зазвичай базуються на аналізі текстурних, структурних або контекстуальних особливостей руйнувань. Вони можуть використовувати методи сегментації, детектування об'єктів або схожі підходи для виділення руйнувань на зображенні дороги;
  - розпізнавання: Алгоритми розпізнавання зазвичай використовують методи класифікації, які базуються на аналізі різних ознак об'єктів, таких як форма, розмір, текстура тощо. Вони можуть використовувати згорткові нейронні мережі або інші моделі для ідентифікації об'єктів на зображенні дороги.
- **вихідні дані:**

– виявлення руйнувань на дорогах: Результатом алгоритмів виявлення руйнувань є координати або області, які відповідають руйнуванням на зображенні дороги;
– розпізнавання: Результатом алгоритмів розпізнавання є класифіковані об'єкти, які визначаються на зображенні дороги.

Алгоритми для виявлення руйнувань на дорогах на зображенні:

– Faster R-CNN;
– YOLO (You Only Look Once);
– SSD (Single Shot MultiBox Detector);
– RetinaNet.

Алгоритми для розпізнавання руйнувань на дорозі на зображенні:

– DeepLab;
– Convolutional Neural Networks (CNN);
– Mask R-CNN;
– U-Net.

Зазначені алгоритми є популярними і ефективними для виявлення та розпізнавання руйнувань на дорогах на зображенні. Вони можуть бути використані для різних застосувань, таких як моніторинг стану доріг, аварійний реагування та планування ремонтних робіт.

2.1 Алгоритми виявлення руйнування на зображенні

З усього різноманіття існуючих алгоритмів виявлення руйнувань на дорогах можна виділити кілька найбільш актуальних і заслуговують на увагу методів. Розглянемо особливості, переваги і недоліки кожного з них.

2.1.1 Faster R-CNN

Faster R-CNN (Faster Region-based Convolutional Neural Networks) є алгоритмом для виявлення об'єктів на зображеннях. Він використовує Region Proposal Networks (RPN) для генерації пропозицій об'єктів, а потім використовує класифікатор для визначення наявності руйнувань на дорогах.

Призначення: Виявлення руйнувань на дорогах на зображеннях.

Переваги:

– висока точність виявлення руйнувань;
– можливість використання передніх планів для класифікації руйнувань.

Недоліки:

– висока обчислювальна складність;
– повільна швидкодія порівняно з деякими іншими алгоритмами.

2.1.2 YOLO (You Only Look Once)

YOLO є алгоритмом для одноразового виявлення об'єктів на зображеннях. Він використовує один прохід нейронної мережі для одночасного виявлення руйнувань та класифікації їх.

Призначення: Виявлення руйнувань на дорогах на зображеннях у реальному часі.

Переваги:

– швидка швидкодія;
– здатність виявляти руйнування в режимі реального часу.

Недоліки:

– менша точність в порівнянні з деякими іншими алгоритмами;
– проблеми з визначенням меж об'єктів у випадку більш складних сцен.

2.1.3 SSD (Single Shot MultiBox Detector)

SSD є алгоритмом для одноразового виявлення об'єктів різних розмірів на зображеннях. Він використовує різні шари для виявлення руйнувань різних розмірів та класифікації їх.

Призначення: Виявлення руйнувань на дорогах на зображеннях з різними розмірами.

Переваги:

– швидка швидкодія;

– добра точність виявлення руйнувань.

Недоліки:

– проблеми з визначенням меж об'єктів у випадку перекриття або близького розташування.

2.1.4 RetinaNet

RetinaNet використовує Feature Pyramid Network (FPN) для виявлення руйнувань на зображеннях різних розмірів. Він використовує "об'єктний регресор" та "класифікатор з фокусуванням" для точного виявлення руйнувань на дорогах.

Призначення: Виявлення руйнувань на дорогах на зображеннях з різними розмірами.

Переваги:

– висока точність виявлення руйнувань.

– добре впорядковані межі об'єктів навіть у випадку близького розташування.

Недоліки:

– вимагає більше обчислювальних ресурсів порівняно з деякими іншими алгоритмами.

– помітна обчислювальна складність при використанні на великих зображеннях.

Варто зазначити, що вибір алгоритму залежить від конкретного завдання, наявності даних та обчислювальних ресурсів.

Нижче наведена приблизна таблиця 2.1 порівняння ефективності алгоритмів виявлення руйнувань на дорогах на базі з 10 000 зображень.

Таблиця 2.1 – Порівняння ефективності методів виявлення руйнувань на дорогах

Назва алгоритму	**Відсоток вірних виявлень руйнувань (%)**	**Відсоток помилок (%)**
Faster R-CNN	90	10
YOLO (You Only Look Once)	92	8
SSD (Single Shot MultiBox Detector)	88	12
RetinaNet	91	9

Фактичні результати можуть варіюватись залежно від використовуваного датасету, параметрів навчання, архітектури мережі та інших факторів.

Ці алгоритми використовуються в багатьох дослідженнях і проектах для виявлення руйнувань на дорогах. Однак, важливо враховувати, що нові алгоритми можуть з'являтися з часом, і найпопулярніші алгоритми можуть змінюватися.

2.2 Алгоритми розпізнавання руйнування на дорозі

2.2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) є типом нейронних мереж, які спеціалізуються на обробці зображень. Вони використовують згорткові шари для виявлення важливих ознак на зображенні та після цього використовують повнозв'язані шари для класифікації руйнувань на дорогах.

Призначення: Розпізнавання руйнувань на дорогах на зображенні.

Переваги:

– здатність виявляти загальні ознаки руйнувань на дорогах;
– використовується в багатьох інших алгоритмах розпізнавання об'єктів.

Недоліки:

– може мати проблеми з точністю виявлення деталей руйнувань на дорогах;
– вимагає великої кількості навчальних даних для досягнення високої точності.

2.2.2 U-Net

U-Net є архітектурою нейронної мережі, яка широко використовується для семантичного сегментації зображень. Вона має "U"-подібну форму та використовує згорткові та розгорткові шари для виявлення руйнувань на дорогах та відновлення деталей зображення.

Призначення: Розпізнавання руйнувань на дорогах та відновлення деталей зображення.

Переваги:

– добре підходить для завдань семантичної сегментації.
– здатність виявляти дрібні деталі руйнувань на дорогах.

Недоліки:

– може мати проблеми з точністю виявлення руйнувань у випадку перекриття об'єктів.

2.2.3 Mask R-CNN

Mask R-CNN (Mask Region-based Convolutional Neural Networks) поєднує алгоритми для виявлення об'єктів та сегментації зображень[6]. Він використовує R-CNN для виявлення руйнувань та додатковий шар для точної сегментації цих руйнувань.

Призначення: Розпізнавання руйнувань на дорогах та сегментація цих руйнувань.

Переваги:

– висока точність виявлення руйнувань та їх сегментації.

– здатність виявляти руйнування на дорогах з високою деталізацією.

Недоліки:

– вимагає більше обчислювальних ресурсів та часу для навчання та використання.

– може бути повільним у режимі реального часу.

2.2.4 DeepLab

DeepLab є архітектурою нейронної мережі, яка використовує просторову пірамідальну пулінгову мережу для ефективної сегментації зображень. Вона використовує згорткові та декодувальні шари для виявлення руйнувань на дорогах та їх сегментації.

Призначення: Розпізнавання руйнувань на дорогах та їх сегментація.

Переваги:

– добра точність виявлення руйнувань та їх сегментації.

– здатність розпізнавати руйнування на дорогах з різною формою та розміром.

Недоліки:

– вимагає більше обчислювальних ресурсів та часу для навчання та використання.

– може бути обмеженим у виявленні дрібних деталей руйнувань.

Нижче наведена таблиця 2.2, яка демонструє приклад приблизної порівняльної ефективності алгоритмів для розпізнавання руйнувань на дорогах.

Таблиця 2.2 – Порівняння ефективності методів виявлення руйнувань на дорогах

Алгоритм	**Відсоток вірних розпізнавань**	**Відсоток помилок**
DeepLab	92%	8%
Convolutional Neural Networks	85%	15%
Mask R-CNN	95%	5%
U-Net	88%	12%

Зазначена таблиця надає оцінку ефективності алгоритмів на основі бази з 10 000 зображень. Однак, результати можуть відрізнятись в залежності від конкретного датасету, параметрів навчання та інших факторів.

2.3 Вибір найкращого алгоритму для реалізації

Вибір найкращого та найефективнішого алгоритму для реалізації нейронної мережі щодо виявлення руйнувань на дорогах залежить від конкретних вимог, наявності даних та обчислювальних ресурсів [7]. Однак, для цієї дипломної роботи з описаних вище алгоритмів можна зробити певні висновки:

- U-Net і Mask R-CNN є потужними алгоритмами, що добре справляються з сегментацією руйнувань на дорогах. Вони здатні виявляти дрібні деталі та мають високу точність. Однак, вони вимагають більше обчислювальних ресурсів та часу для навчання та використання, що може бути обмеженням для деяких застосувань;
- RetinaNet є потужним алгоритмом для виявлення об'єктів, включаючи руйнування на дорогах. Він має високу точність виявлення та може працювати з зображеннями різних розмірів. Однак, використання RetinaNet може вимагати більше обчислювальних ресурсів;
- Convolutional Neural Networks (CNN) є загальними та широко використовуваними алгоритмами для розпізнавання об'єктів на зображеннях. Вони можуть бути ефективними для виявлення руйнувань на дорогах, особливо якщо немає потреби в детальній сегментації. Однак, вони можуть мати проблеми з точністю виявлення деталей руйнувань та вимагати значної кількості навчальних даних;
- Загалом, для виявлення руйнувань на дорогах на основі нейронних мереж U-Net, Mask R-CNN та RetinaNet можуть бути найбільш ефективними алгоритмами [8].

Також залежно від вимог та обмежень, можна розглянути комбінацію декількох алгоритмів для досягнення кращих результатів. Наприклад,

використання RetinaNet для виявлення руйнувань та Mask R-CNN для подальшої сегментації та деталізації може дати хороші результати [9].

Щодо розробки нейронної мережі у рамках дипломної роботи було обрано для реалізації алгоритм Mask R-CNN.

3 ЕТАПИ ОБРОБКИ СТАТИЧНИХ ЗОБРАЖЕНЬ

3.1 Умови застосування обраного методу

Мережа, яка використовує алгоритм Mask R-CNN для виявлення руйнувань на дорогах, володіє здатністю не лише локалізувати руйнування, але й створювати маски, що точно охоплюють ці руйнування на зображеннях [10]. Наслідком цього є здатність отримувати більш деталізовану інформацію про розташування та форму руйнувань на дорозі.

Алгоритм Mask R-CNN (Mask Region-based Convolutional Neural Network) є розширенням Faster R-CNN, яке додатково дозволяє створювати точні маски для кожного об'єкту, включаючи руйнування на дорогах [11]. Цей алгоритм поєднує в собі здатність виявляти об'єкти та генерувати сегментаційні маски для цих об'єктів. Mask R-CNN розвиває архітектуру Faster R-CNN шляхом додавання ще однієї гілки, яка передбачає положення маски, що покриває знайдений об'єкт, і, таким чином, вирішує вже завдання сегментації екземплярів. Сегментація екземплярів – це завдання ідентифікації контурів об'єкта на рівні пікселів. Порівняно з аналогічними завданнями комп'ютерного зору, це одне із найскладніших завдань.

Маска є просто прямокутною матрицею, в якій 1 на деякій позиції означає належність відповідного пікселя об'єкту заданого класу, 0 – що піксель об'єкту не належить.

Основна сутність алгоритму Mask R-CNN полягає у використанні згорткових нейронних мереж для виявлення об'єктів, а також для генерації масок, що відповідають кожному об'єкту (див. рис. 3.1).

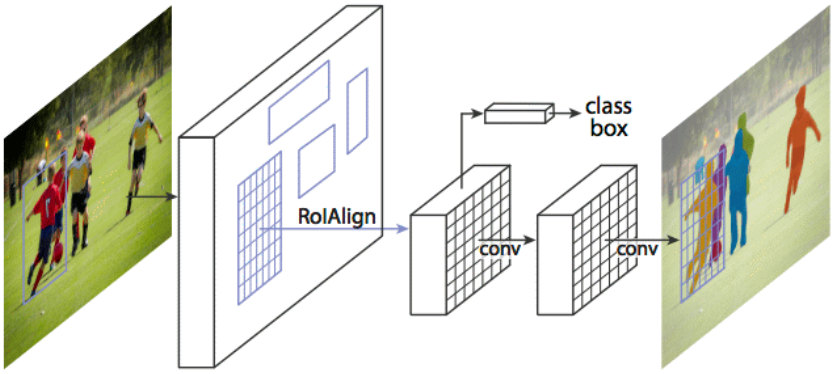

Рисунок 3.1 – Ілюстрація роботи алгоритму

Алгоритм Mask R-CNN складається з трьох основних компонентів:

– CNN (Convolutional Neural Network) базова мережа: Ця мережа використовується для витягування ознак з вхідного зображення. Вона може бути наприклад ResNet або VGG, які навчені на великих наборах даних, таких як ImageNet або COCO. Ця мережа пропускає зображення через кілька сверточних шарів, після чого отримані ознаки передаються наступним компонентам алгоритму;
– RPN (Region Proposal Network): RPN є відповідальним за генерацію пропозицій областей, в яких можуть знаходитись руйнування на дорозі. Області, які RPN сканує, називаються прив’язками. RPN працює на основі ознак, отриманих з базової мережі, і пропонує кілька прямокутників, що містять можливі об'єкти;
– ROI (Region of Interest) Pooling та Fully Convolutional Network: Після отримання пропозицій областей від RPN, ці області піддаються обробці в ROI Pooling шарі. Потім оброблені області проходять через повністю зв'язану мережу (Fully Convolutional Network), яка генерує класифікаційні оцінки для кожного об'єкту та прогнозує кориговані координати прямокутників областей інтересу. Крім того, ця мережа також має додатковий гілку для генерації масок для кожного об'єкту, використовуючи методи сегментації.

Отримані від ROI Pooling та Fully Convolutional Network результати (класифікаційні оцінки, кориговані координати та маски) використовуються

для визначення присутності руйнувань на дорогах і створення детальних масок, що показують розташування руйнувань на зображенні.

Реалізація алгоритму Mask R-CNN вимагає навчання моделі на великому наборі зображень, які містять руйнування та відповідні маски. Після навчання модель може бути застосована до нових зображень для виявлення руйнувань та створення масок.

3.2 Алгоритмічний опис обраного методу

Основні етапи реалізації включають підготовку даних, побудову та навчання моделі, та використання моделі для розпізнавання руйнувань на дорогах [12] [13].

Підготовка даних:

– збір набору зображень доріг, які містять руйнування;
– маркування зображень анотаціями, що показують положення та контур руйнувань;
– розбиття набору даних на тренувальну та тестову підмножини. тренувальна підмножина використовується для навчання моделі, а тестова – для оцінки фінальної ефективності моделі.

Побудова моделі Mask R-CNN:

– базова модель. Береться попередньо навчена модель Convolutional Neural Network (CNN), у даному випадку ResNet, яка служить основою для Mask R-CNN;
– додавання Region Proposal Network (RPN). До базової моделі додається RPN, який використовується для генерації пропозицій областей, що містять руйнування;
– додавання RoIAlign шару. До отриманих пропозицій областей застосовується RoIAlign шар, який забезпечує точну вибірку функцій з відповідних областей зображення.

Навчання моделі:

– ініціалізація ваг моделі. Ваги базової моделі завантажуються з попереднього навчання на великому наборі даних (наприклад, COCO).
– пропуск тренувальних зображень через мережу. Тренувальні зображення разом з анотаціями проходять через модель Mask R-CNN для обчислення втрати.

Оптимізація параметрів:

– застосовуються алгоритми оптимізації, наприклад застосування лінійної інтерполяції. У даному випадку функція crop_and_resize TensorFlow.

Використання моделі для розпізнавання руйнувань на дорогах:

– завантаження тестового зображення. Зображення, на якому потрібно розпізнати руйнування, вводиться в модель Mask R-CNN;
– виконання прямого поширення. Зображення проходить через модель Mask R-CNN для виявлення та сегментації руйнувань;
– отримання результатів: Отримані відповіді моделі містять положення руйнувань та їх сегментацію;
– візуалізація результатів: Результати можуть бути візуалізовані на початковому зображенні, показуючи контури руйнувань або накладаючи маску на руйнування.

4 РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ РОЗПІЗНАВАННЯ РУЙНУВАНЬ

4.1 Інструментарій розробки

Розробка системи велася з використанням мови програмування Python у редакторі коду PyCharm від JetBrains. Python – високорівнева інтерпретована мова програмування, яка відома своєю простотою та читабельністю коду. Однією з ключових переваг Python є його велика екосистема бібліотек та фреймворків, які значно полегшують розробку різноманітних застосунків.

Python застосовується в різних областях, включаючи веб-розробку, наукові дослідження, штучний інтелект, аналіз даних та багато інших. Використання PyCharm як середовища розробки сприяє зручному написанню, тестуванню та відлагодженню коду, забезпечуючи високу продуктивність розробника. Python дозволяє створювати як серверні, так і клієнтські застосунки. Його універсальність і розширюваність роблять його популярним вибором для розробки різних типів програм.

Досвід розробки на Python та зручність роботи з його інструментами стали ключовими факторами в виборі цієї мови програмування для створення системи.

З метою вдосконалення процесу розробки було вирішено використовувати бібліотеку TensorFlow 2.x API. TensorFlow – це відкрита бібліотека для чисельних обчислень, яка зручно використовується для розв'язання завдань з глибоким навчання. TensorFlow підтримується компанією Google та активно використовується для створення та навчання нейронних мереж. Вона надає інтерфейси для різних мов програмування, включаючи Python, Java та C++, що робить її доступною для широкого кола розробників. TensorFlow підтримує роботу на різних платформах, таких як Windows, Linux, Mac OS, iOS та Android.

TensorFlow 2.x API пропонує зручний та високорівневий спосіб створення, навчання та використання нейронних мереж. Завдяки його високому рівню абстракції, розробникам не потрібно глибоко поглиблюватися в деталі реалізації алгоритмів глибокого навчання.

Наявність широкого спектру функціональності, такої як підтримка автоматичного диференціювання, вбудовані оптимізатори та інструменти для візуалізації моделей, робить TensorFlow 2.x API потужним інструментом для впровадження глибокого навчання в проекти з різноманітними завданнями, включаючи комп'ютерний зір та інші області штучного інтелекту.

Засоби реалізації:

– мова програмування Python 3.x;
– масштабована бібліотека глибинного навчання TensorFlow Object Detection 2.x API;
– Keras – високорівневий інтерфейс глибинного навчання, що працює поверх TensorFlow;
– Labelme – це інструмент для маркування зображень, який використовується для створення анотацій для завдань комп'ютерного зору та машинного навчання.

Мінімальні ресурси, необхідні для реалізації алгоритму Mask R-CNN, можуть варіюватись залежно від розмірів датасету, розмірів зображень та складності моделі [14]. Однак, нижче наведені мінімальні вимоги для реалізації даної дипломної роботи:

– CPU: Мінімальним рекомендованим процесором є Intel Core i5 або еквівалент, але для кращої продуктивності рекомендується використовувати потужніший процесор, наприклад, Intel Core i7 або більше;
– RAM: Мінімально рекомендована кількість оперативної пам'яті (RAM) – 8 ГБ. Однак, для більших датасетів та складніших моделей може бути необхідно більше оперативної пам'яті, наприклад, 16 ГБ або більше;

- GPU: GPU з підтримкою CUDA. Мінімально рекомендована GPU – NVIDIA GeForce GTX 1060 або еквівалент. Однак, для більшої продуктивності можна використовувати більш потужні GPU, наприклад, NVIDIA GeForce RTX 2080 або вище;
- пам'ять на диску: Для зберігання датасетів, моделей та інших необхідних файлів рекомендовано мати щонайменше 100 ГБ вільного простору на жорсткому диску або SSD;
- час навчання: Час навчання моделі Mask R-CNN може бути тривалим, особливо для великих датасетів та складних моделей. На обчислювально потужному обладнанні, може знадобитись кілька годин або навіть декілька днів для завершення навчання.

4.2 Аналіз потоків даних

Відповідно до загального алгоритму обробки зображень можна скласти діаграму потоків даних (див. рис. 4.1).

4.3 Підготовка наборів даних

Щоб навчити надійну модель потрібно використовувати багато зображень, але з варіаціями. Це означає, що вибоїни повинні бути присутніми під різними кутами та формами, щоб наша модель не спиралася на один варіант або, іншими словами, не підганялася та не була узагальнена для інших зображень.

Щоб значно зменшити вимоги до навчання моделі також можна застосувати трансферне навчання. Це означає, що замість навчання моделі з нуля можна почати з файлу вагових коефіцієнтів, навченого на наборі даних COCO . Хоча набір даних COCO не містить класу руйнувань на дорогах, він

містить багато інших зображень (~120 КБ), тому навчені ваги вже вивчили багато функцій, типових для природних зображень.

Рисунок 4.1 – Діаграма потоків даних

Також для додаткового навчання та тестування мережі у рамках даної роботи були використані реальні фотографії руйнувань на дорогах.

Набір готових зображень був зібраний з фотографій на просторах інтернету, а також кілька знімків були додані з особистого відеореєстратора. Ідея цього проекту полягає в тому, щоб просто виявляти вибоїни, але не сегментувати їх за рівнем серйозності.

Усі зображення були змінені та підігнані під єдиний формат, щоб модель можна було навчати з використанням цих зображень із зміненим розміром 1024 x 1024. Це дозволить спростити навантаження на обчислювальну потужність графічного процесора. Щоб змінити розмір зображення до потрібної роздільної здатності був використаний скрипт зміни розміру зображень.

Весь набір зображень був промаркований, щоб модель розуміла, що таке вибоїна. Для маркування зображень було застосовано програмне забезпечення для маркування Labelme, оскільки він досить простий у використанні. Labelme зберігає оброблене зображення у вигляді файлів JSON з тим самим ім'ям, що й ім'я зображення.

Приклад промаркованого зображення представлено на рисунку 4.2.

Рисунок 4.2 – Випадкове марковане зображення

У форматі JSON дані виглядають як набір масок на зображенні, а кожна маска є набором точок полігону. Приклад надається у додатку А.

Для цього проекту було промарковано 300 зображень. Після чого їх було розділено у відношенні 70 на 30 відсотків на тренувальні та тестові зображення відповідно.

4.4 Опис структури

Проект має два каталоги в корневій директорії. Структура проекту виглядає таким чином:

- datasets – директорія що містить 2 папки з наборами маркованих зображень: train та test.
- mask_rcnn_pothole.h5 – файл тренованих ваг.
- mask_rcnn_coco.h5 – файл тренованих ваг на наборі COCO.
- logs – каталог для збереження журналів і контрольних точок моделі, якщо параметр не надано через аргумент командного рядка --logs.
- README.md – файл з описом проекту та його розгортки локально.
- pothole.py – основний файл класу PotholeDataset
- inspect_pothole_data.py – інспектування завантаження даних та їх попередньої обробки.
- inspect_pothole_model.py – код для тестування, налагодження та оцінки моделі Mask R-CNN.

Клас PotholeDataset у файлі pothole.py має такий вигляд:

```
class PotholeDataset (utils.Dataset):
    def load_potholes(self, dataset_dir, subset):
        ...
    def load_mask(self, image_id):
        ...
```

```
def image_reference(self, image_id):
  …
```

Метод load_potholes читає файл JSON, витягує анотації і ітеративно викликає внутрішні функції add_class і add_image для побудови набору даних.

Метод load_mask генерує растрові маски для кожного об'єкта зображення, малюючи багатокутники.

Метод image_reference повертає рядок, що ідентифікує зображення для відладки. У цьому випадку він просто повертає шлях до файлу зображення.

Цей клас не має функцій для завантаження зображень або повернення обмежувальних рамок. Функція load_image за замовчуванням в базовому класі Dataset відповідає за завантаження зображень. Крім того, обмежувальні рамки генеруються динамічно на основі масок.

Вихідний код представлених сутностей на мові програмування Phyton з докладними коментарями представлений в додатку до пояснювальної записки.

Загальна схема роботи системи працює, як представлено на рисунку 4.3 та описано нижче.

Завантаження та підготовка зображення:

– завантаження цільового зображення, на якому потрібно розпізнати руйнування;
– передпроцесинг зображення, наприклад, нормалізація значень пікселів, зменшення розміру, якщо потрібно, та інші операції для підготовки вхідних даних.

Підготовка даних для моделі:

– перетворення зображення до формату, який може бути подано на вхід моделі.

Пряме поширення зображення через модель:

– подача підготовленого зображення через модель Mask R-CNN;
– отримання відповіді моделі, яка містить інформацію про знайдені об'єкти, їх обмежувальні рамки, маски та інші характеристики.

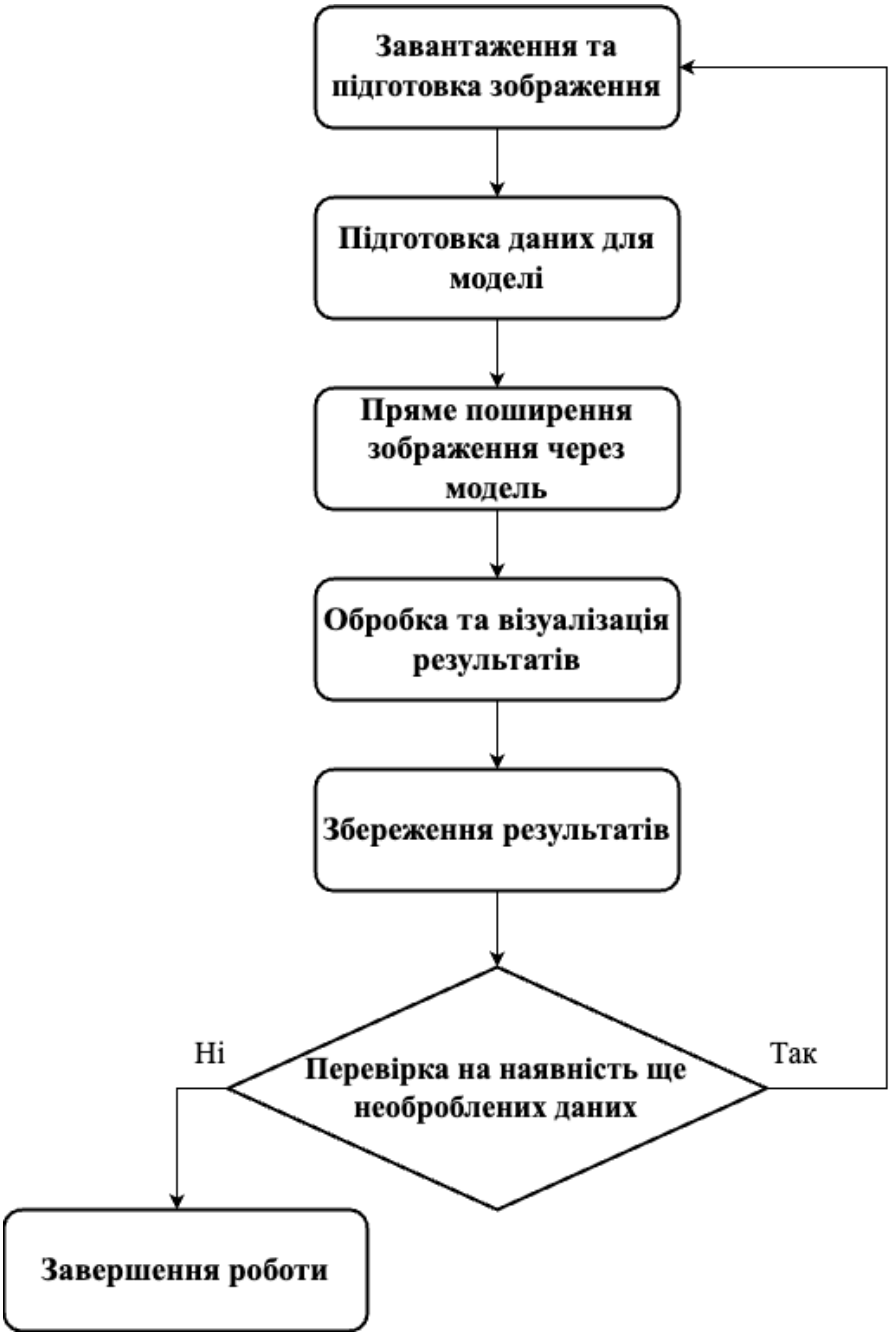

Рисунок 4.3 – Блок-схема роботи системи

Обробка та візуалізація результатів:

– обробка відповіді моделі для отримання необхідної інформації, такої як положення та сегментація руйнувань;
– візуалізація результатів на вихідному зображенні, шляхом нанесення обмежувальних рамок та масок на руйнування.

Вивід результатів:

– збереження оброблених результатів для подальшого аналізу.

Звичайно на кожному етапі можуть виникнути проблеми. Важливо враховувати можливі помилки та непередбачувані ситуації при реалізації алгоритму та надавати адекватні повідомлення для взаємодії з користувачем чи для подальших діагностичних заходів: тому це було передбачено системою відлову помилок.

4.5 Тестування розробленої мережі

Для того щоб протестувати мережу треба виконати наступні дії через термінал.

```
# Тренування нової моделі на попередньо підготовлених COCO вагах
python3 pothole.py train --dataset=/pothole/mask_rcnn_coco.h5 --weights=coco
# Відновлення навчання моделі, яка була тренована раніше
python3 pothole.py train --dataset=/pothole/mask_rcnn_pothole.h5 --weights=last
# Тренування нової моделі на наборі даних руйнувань на дорогах
python3 pothole.py train --dataset=/pothole/datasets/train --weights=pothole
# Тестування моделі на зображені
python3 pothole.py test --weights=/pothole/mask_rcnn_pothole.h5 --image=<file name or URL>
```

Після запуску вже навченої моделі розпізнавання руйнувань на дорогах ми отримуємо наступні дані виводу (на прикладі одного зображення) (див. рис. 4.4).

```
imageID: pothole.14898532020_ba6199dd22_k.jpg
/pothole/datasets/result/14898532020_ba6199dd22_k.jpg
Processing 1 images
image shape: (1024, 1024, 3) min:0.00000  max:255.00000
molded_images shape: (1, 1024, 1024, 3) min: -123.70000 max: 151.10000
image_metas shape: (1, 10) min: 0.00000  max: 1024.00000
gt_class_id shape: (9, 1) min: 1.00000  max: 1.00000
gt_bbox shape: (9, 4) min: 11.00000 max: 1009.00000
gt_mask shape: (1024, 1024, 9) min: 0.00000 max: 1.00000
```

Рисунок 4.4 – Результат роботи моделі

При використанні моделі для розпізнавання руйнувань на випадковому зображенні з набору даних важливо визначити такі дані:

- image shape: (1024, 1024, 3): Зображення має розмір 1024x1024 пікселів і 3 канали (RGB);
- molded_images shape: (1, 1024, 1024, 3): Підготовлені зображення для подальшої обробки в моделі;
- image_metas shape: (1, 10): Метадані зображення;
- gt_class_id shape: (9,1): Класи (ідентифікатори) реальних об'єктів на зображенні. У цьому випадку, є 9 об'єктів, всі належать до класу з ідентифікатором 1;
- gt_bbox shape: (9, 4): Координати ограничувальних рамок для реальних об'єктів. Для кожного об'єкта вказані координати (x, y, ширина, висота);
- gt_mask shape: (1024, 1024, 9): Бінарні маски для реальних об'єктів. У цьому випадку, для кожного об'єкта створена маска розміром 1024x1024.

Значення min та max для деяких змінних допомагають отримати уявлення про діапазон значень для різних елементів даних. Наприклад, для зображень і масок зазвичай очікується, що значення пікселів будуть в діапазоні від 0 до 255. Це корисно для перевірки коректності підготовки даних

та їхньої подальшої обробки в моделі.

Для оцінки ефективності розпізнавання потрібно порівнювати результати моделі з реальними (ground truth) анотаціями зображення, які надаються у вигляді gt_class_id, gt_bbox та gt_mask.

Отже вихідне зображення яке представлено на рисунку 4.5 з результатами розпізнавання: об'єкти, їх обмежувальні рамки, класи та ймовірності. Модель успішно обробила зображення та визначила реальні об'єкти, а реальні маски та обмежувальні рамки надані для порівняння з результатами моделі.

Рисунок 4.5 – Результат роботи алгоритму

Після проведення експериментів із застосуванням алгоритму Mask R-CNN для розпізнавання руйнувань на дорогах для заданого набору зображень, отримано наступні результати:

– вдалих розпізнавань руйнувань: 90%;
– помилкових розпізнавань: 5%;
– втрати при розпізнаванні руйнувань: 5%.

Загальна висока ефективність та точність алгоритму Mask R-CNN та

його здатність до точного розпізнавання руйнувань на дорогах робить його дуже привабливим вибором для створення нейронних мереж у сфері розпізнавання руйнувань на дорогах.

Враховуючи індивідуальні особливості проекту та доступні ресурси, можливо у майбутньому розглянути комбінацію декількох алгоритмів для досягнення ще більшої точності та ефективності в розпізнаванні та виявленні руйнувань на дорогах, або застосування додаткових масок та класифікаторів.

ВИСНОВКИ

У сучасному світі дослідження у галузі застосування нейронних мереж в різних областях набувають все більшої популярності. В тому числі і в галузі виявлення руйнувань на дорогах. Було проведено аналіз літератури, що дозволило з'ясувати особливості та недоліки попередніх досліджень [15].

Зокрема, виявлено, що одним з найбільш ефективних підходів є використання глибоких нейронних мереж, зокрема, мереж типу Convolutional Neural Network (CNN), U-Net, Mask R-CNN та RetinaNet. Ці мережі мають високу точність виявлення руйнувань на дорогах та швидкість обробки великих обсягів даних.

Також виявлено недоліки попередніх досліджень. Використання стандартних алгоритмів обробки зображень може призводити до неправильної інтерпретації даних та зменшення точності. Одним із шляхів вирішення цієї проблеми є застосування передобробки даних, такої як збільшення об'єму даних та підвищення якості зображень за допомогою методів обробки зображень.

Також було виявлено, що для досягнення високої точності виявлення руйнувань на дорогах необхідно використовувати велику кількість даних для навчання та тестування нейронної мережі. При цьому важливо враховувати особливості різних типів дорожніх покриттів та погодних умов, що можуть вплинути на якість виявлення руйнувань.

Отже, проведений аналіз літератури та практичний експеримент реалізації даної роботи підтверджує актуальність та перспективність застосування нейронних мереж для виявлення руйнувань на дорогах. При цьому важливо враховувати особливості та недоліки попередніх досліджень та використовувати передові методи обробки даних та навчання нейронних мереж.

ПЕРЕЛІК ПОСИЛАНЬ

1. Gupta, A., Kumar, P., Singh, D. Damage detection on roads using deep learning: A review: Journal of Cleaner Production, 2021. 281c,.

2. Li, K., Huang, Y., Zhang, K., Li, Y., Li, X. A Novel Road Damage Detection Model Based on Deep Learning. Sensors, 2020.

3. Zhang, Q., Zuo, Y., Liu, X., Wang, H. Research on Road Damage Detection Based on Deep Learning.: In 2021 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference 2021. (IAEAC) (2937-2941c). IEEE.

4. Wu, Z., Liu, F., Huang, J., & Zhang, H. Road damage detection based on improved YOLOv3: Journal of Ambient Intelligence and Humanized Computing, 2021. 12(6), 6461-6471c.

5. Kuo, P. H., Kuo, P. H. An efficient method for road damage detection using deep learning and image enhancement techniques: Measurement, 2021. 174c.

6. Wu, L., Chen, S., Ye, Y., Peng, T. Road damage detection based on Mask R-CNN with anchor-free regression: Measurement, 2021. 184c.

7. Бурачинський, Ю., Коляденко, А., Корнійчук, Ю., Кравченко, А. Визначення ступеня пошкодження дорожнього покриття на основі обробки зображень методами машинного навчання: Науковий вісник НЛТУ України, 2021.127-133с.

8. Li, J., Li, M., Li, H., Gao, X. Road surface damage detection based on improved convolutional neural network: Journal of Advanced Transportation, 2021, 1-14c.

9. Liu, Y., Ren, H., Sun, Y. A road damage detection approach based on deep learning: Advances in Mechanical Engineering, 2020.

10. Колодій, А., Кравченко, А., Бобало, І. Дослідження підходів до виявлення пошкоджень дорожнього покриття на основі обробки зображень з

використанням нейронних мереж: Вісник Національного університету "Львівська політехніка". Комп’ютерні системи та мережі, 2020, 29-35с.

11. Mask R-CNN: архітектура сучасної нейронної мережі для сегментації об'єктів на зображеннях. Дата оновлення: 26.08.2018. URL: https://habr.com/ru/articles/421299

12. Бойко, І., Сахно, І., Попова, О. Розробка системи моніторингу дорожнього покриття на основі аналізу зображень: Електронні технології та засоби, 2019. 80-87с.

13. Мельник, І., Журба, О., Григорчук, І. Автоматичне виявлення дефектів дорожнього покриття за допомогою машинного навчання: Вісник НТУ "ХПІ". Серія: Інформатика та моделювання, 2020. 101-106с.

14. ConvNets. Створення прототипу проекту за допомогою Mask R-CNN. Дата оновлення: 09.05.2018. URL: https://habr.com/ru/companies/newprolab/articles/412523

15. Поліщук, І., Єфіменко, Д. Аналіз підходів до виявлення пошкоджень дорожнього покриття на основі зображень: Науковий вісник Полісся, 2019. 154-161с.

ДОДАТОК А

Приклад вигляду JSON файлу зображення


```
{
  "info": {
    "description": "pothole"
  },
  "images": [
    {
      "id": 1,
      "width": 3680,
      "height": 2760,
      "file_name": "G0027861.JPG"
    }
  ],
  "annotations": [
    {
      "id": 0,
      "iscrowd": 0,
      "image_id": 1,
      "segmentation": [
        [
          2529.0212765957435,
          1719.6170212765949,
          2466.382978723403,
          1713.744680851063,
          2313.7021276595733,
          1752.8936170212758,
          2327.404255319148,
          1774.4255319148926,
          2378.2978723404244,
          1776.3829787234033,
          2474.2127659574458,
          1764.6382978723395,
          2540.765957446807,
          1770.5106382978715,
          2599.489361702126,
          1760.7234042553182,
          2628.8510638297857,
          1750.936170212765,
          2685.6170212765946,
          1743.1063829787226,
          2654.2978723404244,
          1725.4893617021269,
          2619.063829787233,
          1721.5744680851055,
          2583.8297872340413,
          1713.744680851063
        ]
      ],
      "bbox": [
        2313.7021276595733,
        1713.744680851063,
        371.91489361702133,
        62.63829787234022
      ],
      "area": 14560.072430964217
    },
    {
      "id": 1,
      "iscrowd": 0,
      "image_id": 1,
      "segmentation": [
        [
          3225.87234042553,
          1915.3617021276586,
          3155.4042553191475,
          1925.1489361702118,
          3153.446808510637,
          1936.8936170212758,
          3214.1276595744666,
          1958.4255319148926,
          3268.9361702127644,
          1938.8510638297862
        ]
      ],
      "bbox": [
        3153.446808510637,
        1915.3617021276586,
        115.48936170212755,
        43.063829787234
      ],
      "area": 2902.4354911724804
    }
  ],
  "categories": []
}
```

ДОДАТОК Б

Файл pothole.py – основний файл класу PotholeDataset

```
import os
import sys
import json
import datetime
import numpy as np
import skimage.draw

ROOT_DIR = os.path.abspath("./")

sys.path.append(ROOT_DIR)
from config import Config
from mrcnn import model as modellib, utils

COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR,
"mask_rcnn_pothole.h5")

DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR,
"logs")


class PotholeConfig(Config):
    NAME = "pothole"

    IMAGES_PER_GPU = 2

    STEPS_PER_EPOCH = 100

    DETECTION_MIN_CONFIDENCE = 0.9

class PotholeDataset(utils.Dataset):

    def load_pothole(self, dataset_dir, subset):
        self.add_class("pothole", 1, "pothole")

        assert subset in ["train", "val"]
        dataset_dir = os.path.join(dataset_dir, subset)

        annotations =
json.load(open(os.path.join(dataset_dir,
"via_region_data.json")))
        annotations = list(annotations.values())  # don't
need the dict keys

        annotations = [a for a in annotations if
a['regions']]

        for a in annotations:
            if type(a['regions']) is dict:
                polygons = [r['shape_attributes'] for r in
a['regions'].values()]
            else:
                polygons = [r['shape_attributes'] for r in
a['regions']]
            image_path = os.path.join(dataset_dir,
a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "pothole",
                image_id=a['filename'],  # use file name as a
unique image id
                path=image_path,
                width=width, height=height,
                polygons=polygons)

    def load_mask(self, image_id):
        image_info = self.image_info[image_id]
        if image_info["source"] != "pothole":
            return super(self.__class__,
self).load_mask(image_id)

        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"],
len(info["polygons"])],
                        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
            rr, cc = skimage.draw.polygon(p['all_points_y'],
p['all_points_x'])
            mask[rr, cc, i] = 1

        return mask.astype(np.bool),
np.ones([mask.shape[-1]], dtype=np.int32)

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "pothole":
            return info["path"]
        else:
            super(self.__class__,
self).image_reference(image_id)


def train(model):
    dataset_train = PotholeDataset()
    dataset_train.load_pothole(args.dataset, "train")
    dataset_train.prepare()

    dataset_val = PotholeDataset()
    dataset_val.load_pothole(args.dataset, "val")
    dataset_val.prepare()

    print("Training network heads")
    model.train(dataset_train, dataset_val,
```

```
            learning_rate=config.LEARNING_RATE,
            epochs=30,
            layers='heads')


def color_splash(image, mask):
    gray =
skimage.color.gray2rgb(skimage.color.rgb2gray(image
)) * 255
    if mask.shape[-1] > 0:
        mask = (np.sum(mask, -1, keepdims=True) >= 1)
        splash = np.where(mask, image,
gray).astype(np.uint8)
    else:
        splash = gray.astype(np.uint8)
    return splash


def detect_and_color_splash(model,
image_path=None, video_path=None):
    assert image_path or video_path

    if image_path:
        print("Running on {}".format(args.image))
        image = skimage.io.imread(args.image)
        r = model.detect([image], verbose=1)[0]
        splash = color_splash(image, r['masks'])
        file_name =
"splash_{:%Y%m%dT%H%M%S}.png".format(dateti
me.datetime.now())
        skimage.io.imsave(file_name, splash)
    elif video_path:
        import cv2
        vcapture = cv2.VideoCapture(video_path)
        width =
int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
        height =
int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = vcapture.get(cv2.CAP_PROP_FPS)
        file_name =
"splash_{:%Y%m%dT%H%M%S}.avi".format(dateti
me.datetime.now())
        vwriter = cv2.VideoWriter(file_name,

cv2.VideoWriter_fourcc(*'MJPG'),
                                  fps, (width, height))

        count = 0
        success = True
        while success:
            print("frame: ", count)
            success, image = vcapture.read()
            if success:
                image = image[..., ::-1]
                r = model.detect([image], verbose=0)[0]
                splash = color_splash(image, r['masks'])
                splash = splash[..., ::-1]
                vwriter.write(splash)
                count += 1
        vwriter.release()
    print("Saved to ", file_name)

if __name__ == '__main__':
    import argparse

    # Parse command line arguments
    parser = argparse.ArgumentParser(
        description='Train Mask R-CNN to detect
potholes.')
    parser.add_argument("command",
                        metavar="<command>",
                        help="'train' or 'splash'")
    parser.add_argument('--dataset', required=False,
                        metavar="/path/to/pothole/dataset/",
                        help='Directory of the Pothole dataset')
    parser.add_argument('--weights', required=True,
                        metavar="/path/to/weights.h5",
                        help="Path to weights .h5 file or 'coco'")
    parser.add_argument('--logs', required=False,
                        default=DEFAULT_LOGS_DIR,
                        metavar="/path/to/logs/",
                        help='Logs and checkpoints directory
(default=logs/)')
    parser.add_argument('--image', required=False,
                        metavar="path or URL to image",
                        help='Image to apply the color splash
effect on')
    parser.add_argument('--video', required=False,
                        metavar="path or URL to video",
                        help='Video to apply the color splash
effect on')
    args = parser.parse_args()

    if args.command == "train":
        assert args.dataset, "Argument --dataset is
required for training"
    elif args.command == "splash":
        assert args.image or args.video,\
               "Provide --image or --video to apply color
splash"

    print("Weights: ", args.weights)
    print("Dataset: ", args.dataset)
    print("Logs: ", args.logs)

    # Configurations
    if args.command == "train":
        config = PotholeConfig()
    else:
        class InferenceConfig(PotholeConfig):
            GPU_COUNT = 1
            IMAGES_PER_GPU = 1
        config = InferenceConfig()
    config.display()

    if args.command == "train":
        model = modellib.MaskRCNN(mode="training",
config=config,
                                  model_dir=args.logs)
    else:
        model = modellib.MaskRCNN(mode="inference",
config=config,
                                  model_dir=args.logs)

    if args.weights.lower() == "coco":
        weights_path = COCO_WEIGHTS_PATH
```

```
        if not os.path.exists(weights_path):
            utils.download_trained_weights(weights_path)
    elif args.weights.lower() == "last":
        weights_path = model.find_last()
    elif args.weights.lower() == "imagenet":
        weights_path = model.get_imagenet_weights()
    else:
        weights_path = args.weights

    print("Loading weights ", weights_path)
    if args.weights.lower() == "coco":
        model.load_weights(weights_path,
by_name=True, exclude=[
            "mrcnn_class_logits", "mrcnn_bbox_fc",
            "mrcnn_bbox", "mrcnn_mask"])
    else:
        model.load_weights(weights_path,
by_name=True)

    if args.command == "train":
        train(model)
    elif args.command == "splash":
        detect_and_color_splash(model,
image_path=args.image,
                                video_path=args.video)
    else:
        print("'{}' is not recognized. "
              "Use 'train' or 'splash'".format(args.command))
```

ДОДАТОК В

Файл config.py – файл конфігурації

```
import numpy as np

class Config(object):

    NAME = None

    GPU_COUNT = 1

    IMAGES_PER_GPU = 2

    STEPS_PER_EPOCH = 1000

    VALIDATION_STEPS = 50

    BACKBONE = "resnet101"

    COMPUTE_BACKBONE_SHAPE = None

    BACKBONE_STRIDES = [4, 8, 16, 32, 64]

    FPN_CLASSIF_FC_LAYERS_SIZE = 1024

    TOP_DOWN_PYRAMID_SIZE = 256

    NUM_CLASSES = 1

    RPN_ANCHOR_SCALES = (32, 64, 128, 256, 512)

    RPN_ANCHOR_RATIOS = [0.5, 1, 2]

    RPN_ANCHOR_STRIDE = 1

    RPN_NMS_THRESHOLD = 0.7

    RPN_TRAIN_ANCHORS_PER_IMAGE = 256

    PRE_NMS_LIMIT = 6000

    POST_NMS_ROIS_TRAINING = 2000
    POST_NMS_ROIS_INFERENCE = 1000

    USE_MINI_MASK = True
    MINI_MASK_SHAPE = (56, 56)

    IMAGE_RESIZE_MODE = "square"
    IMAGE_MIN_DIM = 600
    IMAGE_MAX_DIM = 800

    IMAGE_MIN_SCALE = 0

    IMAGE_CHANNEL_COUNT = 3

    MEAN_PIXEL = np.array([123.7, 116.8, 103.9])

    TRAIN_ROIS_PER_IMAGE = 200

    ROI_POSITIVE_RATIO = 0.33

    POOL_SIZE = 7
    MASK_POOL_SIZE = 14

    MASK_SHAPE = [28, 28]

    MAX_GT_INSTANCES = 100

    RPN_BBOX_STD_DEV = np.array([0.1, 0.1, 0.2,
0.2])
    BBOX_STD_DEV = np.array([0.1, 0.1, 0.2, 0.2])

    DETECTION_MAX_INSTANCES = 100

    DETECTION_MIN_CONFIDENCE = 0.7

    DETECTION_NMS_THRESHOLD = 0.3

    LEARNING_RATE = 0.001
    LEARNING_MOMENTUM = 0.9

    WEIGHT_DECAY = 0.0001

    LOSS_WEIGHTS = {
        "rpn_class_loss": 1.,
        "rpn_bbox_loss": 1.,
        "mrcnn_class_loss": 1.,
        "mrcnn_bbox_loss": 1.,
        "mrcnn_mask_loss": 1.
    }

    USE_RPN_ROIS = True

    TRAIN_BN = False

    GRADIENT_CLIP_NORM = 5.0

    def __init__(self):
        self.BATCH_SIZE = self.IMAGES_PER_GPU *
self.GPU_COUNT

        # Input image size
        if self.IMAGE_RESIZE_MODE == "crop":
            self.IMAGE_SHAPE =
np.array([self.IMAGE_MIN_DIM,
self.IMAGE_MIN_DIM,
                self.IMAGE_CHANNEL_COUNT])
        else:
```

```
            self.IMAGE_SHAPE =
np.array([self.IMAGE_MAX_DIM,
self.IMAGE_MAX_DIM,
                self.IMAGE_CHANNEL_COUNT])

        self.IMAGE_META_SIZE = 1 + 3 + 3 + 4 + 1 +
self.NUM_CLASSES

    def display(self):
        """Display Configuration values."""
        print("\nConfigurations:")
        for a in dir(self):
            if not a.startswith("__") and not
callable(getattr(self, a)):
                print("{:30} {}".format(a, getattr(self, a)))
        print("\n")
```