МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувачка кафедри,
д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Розробка та вдосконалення алгоритмів управління БПЛА для місій пошуку та порятунку

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія»

Здобувач _____Павло ЛОСІЦЬКИЙ

підпис

«__» _____ 2024_ р.

Керівник д-р техн. наук, проф. _____ Ірина ЖУРАВСЬКА

підпис

«_ _» _____ 2024_ р.

Миколаїв – 2024

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____ Ірина ЖУРАВСЬКА

«_____» _____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Лосіцького Павла Михайловича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи
Розробка та вдосконалення алгоритмів управління БПЛА для місій пошуку та порятунку

Затверджена наказом ректора ЧНУ ім. Петра Могили від 16.09.2024 № 236.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом роботи є апаратно-програмний комплекс на базі Raspberry Pi, що дозволило б покращити якість визначення та категоризації об’єктів під час пошуково-рятувальних операцій.

4. Перелік питань, що підлягають розробці:
1) огляд сучасних методів та засобів які використовуються для місій пошуку та порятунку;
2) аналіз переваг та недоліків існуючих систем;
3) розробити апаратну частину комплексу, яка складається з Raspberry Pi 3 Model B та пристрою відеозахоплення;
4) розробити програмну частину комплексу, яка складається з фреймворку TensorFlow Lite в поєднані зі згортковою нейроною мережею GoogLeNet .

5. Перелік графічних матеріалів
слайди презентації

6. Завдання до спеціальної частини _

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ _Ірина ЖУРАВСЬКА_
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ _Павло ЛОСІЦЬКИЙ_
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної магістерської роботи

Тема: Розробка та вдосконалення алгоритмів управління БПЛА для місій пошуку та порятунку

№ з/п	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2024	16.09.2024	*Виконано*
2.	Огляд літератури за темою роботи	17.09.2024	22.09.2024	*Виконано*
3.	Складання календарного плану КМР	24.09.2024	25.09.2024	*Виконано*
4.	Аналіз предметної області	25.09.2024	27.09.2024	*Виконано*
5.	Розробка проєктних рішень	28.09.2024	30.09.2024	*Виконано*
6.	Моделювання	01.10.2024	02.10.2024	*Виконано*
7.	Конструювання АПК	02.10.2024	04.10.2024	*Виконано*
8.	Перевірка працездатності, тестування та апробація розробленого АПК	05.10.2024	05.10.2024	*Виконано*
9.	Аналіз результатів тестування	15.10.2024	20.10.2024	*Виконано*
10.	Оформлення КМР та презентації	05.11.2024	20.11.2024	*Виконано*
11.	Відгук керівника КМР	25.11.2024	27.11.2024	*Виконано*
12.	Попередній захист	28.11.2024	28.11.2024	*Виконано*
13.	Рецензування			*Виконано*
14.	Захист кваліфікаційної роботи	19.12.2024	20.12.2024	*Виконано*

Керівник роботи _____ Ірина ЖУРАВСЬКА
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Павло ЛОСІЦЬКИЙ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи
«Розробка та вдосконалення алгоритмів управління БПЛА для місій пошуку та порятунку»
Здобувач гр. 605м: Лосіцький Павло Михайлович
Керівник: завідувачка кафедри КІ, д-р техн. наук, професор Журавська І. М.

У даній роботі проведено дослідження алгоритмів управління безпілотними літальними апаратами (БПЛА) для виконання місій пошуку та порятунку. Розробка таких алгоритмів є актуальною через зростаючий попит на автоматизовані системи, здатні виконувати складні завдання у важкодоступних чи небезпечних умовах.

Об'єктом дослідження є процес управління БПЛА для пошуково-рятувальних операцій. Предметом дослідження є алгоритми навігації та ухилення від перешкод, а також методи збору й аналізу даних під час виконання місій.

Метою роботи є підвищення ефективності здійснення пошукових операцій за рахунок розробки і вдосконалення алгоритмів управління БПЛА, що дозволяють враховувати особливості місцевості, погодні умови та інші фактори, які можуть впливати на виконання місій.

Практична значимість дослідження полягає в підвищенні ефективності рятувальних операцій за допомогою БПЛА, що зможе скоротити час реагування та збільшити шанси на успіх у складних ситуаціях.

Результати роботи можуть бути використані в реальних рятувальних операціях, а також сприяти подальшому розвитку технологій автономного управління БПЛА у різних галузях, зокрема в обороні, логістиці та сільському господарстві.

Робота включає дослідження сучасних технологій навігації та автоматизації БПЛА, аналіз існуючих методів пошуку та порятунку, а також тестування розроблених алгоритмів. Робота складається з 64 сторінок (без додатків), 1 табл., 34 рис. та 2 додатків. У процесі дослідження було використано 23 джерела посилання.

Ключові слова: БПЛА, алгоритми управління, пошуково-рятувальні операції, Raspberry Pi, TensorFlow, згорткова нейронна мережа, GoogLeNet, навчання моделей, категоризація об’єктів порятунку.

ABSTRACT

of the Master's Thesis

«Development and Improvement of UAV Control Algorithms for Search and Rescue Missions»

Applicant: Lositskyi Pavlo Mykhailovych

Superviser: Head of the Computer Engineering Department, Doctor of Technical Sciences, Professor Zhuravska Iryna Mykolaivna

This study explores management algorithms for unmanned aerial vehicles (UAVs) used in search and rescue missions. The development of such algorithms is highly relevant due to the growing demand for automated systems capable of performing complex tasks in hard-to-reach or hazardous environments.

The object of the research is the UAV management system designed for search and rescue operations. The subject of the research includes navigation and obstacle avoidance algorithms, as well as data collection and analysis methods employed during mission execution.

The aim of the work is to increase the efficiency of search operations by developing and improving UAV control algorithms that consider terrain features, weather conditions, and other factors that may affect mission performance.

This study's practical significance lies in enhancing the effectiveness of rescue operations by using UAVs, reducing response time, and increasing success rates in complex situations.

The results of this work may be applicable in real rescue operations. They could contribute to further advancements in autonomous UAV management technologies across various sectors, including defense, logistics, and agriculture. The work involves researching modern UAV navigation and automation technologies, analyzing existing search and rescue methods, and testing the developed algorithms. The Master's Thesis comprises 64 pages (excluding appendices), 1 table, 34 figures, and 2 appendices. The study references 23 sources.

Keywords: UAV, management algorithms, search and rescue operations, Raspberry Pi, TensorFlow, CNN, GoogLeNet, model training, rescue object categorization.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БПЛА – безпілотний літальний апарат

ОП – оперативна пам’ять

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ШІ – штучний інтелект

CNN – Convolutional Neural Network

CSI – Camera Serial Interface

GNSS – Global Navigation Satellite System

GPS – Global Positioning System

PID – Proportional–Integral–Derivative Controller

RTH – Return-to-Home

RTK – Real-Time Kinematic

SLAM – Simultaneous Localization and Mapping

SSD – Single Shot MultiBox Detector

VIO – Visual Inertial Odometry

YOLO – You Only Look Once

ВСТУП

Сьогодні використання безпілотних літальних апаратів (БПЛА або дронів) стає все більш популярним завдяки їх полівалентності та здатності виконувати різноманітні функції в різноманітних середовищах. Застосування БПЛА є особливо важливим у сфері пошуку та рятування, оскільки забезпечує швидке та ефективне реагування на надзвичайні ситуації, що виникають у гірській місцевості, зонах стихійного лиха чи поблизу завалів.

БПЛА мають різноманітні можливості: вони можуть спостерігати за ландшафтом зверху, можуть передавати відео та зображення в режимі реального часу, ідентифікувати жертв та брати участь у рятувальних операціях. Завдяки вдосконаленій природі алгоритмів комп’ютерного зору, машинного навчання та штучного інтелекту ці пристрої здатні не лише розпізнавати об’єкти на землі, а й розрізняти їх, що має велике значення для ефективного пошуку та порятунку.

Об'єктом роботи є процес управління алгоритмами БПЛА під час пошуково-рятувальних операцій.

Предметом роботи є методи та алгоритми комп’ютерного збору, аналізу даних та категоризація об’єктів, які можуть використовувати БПЛА під час пошуку цілей.

Метою даної роботи є підвищення ефективності здійснення пошукових операцій за рахунок розробки і вдосконалення алгоритмів управління БПЛА, що дозволяють враховувати особливості місцевості, погодні умови та інші фактори, які можуть впливати на виконання місій. В роботі запропоновано здійснювати категоризацію об’єктів (замість ресурсоємного розпізнавання) за допомогою штучного інтелекту. Це сприятиме підвищенню точності та швидкості реагування під час пошуково-рятувальних операцій, дозволяючи БПЛА автоматично розпізнавати об’єкти в різноманітних ситуаціях та оперативно інформувати рятувальні команди про знайдених постраждалих.

Для досягнення мети слід виконати такі завдання:

1) провести аналіз моделей, алгоритмів управління та вимог до БПЛА для пошуково-рятувальних місій;

2) провести моделювання та проєктування системи для пошуково-рятувальних операцій за допомогою БПЛА;

3) розробити алгоритми, які можуть розрізняти людей, тварин або предмети;

4) реалізувати систему категоризації об’єктів для порятунку на базі Raspberry PI та Tensorflow Lite;

5) виконати тестування апаратно-програмної системи для місій пошуку та порятунку з використанням БПЛА.

Практична значимість цього дослідження полягає у вдосконаленні процесів аварійно-рятувальних робіт за допомогою БПЛА, що дозволить значно зменшити час реагування на надзвичайні ситуації, а також підвищити точність ідентифікації постраждалих і збільшити ефективність роботи рятувальних команд. Розроблені алгоритми мають потенціал для застосування в інших сферах, зокрема в військовій справі, екологічному моніторингу, інспекції інфраструктури та цивільній безпеці. У цих сферах точна ідентифікація об'єктів та їх аналіз є критично важливими завданнями.

Таким чином, вдосконалення алгоритмів управління БПЛА для пошуково-рятувальних місій є важливим кроком, що може значно підвищити ефективність таких операцій і сприяти розвитку нових технологій, які матимуть вплив на широкий спектр галузей людської діяльності.

Дана кваліфікаційна робота пройшла апробацію на XXI Міжнародній науковій конференції «Ольвійський форум – 2024: стратегії країн Причорноморського регіону в геополітичному просторі». Матеріали доповіді були опубліковані у збірнику конференції [1].

1. АНАЛІЗ МОДЕЛЕЙ, АЛГОРИТМІВ УПРАВЛІННЯ ТА ВИМОГ ДО БПЛА ДЛЯ ПОШУКОВО-РЯТУВАЛЬНИХ МІСІЙ

Сьогодні БПЛА відіграють значну роль у пошуково-рятувальних операціях, ці апарати забезпечують швидке та ефективне обстеження територій, які мають обмежений або небезпечний доступ для людини. Особливий інтерес представляє концептуалізація «дронів швидкої допомоги», яка обговорюється в дослідженні, яке аналізує дизайн і створення дрона, здатного виявляти людей і надавати першу допомогу. Концепції цих дронів описали S. Mano та V.M. Sreehari в 6 розділі книги «Internet of Drones». (див. «Розділ 6. Дрон швидкої допомоги для порятунку – Концептуальний дизайн, системи виявлення життя та розробку прототипів) [2].

Ця ідея поєднує виявлення життя з використанням роботів, особливо щодо стабільності та руху дронів, а також обмеженого простору або складної місцевості.

Окремою групою серед дронів є так звані SAR-дрони або пошуково-рятувальні дрони (англ. Search and Rescue, SAR) – це БПЛА, які використовуються для пошуку та встановлення місцезнаходження людей, які зникли безвісти або постраждали в надзвичайних ситуаціях [3].

1.1 Класифікація SAR-дронів

Пошуково-рятувальні дрони доступні в різних розмірах, від невеликих портативних моделей до більших, більш надійних безпілотних літальних систем (БПЛА). На розмір пошуково-рятувального безпілотника впливають такі фактори, як його призначення, корисне навантаження, дальність польоту та конкретні цілі місії. Ось список поширених типів пошуково-рятувальних дронів залежно від розміру.

Малі/міні-дрони: маленькі, компактні та легкі зазвичай використовуються для пошуку та порятунку. Їх може пересувати одна людина і запускати вручну. Ці дрони зазвичай важать 2 кг, а розмах крил становить приблизно 30–50 см. Незважаючи на їхні менші розміри, вони все ще здатні перевозити камери, які спостерігають за тепловими властивостями об’єктів, та інше обладнання, необхідне для пошуку та порятунку.

Безпілотники середнього розміру: пошуково-рятувальні безпілотні літальні апарати середнього розміру більші та потужніші, ніж їхні менші аналоги. Зазвичай вони мають вагу 2–25 кг і розмах крил приблизно 1–10 см. Ці безпілотники мають більші можливості для більш просунутих датчиків, мають більший час польоту та можуть працювати в більш несприятливих умовах. Зазвичай їм потрібна спеціальна зона для запуску та посадки через їхній об’єм.

Великі/професійні дрони: доступні як великі дрони, призначені для загального використання, так і великі пошуково-рятувальні дрони. Вони можуть мати вагу понад 25 кг і мати розмах крил у кілька метрів. Ці безпілотники можуть нагадувати малі літальні апарати, і для їх обслуговування та керування потрібна спеціальна команда. Вони можуть виконувати велике корисне навантаження, включаючи спеціалізовані датчики, численні камери та передові комунікаційні пристрої.

1.2 Транспортувальні дрони

«Ambulance Drone for Rescue» описує концепцію розробки та впровадження дрона, який за своєю конструкцією та розробкою схожий на автомобіль швидкої допомоги, особливо в ситуаціях, коли швидка доставка запасів або надання першої допомоги є надзвичайно важливими [4]. Розділ присвячений системам виявлення життя та роботизованим механізмам, які дозволяють безпілотникам не тільки швидко дістатися до жертв, але й використовувати різні інструменти екстреної допомоги. Дрони мають як рушії

для польоту, так і колеса для пересування по поверхні, які дозволяють обмежити політ (наприклад, у приміщенні або у високій траві).

Основними компонентами систем є роботизовані руки, виготовлені з композитного матеріалу, а також механізм, який нахиляє безпілотник для підтримки його стабільності.

Star-X VTOL-6520HP – безпілотний літальний апарат вертикального зльоту та посадки (VTOL), призначений для комплексного моніторингу, виявлення та рятування. Завдяки поєднанню можливостей роторних і крилатих дронів, цей апарат може злітати і приземлятися у вертикальному режимі, цей режим корисний на обмежених територіях і складних ландшафтах [5].

Дрон оснащений потужними двигунами та довгою батареєю, що дозволить йому літати до 2 годин. Це сприяє ідеальному застосуванню довготривалих операцій, таких як пошуково-рятувальні роботи та моніторинг інфраструктури. Додаткові функції включають корисні навантаження, які містять камери, теплові камери та датчики, які сприяють ефективному збору даних і аналізу рельєфу в режимі реального часу.

Star-X VTOL-6520HP відрізняється високою стійкістю до погодних умов і здатністю до автономного польоту з можливістю уникати перешкод, що підвищує безпеку і точність виконання завдань у складних зонах або зонах високого ризику для людини.

Дослідження Жанети Пржиходової присвячені створенню концепції рятувального дрона, який призначений для використання у важкодоступних районах, зокрема в гірській місцевості, під час пошуково-рятувальних робіт [6]. Концепція проєкту полягає у створенні одного з перших автономних безпілотників, здатних не тільки доставляти рятувальників на місце події, але й евакуювати постраждалих і надавати їм першу допомогу. На відміну від традиційних дронів, цей літальний апарат має унікальну конструкцію:

рятувальник знаходиться всередині літального апарату і керує ним за допомогою планшета або джойстика.

Основною інновацією цього безпілотника є поєднання автономних транспортних функцій, які включають рятувальну місію, що дозволяє швидко переміщати рятувальника до жертви, а також забезпечує безпечний засіб транспортування в місцях, де інші методи неефективні або становлять ризик. Дрон виготовлено з легких і міцних матеріалів, які зосереджені на ефективності електродвигунів і здатності до швидкої мобілізації. Також він укомплектований необхідним медичним обладнанням, що дозволяє рятувальникам негайно надати першу допомогу. Важливою складовою проєкту є вивчення ринку та дослідження супутніх технологій, які підвищать ефективність безпілотника, зокрема, використання передових матеріалів та джерел енергії.

Цей проєкт описує функцію дронів у майбутньому в аварійно-рятувальній діяльності та визначає кілька технічних і нормативних питань, включаючи безпеку пристрою, його безперервну роботу та дизайн пристрою для складних умов.

1.3 Пошукові БПЛА

У сучасних SAR-рішеннях використовуються різні дрони, серед яких виділяються DJI Matrice 300 RTK, Parrot Anafi USA, і Autel Robotics EVO II Dual [7–9]. Ці апарати відомі своїми високотехнологічними рішеннями для навігації та аналізу даних.

DJI Matrice 300 RTK, наприклад, оснащений алгоритмами на основі машинного навчання для розпізнавання об'єктів та стабільної роботи в умовах перешкод. Parrot Anafi USA використовує інфрачервоні сенсори та алгоритми автоматичного виявлення теплових слідів, що є важливим для пошуку постраждалих у важкодоступних місцях. Autel Robotics EVO II Dual застосовує технології глибокого навчання для обробки відеопотоку в

реальному часі, що допомагає БПЛА ефективно визначати маршрути та уникати перешкод.

Таким чином, розвиток і вдосконалення алгоритмів навігації, ухилення від перешкод та аналізу даних є ключовими факторами підвищення ефективності пошуково-рятувальних місій за допомогою БПЛА.

1.3.1 DJI Matrice 300 RTK

DJI Matrice 300 RTK – це дійсно сучасний БПЛА, який дуже популярний як для пошукових, так і для рятувальних операцій завдяки своїй точності, надійності та автономності (рис 1.1). Дрон має модульну конструкцію з можливістю підвішування трьох корисних навантажень, включаючи камери та тепловізори. Дальність польоту сягає до 15 км, а робота триває 55 хвилин.

Рисунок 1.1 – Matrice 300 RTK [7]

Однією з найважливіших особливостей Matrice 300 RTK є його система RTK (Real Time Kinematic, укр. «кінематика реального часу»), яка забезпечує точне позиціонування дрона на рівні сантиметра навіть при використанні в складних умовах. Дрон буде використовувати шестиосьові датчики, а також алгоритми планування траєкторії для автономного вибору безпечного маршруту, ухиляючись від перешкод за допомогою VIO (Visual Inertial Odometry, укр. «візуальна інерціальна одометрія»).

Функції розпізнавання об’єктів і відстеження об’єктів можна безпосередньо віднести до розширених алгоритмів машинного навчання, вбудованих у Matrice 300 RTK. Під час польоту користувачі можуть тримати об’єкт у центрі кадрів за допомогою системи Smart Track, тоді як маршрути можна зберігати та автоматизувати завдання перевірки за допомогою Live Mission Recording та алгоритмів AI Spot-Check.

Крім того, він оснащений тепловізорами, які допомагають виявляти сліди тепла, а OcuSync Enterprise гарантує передавання даних на відстані до 15 км.

Він використовує алгоритми навігації, розпізнавання об’єктів і уникнення перешкод для виконання функцій у середовищі, де подія потребує значної автономності.

1.3.2 Parrot Anafi USA

Parrot Anafi USA – сучасний дрон, призначений для виконання професійних завдань у пошуково-рятувальних операціях; під час перевірок; і в безпеці (рис 1.2). Його мінімалістичний дизайн (500 г) і невеликий розмір роблять його досить портативним; і він може залишатися в польоті близько 32 хв, зберігаючи радіус зв'язку близько 5 км. Дві оптичні камери, а також датчики тепловізора гарантують точність виявлення об’єктів дроном навіть за складних обставин.

Рисунок 1.2 – Parrot Anafi USA [8]

Основна камера – це камера з 32-кратним збільшенням, що забезпечує відстань до 5 км. Це робить дуже зручним сканування великих площ. Тепловізори FLIR Boson виявляють теплові сигнатури, тому ви можете помітити людей вночі чи під уламками. Алгоритми автоматично об’єднують теплові та оптичні дані для створення дуже детальних теплових карт для ідентифікації жертв або джерел небезпек

Він має точність позиціонування безпеки GPS, автоматично повертається до місця запуску, коли він втрачає сигнал або якщо акумулятор розряджається, алгоритми штучного інтелекту покращують стабільність зображення та кращу якість відео, а захист IP53 від проникнення пилу та води дає змогу використовувати дрон навіть у несприятливих умовах. Зашифроване з’єднання AES-XTS із довжиною 512-бітного ключа робить це надійним з’єднанням для розгортання урядових або екстрених служб, де безпека даних є важливою проблемою.

Parrot Anafi USA – це невеликий, але потужний пакет, який підходить для пошуково-рятувальних операцій, укомплектований обладнанням для виявленням теплової сигнатури, високою якістю фотографій і захистом від зовнішнього впливу.

1.3.3 Autel Robotics EVO II Dual

Дрон Autel Robotics EVO II Dual з оптичною камерою 8K і тепловізором FLIR Boson добре підходить для пошуку та перевірки в суворих умовах; це справжня сила (рис. 1.3). Справжній потенціал оптичної камери та тепловізійної камери, що працюють разом, полягає в кращій ідентифікації теплової сигнатури цілі в повній темряві, тумані чи диму. Він має тривалість польоту 40 хвилин і має радіус дії 9 км.

Рисунок 1.3 – Autel Robotics EVO II Dual [9]

Подвійний квадрокоптер EVO II поставляється з дванадцятьма основними камерами разом із 12 датчиками уникнення перешкод у шести основних напрямках для автоматизації уникнення зіткнень. Версія 2.0 може похвалитися підтримкою інтелектуального відстеження об’єктів за допомогою використання алгоритмів штучного інтелекту. Тепловізійна матриця дозволить регулювати колірні палітри зображень, допомагаючи операторам адаптуватися до конкретних умов.

Алгоритми агрегували тепловізійні та оптичні дані в єдину повну картину місцевості, багатопотокова оптимізація відео забезпечує безперебійну та стабільну на великі відстані передачу високоякісного відео. Корпус EVO II Dual розрахований на екстремальні температури (від мінус 10 °C до + 40 °C), а шифрування AES-256 зберігає дані в безпеці, що робить його придатним для використання урядом і рятувальними службами.

Autel EVO II Dual, безумовно, є універсальним пристроєм із передовими оптичними та тепловізійними технологіями, завдяки якому рятувальні операції та перевірки в несприятливих режимах і умовах можна легко виконувати за допомогою автономних режимів польоту та штучного інтелекту.

1.4 Огляд та аналіз алгоритмів управління БПЛА для пошуково-рятувальних місій

Сучасні пошуково-рятувальні операції з БПЛА мають великий компонент алгоритмічного виявлення та ідентифікації. Вони розділені на традиційні методи обробки зображень і сучасні обчислювальні методи, засновані на штучному інтелекті (ШІ). Основна мета цих алгоритмів – розпізнавати об’єкти в різних середовищах, зокрема в умовах поганої видимості або в гірській місцевості.

Одним із найбільш класичних підходів, який часто використовується в задачах розпізнавання, є метод гістограм і порогового аналізу для виявлення меж об'єктів. Цей метод забезпечує швидку обробку даних, але має проблеми з наявністю низької видимості або складної структури в об’єктах, він зазвичай поєднується з іншим методом, який є більш адаптивним.

Сучасні БПЛА все більше використовують глибокі згорткові нейронні мережі (англ. Convolutional Neural Network, CNN) для розпізнавання об’єктів у відео, отриманих з камер. Одним із прикладів є YOLO (англ. You Only Look Once), який дозволяє виявляти та класифікувати об’єкти на зображенні з високою швидкістю та високою точністю [10]. YOLO використовується в режимі реального часу та ефективно використовується в надзвичайних ситуаціях для швидкої ідентифікації людей, тварин або об’єктів у небезпечних середовищах.

Іншою популярною моделлю є SSD (Single Shot MultiBox Detector), який також забезпечує розширену форму розпізнавання об’єктів, подібну до YOLO, але відрізняється підходом до обробки зображень [11]. SSD сегментує зображення на кілька областей і виконує виявлення кожної з них, що дозволяє досягти високого ступеня точності. Завдяки своїй ефективності та швидкості SSD є ідеальним вибором для вирішення проблем людей або інших об’єктів, які потребують швидкого реагування.

Іншим важливим засобом для розпізнавання об’єктів є нейромережа GoogLeNet (Inception v1–v4), яка завдяки своїй конструкції здатна розпізнавати об’єкти на зображеннях різного розміру та в різних умовах освітлення. GoogLeNet використовує багато різних процесорів і рівнів абстракції, що дозволяє отримати точні результати в ситуаціях, коли освітлення обмежене або природне середовище складне [12].

Камери, які мають тепловізійні можливості, використовуються для розпізнавання температурних аномалій і розпізнавання об’єктів, які неможливо побачити або мають обмежену видимість. У поєднанні з нейронними мережами, такими як YOLO або SSD, теплові камери можна використовувати для визначення присутності людини на основі температури її теплових сигнатур. Ці алгоритми обробляють інформацію в реальному часі та швидко визначають температурні перепади, що значно підвищує ефективність пошуково-рятувальних робіт.

Сучасні моделі ШІ також полегшують навчання обмеженої кількості типів даних, що дозволяє їм швидко адаптуватися до конкретних географічних регіонів або типів об’єктів. Завдяки своїй гнучкості та точності алгоритми на основі глибокого навчання є важливими інструментами для автоматизації процесу розпізнавання об’єктів, що дозволяє швидко приймати рішення в складних динамічних середовищах.

1.5 Специфікація вимог до апаратно-програмного комплексу

Апаратно-програмний комплекс для проведення пошуково-рятувальних робіт з використанням безпілотників призначений для забезпечення ефективного розпізнавання та ідентифікації об’єктів у режимі реального часу. Система повинна працювати на основі використання сучасних технологій комп’ютерного зору та методів навчання, наприклад, моделі GoogLeNet InceptionV3, яка здатна категоризувати об’єкти на зображеннях та відеопотоках, отриманих дроном під час його місії.

Система має забезпечувати високу точність класифікації, що є основною вимогою для її роботи. Модель InceptionV3 дозволяє ідентифікувати людей, тварин та інші об'єкти навіть в умовах обмеженої видимості, таких як погане освітлення, погодні умови або туман.

Після категоризації система повинна автоматично формувати рекомендації щодо подальших дій. Наприклад, коли виявлено людину, яка потребує термінової допомоги, система може надати пропозиції для швидкого реагування.

Результати аналізу та рекомендації мають надсилаються на сервери або хмарне сховище для забезпечення зберігання та передачі даних. Це дозволяє координаторам рятувальних операцій отримувати доступ до необхідної інформації в режимі реального часу та зберігати всю історію виявлених об’єктів і виконаних дій для подальшого аналізу та планування майбутніх місій.

Система має бути розроблена з урахуванням обмежень, які можуть виникнути під час роботи, таких як обмежена енергія або обчислювальна потужність. Для цього використовуються оптимізовані алгоритми, які забезпечують баланс між ефективністю роботи та мінімальним споживанням ресурсів. Це особливо важливо для довгострокових завдань, коли система повинна працювати стабільно без перебоїв.

Програмне забезпечення враховує критичний характер таких завдань за часом. Усі компоненти оптимізовані для максимальної швидкості обробки даних, що дозволяє миттєво приймати рішення навіть у складних умовах, таких як нестабільний зв’язок або екстремальні погодні умови. Це забезпечує високу надійність і безперебійну роботу комплексу, що є ключовим фактором успішного виконання пошуково-рятувальних завдань.

Висновок до розділу 1

Аналіз сучасних рішень і технологій, які застосовуються у безпілотних літальних апаратах для пошуково-рятувальних операцій, показав, що розглянуті моделі – DJI Matrice 300 RTK, Parrot Anafi USA та Autel Robotics EVO II Dual – мають унікальні технічні переваги, що робить їх ефективними інструментами для складних завдань. DJI Matrice 300 RTK вирізняється високою точністю завдяки системі RTK та потужним сенсорам, що важливо у місіях зі складним ландшафтом. Parrot Anafi USA забезпечує компактність, довготривалу автономну роботу й високу точність тепловізійних сенсорів, що дозволяє його використання в мобільних і складних умовах. Autel EVO II Dual, завдяки поєднанню оптичної камери з роздільною здатністю 8K і тепловізора, ефективно сканує місцевість і виявляє об'єкти на великих відстанях.

Попри високий технічний рівень, існують певні проблеми, які потребують подальшого вдосконалення: точність виявлення об'єктів у складних умовах, як-от погана видимість, туман чи дим, потребує інтеграції теплових та оптичних даних для поліпшення результатів. Також обмеження, пов’язані з використанням одного сенсора, можуть бути подолані через розробку багатосенсорних систем, що підвищить загальну ефективність.

У кваліфікаційній роботі частково розглядаються підходи до вирішення цих проблем, зокрема вдосконалення алгоритмів для розпізнавання об'єктів, що сприятиме підвищенню точності та ефективності дронів у пошуково-рятувальних місіях.

2. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ ДЛЯ ПОШУКОВО-РЯТУВАЛЬНИХ ОПЕРАЦІЙ ЗА ДОПОМОГОЮ БПЛА

Використання бібліотек глибокого машинного навчання є першим кроком до побудови автономної системи управління для БПЛА, призначеного для пошуково-рятувальних операцій. З їх допомогою можуть бути реалізовані такі методики, як алгоритми обробки зображень, виявлення об'єктів, уникнення перешкод і планування маршруту. А бібліотеки машинного навчання пропонують усе необхідне для роботи з нейронними мережами та обробки великої кількості даних у реальному часі; це допомагає створити надпотужну та ефективну систему обробки інформації в умовах обмежених обчислювальних ресурсів дрона.

Популярними бібліотеками машинного навчання в цьому проєкті є TensorFlow, PyTorch, Keras, OpenCV. Кожен з них має свої особливості та переваги:

1) TensorFlow: він розроблений Google і є найефективнішою реалізацією для серверних, мобільних і вбудованих пристроїв. TensorFlow є основою вибору для цього дослідження, і, зокрема, він дозволяє використовувати звичайні нейронні мережі. Відомо, що вони відіграють важливу роль в обробці зображень. Однією з головних причин, чому рекомендується TensorFlow, є те, що він сумісний із TensorFlow Lite, де моделі можна оптимізувати для БПЛА, які мають низькі обчислювальні можливості;

2) PyTorch – чудова гнучка та динамічна бібліотека, створена Facebook спеціально для досліджень. Природа швидкого прототипування робить його популярним у наукових проєктах, а також легкий для розуміння синтаксис. Він забезпечує підтримку GPU, що дозволяє виконувати складні обчислення для навчання моделі; однак це не найкращий вибір щодо вбудованих систем;

3) Keras, ймовірно, є однією з найбільш широко використовуваних бібліотек для побудови нейронної мережі зі стислим способом побудови моделей. У цьому дусі Keras насправді існував ізольовано, але потім був

включений у TensorFlow, щоб він слугував інтерфейсом для роботи з TensorFlow із максимальним його використанням;

4) OpenCV – це бібліотека обробки зображень, яка містить основні примітиви комп’ютерного зору для фільтрації, обробки зображень і виявлення об’єктів. Обробка відео в реальному часі надзвичайно швидка з OpenCV, і її можна об’єднати з TensorFlow та іншими нейронними мережами.

З усіх цих бібліотек TensorFlow виявляється більш вигідним вибором для створення систем керування БПЛА, оскільки вона оптимізована для мобільних пристроїв і підтримує TensorFlow Lite. Це дає можливість отримати високоточні обчислення на мізерних ресурсах – тому продуктивність дрона буде фактично оптимізована і збільшити час польоту за рахунок зменшення енергоспоживання. Оскільки TensorFlow також підтримує розподіл обчислень, це забезпечує рівень продуктивності, завдяки чому великі обсяги даних у реальному часі обробляються більш ефективно, що є більш ефективним для автономного пошуку, а також для завдання виявлення об’єктів.

Іншими словами, він надає всі ці функції, забезпечує стабільність, швидкість лікування та адаптивність. Отже, це означає, що це найкращий вибір при розробці проєкту автономного керування БПЛА в умовах складних пошуково-рятувальних завдань.

Наступна діаграма, зображена на рис. 2.1, демонструє два основних етапи архітектури TensorFlow: навчання моделі та розгортання. Спочатку зчитування та обробка даних, а потім навчання моделей за допомогою бібліотеки tf.keras або попередньо навчених моделей у TensorFlow Hub. Стратегія розподілу для збільшення на CPU, GPU або TPU. Отже, після того, як модель навчена, вона зберігається та готова до розгортання. Розгортання можна використовувати за допомогою TensorFlow Serving, як для хмарних, так і для локальних серверів, TensorFlow Lite для мобільних і вбудованих пристроїв, наприклад, або TensorFlow.js для браузера чи Node.js, а також програмування на всіх мовах.

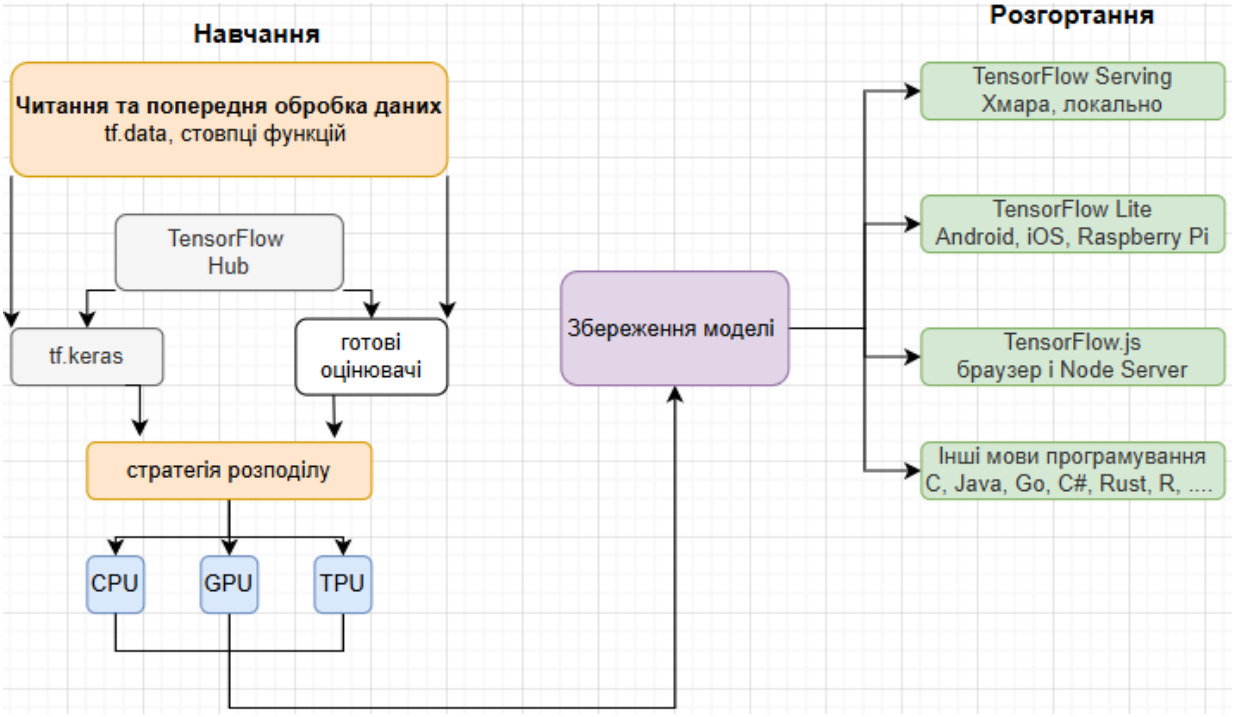


Рисунок 2.1 – Загальна архітектура платформи TensorFlow

TensorFlow надає повну платформу для побудови, навчання та використання моделей машинного навчання, зокрема, вона присвячена використанню нейронних мереж і представлення обчислень на основі графіків.

2.1 Використання GoogLeNet на базі TensorFlow

GoogLeNet є однією з найефективніших структур згорткової нейронної мережі (Convolutional Neural Network, CNN), яка вирізняється своєю глибиною та здатністю виконувати категоризацію та класифікацію об'єктів з високою точністю за мінімальних обчислювальних витрат. Завдяки використанню TensorFlow ця архітектура дозволяє створювати та навчати моделі, здатні працювати в реальному часі, що особливо важливо для автономних систем, таких як безпілотні літальні апарати (БПЛА). Модулі Inception, які є основою GoogLeNet, дозволяють оптимізувати обчислювальні ресурси за допомогою паралельних згорткових шарів з різними розмірами ядер (1×1, 3×3, 5×5), що забезпечує ефективну обробку даних і зменшення розмірностей (рис. 2.2).

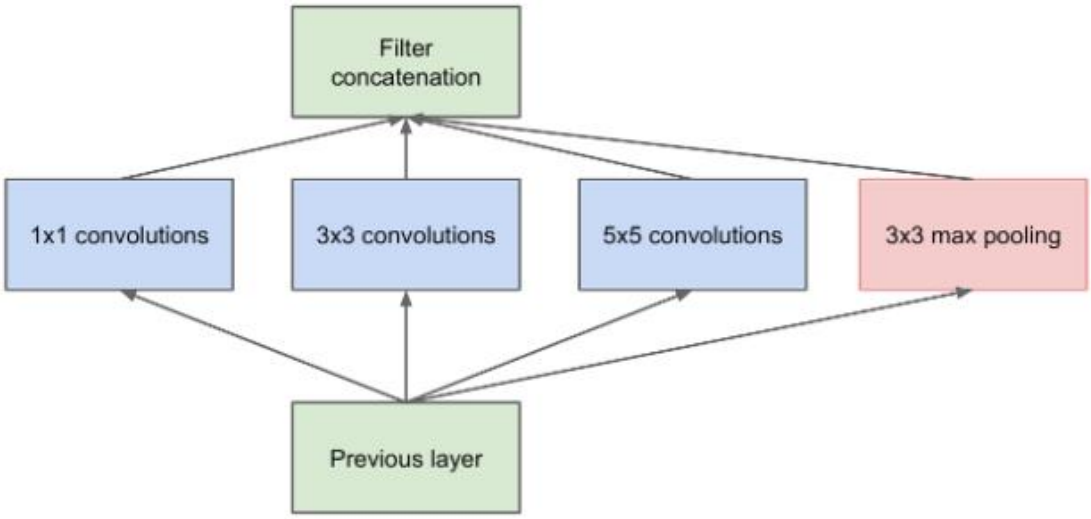


Рисунок 2.2 – Початковий модуль без зменшення розмірності [12]

TensorFlow виступає ключовим інструментом для реалізації цієї архітектури, пропонуючи підтримку обчислювальних графіків, оптимізованих алгоритмів, таких як Adam та RMSprop, а також можливість навчання на великих обсягах даних. Завдяки TensorFlow Lite можливе квантування моделей, що зменшує їх розмір та використання пам’яті, роблячи їх придатними для застосування в мобільних і вбудованих системах. Це особливо важливо для БПЛА, де обмежені обчислювальні ресурси й енергоспоживання грають вирішальну роль.

Архітектура Inception побудована на концепції оптимізації локальних розріджених структур мережі за допомогою доступних щільних компонентів. Ключова ідея полягає в одночасному використанні згорток різного розміру та ядра 1×1 для попереднього зменшення розмірності. Це дозволяє зменшити обчислювальні витрати, зберігаючи ефективність і точність. На вищих рівнях мережі збільшується кількість згорток 3×3 і 5×5, оскільки вони здатні виділяти більш абстрактні ознаки. Однак використання великих згорток на початкових рівнях може бути ресурсомістким, тому додавання шарів 1×1 допомагає значно зменшити обчислювальну складність (рис. 2.3).

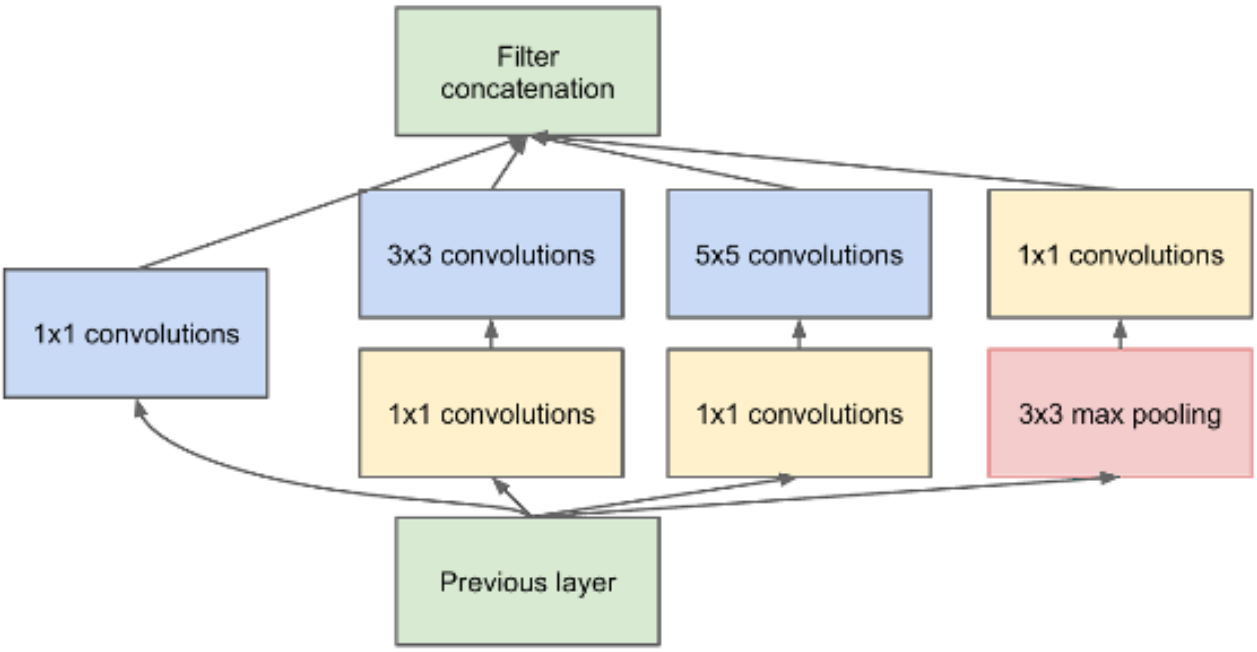


Рисунок 2.3 – Початковий модуль із зменшенням розмірності [12]

Переваги архітектури Inception включають можливість додавання більшої кількості рівнів без пропорційного зростання обчислювальних витрат. Це досягається завдяки зменшенню розмірності перед виконанням більш складних згорток. Крім того, модульна структура дозволяє застосовувати Inception лише на вищих рівнях, залишаючи базові рівні стандартними згортковими шарами для економії ресурсів. Такий підхід забезпечує баланс між точністю та швидкістю, дозволяючи створювати моделі, які підходять для використання в умовах обмежених ресурсів.

Однією з найбільших переваг GoogLeNet є її здатність виконувати точне розпізнавання об’єктів у складних умовах, таких як слабке освітлення чи складний фон. Завдяки розподіленню обчислень між CPU та GPU, яке підтримується TensorFlow, модель оптимізує споживання енергії, залишаючись стабільною навіть у реальних умовах експлуатації. Це робить GoogLeNet цінним інструментом для автономних систем, таких як БПЛА, які використовуються для пошуково-рятувальних операцій або інших задач, що потребують високої точності розпізнавання в реальному часі.

Таким чином, GoogLeNet, побудована на базі TensorFlow, є прикладом гнучкої та продуктивної архітектури, яка підходить для широкого спектра задач комп’ютерного зору. Її здатність ефективно використовувати обмежені ресурси відкриває можливості для інтеграції в мобільні та автономні платформи, створюючи інструменти для складних та вимогливих умов експлуатації.

2.2 Проєктування апаратного забезпечення системи

Основними апаратними компонентами розроблюваної системи є одноплатний комп’ютер Raspberry Pi 3 Model B і камера Raspberry Pi Camera Module V2 (рис. 2.4).

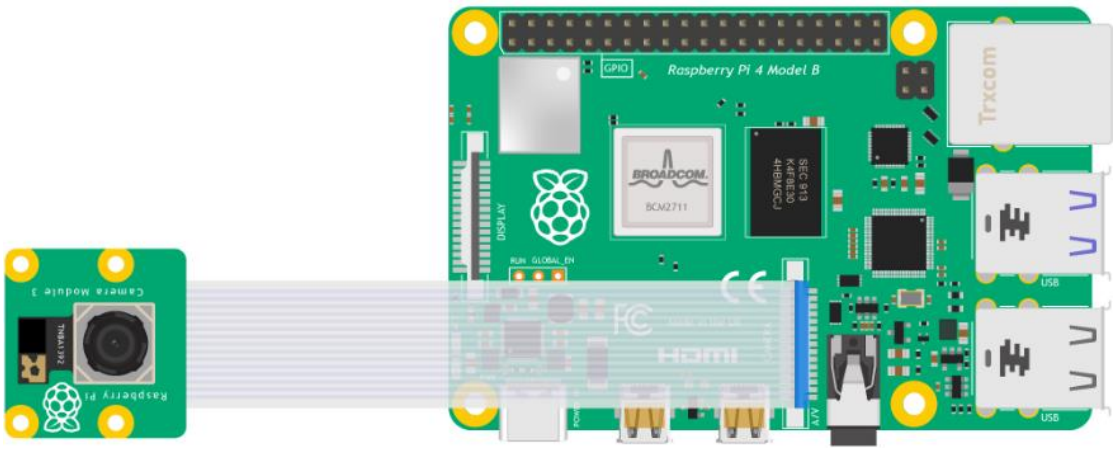


Рисунок 2.4 – Комп’ютер Raspberry Pi 3 Model B і
камера Raspberry Pi Camera Module V2

Мікрокомп’ютер Raspberry Pi 3 Model B був обраний як центральний елемент системи завдяки його портативності, енергоефективності та здатності обробляти зображення з належною обчислювальною потужністю. Пристрій підтримує використання бібліотеки TensorFlow, що дозволяє впроваджувати нейронні мережі, зокрема архітектури, такі як GoogLeNet, для задач категоризації, класифікації та розпізнавання об’єктів. Raspberry Pi 3 Model B оснащений 1 Гбайт оперативної пам’яті (ОП), що забезпечує достатню продуктивність для виконання завдань у режимі реального часу.

Наявність вбудованих модулів Wi-Fi та Ethernet дозволяє легко організувати передачу даних і здійснювати віддалений моніторинг роботи системи.

Камера Raspberry Pi Camera Module V2 також використовується в цій системі. Оснащена сенсором Sony IMX219 з роздільною здатністю 8 Мп, вона здатна записувати відео у форматі 1080p із частотою 30 кадрів на секунду. Камера підключається до порту CSI (Camera Serial Interface, укр. «послідовний інтерфейс камери») на Raspberry Pi 3 Model B, що забезпечує стабільну передачу відеоданих. Завдяки високій якості зображення цей модуль ідеально підходить для додатків комп’ютерного зору, які вимагають чіткого відео для подальшої обробки та розпізнавання об’єктів навіть у складних умовах освітлення.

Поєднання Raspberry Pi 3 Model B і модуля камери V2 формує апаратну основу системи, яка забезпечує стабільність і достатню обчислювальну потужність для розпізнавання об’єктів у реальному часі. Це є особливо важливим для застосувань у пошуково-рятувальних операціях, де надійність і точність відіграють ключову роль.

2.3 Використання власного набору даних для навчання моделі GoogLeNet InceptionNet

На рис. 2.5 нижче наведено набір даних, які були завантажені. Існує поки що три типи зображень, а саме: тварини, будинки та люди. В спільному DataSet приблизно 1500 зображень.

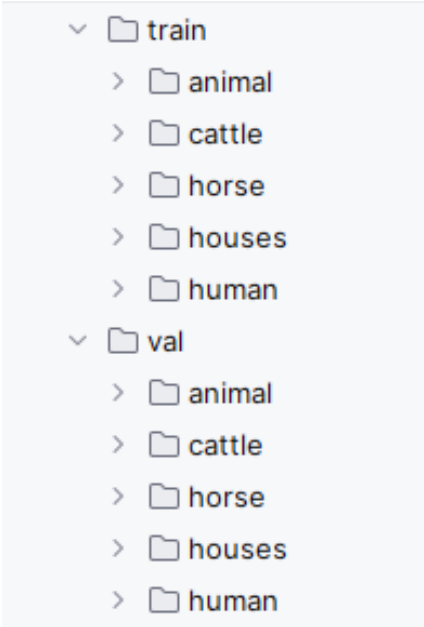


Рисунок 2.5 – Набір даних

Після завантаження набору даних зображення його потрібно розділити на набори даних для навчання та перевірки. Перший набір використовується для навчання даних моделі, а другий в основному використовується для перевірки моделі. Варто створити скрипт *create_labels_files.py*, який зможе безпосередньо генерувати текстові файли для навчання та тестування наборів даних (рис. 2.6). Для зручності розробки проєкту також було побудовано UML-діаграму активностей за допомогою плагіну PlantUML (рис. 2.7). Повний код винесено в додаток А1.

У цьому коді реалізовано дві функції для роботи з текстовими файлами та обробки файлів у директоріях. Функція *write_txt* записує дані у текстовий файл. Вона приймає три параметри: *content* (список даних), *filename* (ім’я файлу для запису) та *mode* (режим запису, за замовчуванням 'w'). Функція створює текстовий файл, де кожен рядок містить дані, розділені пробілами, а після останнього елемента додається символ нового рядка.

Функція *get_files_list* збирає список файлів із заданої директорії та її підкаталогів. Вона категоризує файли за папкою, в якій вони знаходяться (animal, houses, human), та додає відповідну мітку (*label*). Отриманий список містить шляхи до файлів та їхні мітки.

```python
import os
import os.path

def write_txt(content, filename, mode='w'):  2 usages  ubun
    with open(filename, mode) as f:
        for line in content:
            str_line = ""
            for col, data in enumerate(line):
                if not col == len(line) - 1:
                    # Розділяє значення пробілами
                    str_line = str_line + str(data) + " "
                else:

                    str_line = str_line + str(data) + "\n"
            f.write(str_line)

if __name__ == '__main__':
    train_dir = 'dataset/train'  # Тека з навчальними дани
    train_txt = 'dataset/train.txt'  # Шлях до файлу для з
    train_data = get_files_list(train_dir)  # Отримуємо сп
    write_txt(train_data, train_txt, mode='w')  # Зберігає

    val_dir = 'dataset/val'  # Тека з валідаційними даними
    val_txt = 'dataset/val.txt'  # Шлях до файлу для збере
    val_data = get_files_list(val_dir)  # Отримуємо список
    write_txt(val_data, val_txt, mode='w')  # Зберігаємо у
```

Рисунок 2.6 – Часткова структура *create_labels_files.py*

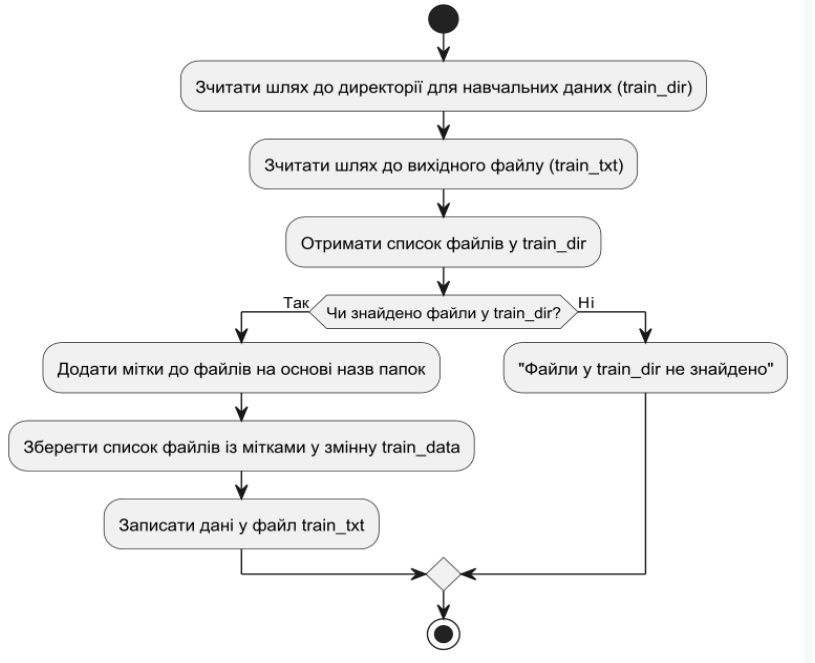


Рисунок 2.7 – UML-діаграма активностей скрипту *create_labels_files.py*

Діаграма активностей *create_labels_files.py* показує роботу з набором даних для тренування, скрипт також працює із набором для валідації і повторює всі ті кроки що і для попереднього.

У головному блоці програми функції використовуються для обробки файлів у теках train і val (навчальні та валідаційні дані). Списки файлів зберігаються у текстових файлах *train.txt* і *val.txt*.

Після чого залишається створити формат даних *.tfrecords*. Тому варто створити скрипт *create_tf_record.py* (рис 2.8), для якого також була побудована UML-діаграма активностей (рис. 2.9). Повний код був винесений в додаток А2.

```
def load_labels_file(filename, shuffle=False):  1 usage   ubuntu *
    """Завантажує список зображень і міток із текстового файлу."""
    images, labels = [], []
    with open(filename) as f:
        lines = f.readlines()
        if shuffle:
            random.shuffle(lines)
        for line in lines:
            path, label = line.strip().split(' ')
            images.append(path)
            labels.append(int(label))
    return images, labels


def read_image(filename, resize_height, resize_width, normalization=False):  2
    """Читає зображення, змінює розмір і, за потреби, нормалізує."""
    bgr_image = cv2.imread(filename)
    if bgr_image is None:
        raise ValueError(f"Cannot read image: {filename}")
    rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
    if resize_height > 0 and resize_width > 0:
        rgb_image = cv2.resize(rgb_image,  dsize: (resize_width, resize_height))
    if normalization:
        rgb_image = rgb_image / 255.0
    return np.asarray(rgb_image)
```

Рисунок 2.8 – Часткова структура *create_tf_record.py*

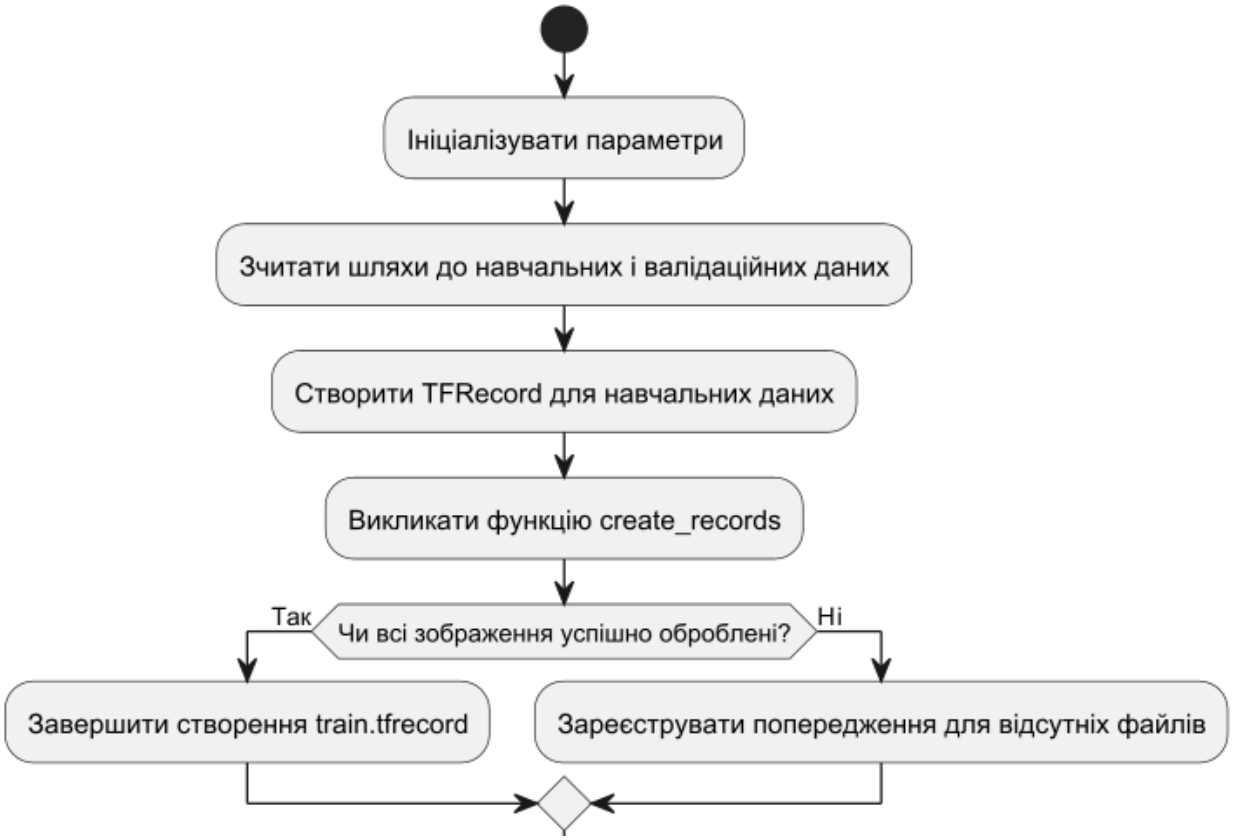


Рисунок 2.9 – UML-діаграма активностей скрипту *create_tf_record.py*

Діаграма активностей *create_tf_record.py* показує роботу з набором даних для тренування, скрипт також працює із набором для валідації і повторює всі ті кроки що і для попереднього

Перша допоміжна функція, *_int64_feature(value)*, створює об'єкт типу tf.train.Feature, який включає ціле число, що потрібно записати у TFRecord. Друга, *_bytes_feature(value)*, перетворює зображення у формат байтів для збереження.

Функція *load_labels_file()* завантажує шляхи до зображень і відповідні мітки з текстового файлу. Вона може перемішувати список, якщо параметр *shuffle* увімкнений.

Функція *read_image()* зчитує зображення з диска, конвертує його у формат RGB, змінює розмір і нормалізує значення пікселів, якщо це потрібно.

Функція *create_records()* створює файл TFRecord для зображень та міток. Вона зчитує шляхи до зображень і перетворює їх у формат, придатний для збереження у TFRecord.

Функція *read_records()* перетворює TFRecord у TensorFlow Dataset. Вона зчитує байтові дані з TFRecord, декодує їх, змінює розмір і нормалізує зображення за потреби.

Функція *disp_records()* показує кілька зображень з TFRecord для перевірки. Використовує matplotlib для візуалізації.

Функція *get_example_nums(tfrecord_file)* підраховує кількість зображень у TFRecord файлі, що корисно для оцінки обсягу даних перед тренуванням моделі.

Висновок до розділу 2

Програмна бібліотека з відкритим кодом для машинного навчання та ШІ TensorFlow була визнана найкращим вибором для реалізації проєкту через її здатність ефективно працювати з обмеженими ресурсами, що має вирішальне значення для дронів. TensorFlow Lite дозволяє оптимізувати моделі для вбудованих систем, зменшуючи енергоспоживання та покращуючи продуктивність, що важливо для автономних дронів, які виконують складні пошуково-рятувальні місії.

Архітектура глибинної згорткової нейромережі GoogLeNet на основі TensorFlow є ідеальним рішенням для розпізнавання об’єктів у реальному часі завдяки своїй здатності зменшувати витрати на обчислення, зберігаючи при цьому високу точність. Модуль Inception є частиною GoogLeNet і дозволяє ефективно обробляти дані та оптимізувати ресурси завдяки паралельним згортковим шарам різного розміру. Це забезпечує високу точність розпізнавання з обмеженою обчислювальною потужністю, що є ключовим аспектом для використання дронів у складних реальних середовищах.

З точки зору апаратного забезпечення, одноплатний мікрокомп’ютер Raspberry Pi 3 Model B і компактна камера Raspberry Pi Camera Module V2 є одним із найкращих та відносно дешевих варіантів для цієї системи. Raspberry Pi має достатню обчислювальну потужність для виконання завдань комп’ютерного зору та є енергоефективним, щоб подовжити час роботи дрона. Камера забезпечує високу якість зображення, яка необхідна для ефективної обробки відеоданих в режимі реального часу навіть у складних умовах освітлення, що дуже важливо для пошуково-рятувальних місій.

Не менш важливий є етап підготовки даних. Створення та збір наборів зображень для навчання моделі та конвертація їх у формат TFRecord забезпечує ефективну організацію та зберігання даних. Це дозволяє оптимізувати процес навчання, необхідний для досягнення високоточного розпізнавання об'єктів. Усі ці етапи разом дозволяють створити стабільну та енергоефективну автономну систему, яка ефективно працюватиме в умовах проведення пошуково-рятувальних операцій.

3. РЕАЛІЗАЦІЯ СИСТЕМИ КАТЕГОРИЗАЦІЇ ОБ’ЄКТІВ ДЛЯ ПОРЯТУНКУ НА БАЗІ RASPBERRY PI ТА TENSORFLOW LITE

У цьому розділі реалізовано систему категоризації об’єктів на апаратній платформі Raspberry Pi із ПЗ TensorFlow Lite, щоб забезпечити ефективну роботу в умовах обмежених ресурсів. Таким чином, можна миттєво розпізнавати та сортувати «збережені» об’єкти з високою точністю та високою швидкістю обробки.

На блок-схемі зображено алгоритм роботи системи для аналізу об’єктів у зоні дії (рис. 3.1).

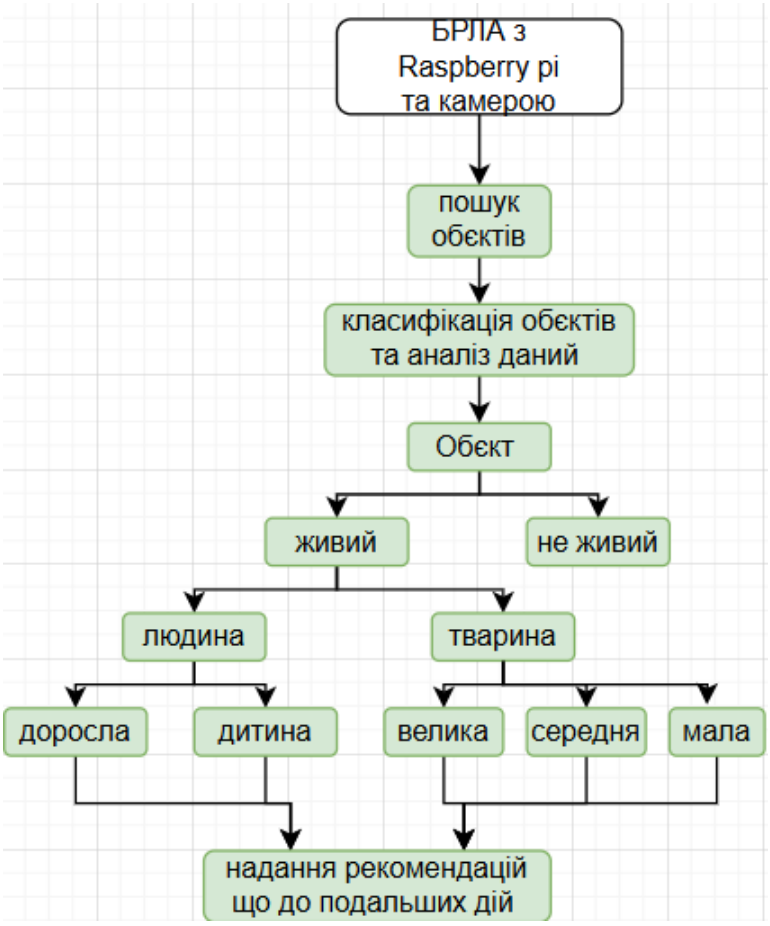


Рисунок 3.1 – Блок-схема алгоритму роботи

Спочатку БПЛА виконує пошук об'єктів за допомогою камери. Далі отримані дані аналізуються, і відбувається категоризація знайдених об’єктів та надання рекомендацій що до їх порятунку.

3.1 Опис апаратних рішень системи

Для реалізації системи розпізнавання об’єктів на базі згорткових нейронних мереж було обрано оптимізоване апаратне рішення, яке забезпечує необхідну обчислювальну потужність та ефективність енергоспоживання. Основним апаратним компонентом став одноплатний комп'ютер Raspberry Pi 3 Model B, який є ідеальним вибором для виконання задач машинного навчання в реальному часі. Цей пристрій оснащений чотириядерним процесором ARM Cortex-A53 та має 1 Гбайт оперативної пам'яті, що дозволяє ефективно обробляти зображення та виконувати обчислення для задач комп'ютерного зору. Для підключення до мережі та інших пристроїв використовуються вбудовані модулі Wi-Fi та Bluetooth.

Зображення для обробки постачаються камерами з високою роздільною здатністю, які підключені до Raspberry Pi. Камери забезпечують якісне зображення в реальному часі, а також можуть підтримувати інфрачервоні режими зйомки для роботи в умовах поганого освітлення. Зображення з камер передаються на пристрій для подальшої обробки. Для прискорення обчислень з використанням глибоких нейронних мереж можна додатково підключити GPU, що дозволяє суттєво зменшити час обробки великих наборів даних і збільшити ефективність системи.

Зберігання даних та моделей здійснюється за допомогою microSD-карти обсягом не менше 8 Гбайт. Для розширення пам'яті використовуються зовнішні носії, такі як USB-накопичувачі або жорсткі диски, підключені до Raspberry Pi. Щоб система могла працювати автономно протягом тривалого часу, використовуються літій-іонні акумулятори, що забезпечують необхідну ємність для безперебійної роботи. Додатково до цієї конфігурації можна підключити сенсори, такі як ультразвукові або інфрачервоні датчики, для покращення точності визначення об’єктів і умов навколишнього середовища.

Це апаратне рішення створює потужну і економічну платформу для виконання задач комп'ютерного зору в реальному часі, що є критично важливим для автономних систем, таких як безпілотні літальні апарати.

3.2 Файлова структура проєкту

Файлова структура проєкту виглядає зрозумілою та функціональною (рис. 3.2). Нижче буде детально описано кожну папку та файл, які використовуються в структурі проєкту.

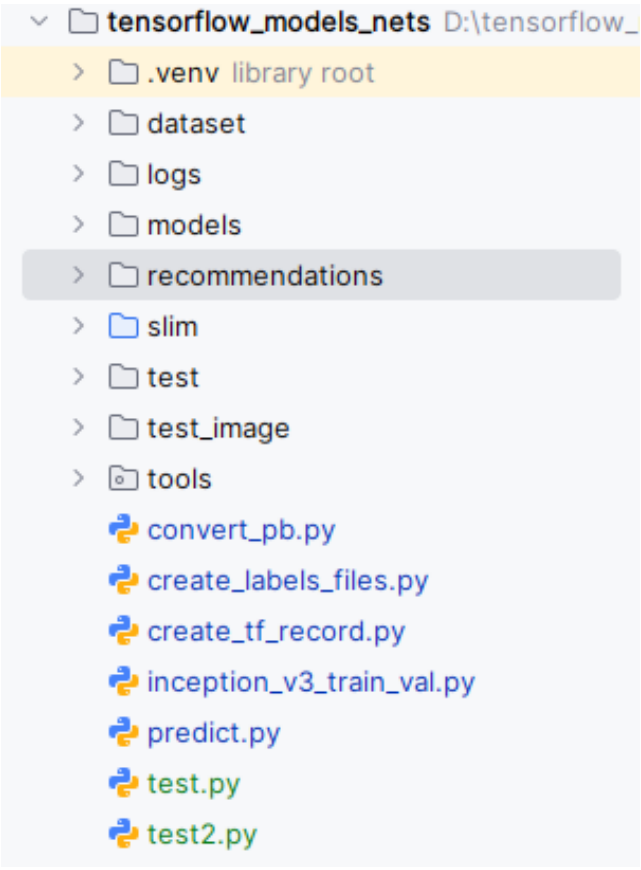


Рисунок 3.2 – Файлова структура

Опис директорій і файлів:

1. *dataset/* – Містить дані для тренування та валідації:

record/: Збережені у форматі TFRecord файли, які підходять для швидкої обробки TensorFlow.

train/: Оригінальні зображення для тренування.

val/: Оригінальні зображення для валідації.

2. *models/* – Директорія для збереження натренованих моделей:

Містить файли моделей, збережені під час тренування, наприклад, *.h5*, *.ckpt*, або *.keras*.

3. *slim/* – Копія Slim модуля з TensorFlow. Використовується для доступу до попередньо натренованих моделей і допоміжних функцій:

Завантажено з GitHub TensorFlow Slim.

4. *test_image*/ – Каталог для тестових зображень:

Використовується для прогнозування результатів або візуального тестування моделі.

Скрипти:

1. *create_labels_files.py* – Генерує *.txt* файли з мітками для тренувальних (*train*/) і валідаційних (*val*/) наборів даних (рис. 3.3).

```
animal\Abyssinian_1.jpg 0
animal\Abyssinian_10.jpg 0
animal\Abyssinian_100.jpg 0
animal\Abyssinian_101.jpg 0
animal\Abyssinian_102.jpg 0
animal\Abyssinian_103.jpg 0
animal\Abyssinian_104.jpg 0
```

Рисунок 3.3 – Структура *.txt*-файлів

2. *create_tf_record.py* – Конвертує вихідні зображення та мітки у формат TFRecord, що є ефективним для використання в TensorFlow.

3. *inception_v3_train_val.py* – Скрипт для тренування та валідації моделі Inception V3. Аналогічний до inception_v1_train_val.py, але використовує архітектуру Inception V3.

4. *predict.py* –Реалізує логіку прогнозування:

– Завантаження натренованої моделі.

– Передбачення результатів для нових зображень з папки test_image/.

3.3 Навчання моделі GoogLeNet InceptionNet V3

Ймовірно, краще мати єдиний навчальний розмір введення, наприклад, 299 × 299 і забути про 224 × 224, який цілком підтримується, але радше рекомендується для попереднього тестування або простих експериментів.

У розроблюваному проєкті використовується скрипт *create_tf_record.py* для підготовки даних у форматі TFRecord. Для розуміння побудови скрипту тренування будо побудовано UML-діаграму активностей (рис. 3.4). Таким чином створюються два файли *train299.tfrecords* і *val299.tfrecords* з даними для наборів навчання і перевірки відповідно (рис. 3.5).

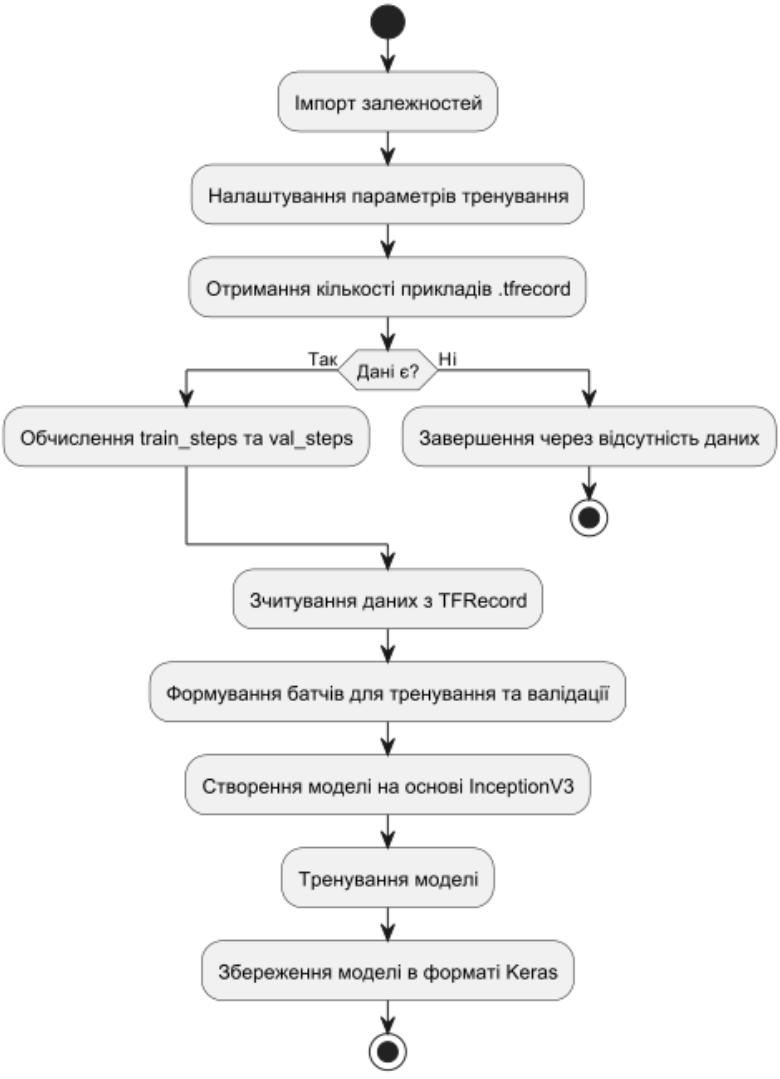


Рисунок 3.4 – UML-діаграма активностей скрипту тренування моделі

```python
from create_tf_record import *  # Імпортуємо функції
import numpy as n2

# === Налаштування ===
resize_height = 229
resize_width = 229
batch_size = 32
epochs = 10

# Шляхи до TFRecord-файлів
train_record_file = "dataset/record/train.tfrecords"
val_record_file = "dataset/record/val.tfrecords"
```

Рисунок 3.5 – Імпорт та налаштування

Розроблюваний проєкт використовує скрипт *create_tf_record.py* для підготовки даних у форматі TFRecord. Таким чином створюються два файли *train299.tfrecords* і *val299.tfrecords* з даними для наборів навчання і перевірки відповідно.

Наступний фрагмент коду – функція *train*, яка є важливою для навчання моделі. Він також має низку вхідних даних, як-от шляхи до файлів TFRecord для навчання, а також перевірки даних, а також параметри обробки зображення, такі як розміри, розмір партії та кількість епох.

Спочатку відображається версія TensorFlow, щоб переконатися, що налаштування середовища є правильними, а потім перевіряється доступність даних, прочитавши файли TFRecord за допомогою функції get_example_nums, яка показує кількість зразків у кожному файлі (рис. 3.6). Якщо кількість зразків дорівнює нулю для будь-якого файлу (навчання перевірки), виводиться інформаційна помилка, яка належним чином попереджає про порожні файли та дозволяє уникнути подальші проблеми під час навчання (див. рис. 3.6).

```python
def train(train_record_file, val_record_file, resize_height, resize_width, batch_size, epochs):
    print(f"Tensorflow version:{tf.__version__}")

    # Перевірка кількості прикладів
    train_nums = get_example_nums(train_record_file)
    val_nums = get_example_nums(val_record_file)
    print(f"Train nums: {train_nums}, Val nums: {val_nums}")

    if train_nums == 0 or val_nums == 0:
        raise ValueError("Train or validation dataset is empty. Check your TFRecord files.")
```

Рисунок 3.6 – Перевірка даних з файлів TFRecord

Дані зчитуються з файлів TFRecord і готуються для того, щоб їх можна було надати для навчання; функції базуються на функціях бібліотеки TensorFlow, що робить завантаження та попередню обробку даних максимально оптимізованими.

Перш за все, у функції *read_records* зчитуються набори даних навчання та перевірки з файлів, які зберігаються у *train_record_file* та *val_record_file* відповідно. Він шукає розміри зображення після цього (*resize_height* і *resize_width*) і визначає, який тип нормалізації також потрібно виконати. У цьому випадку нормалізовано значення пікселів зображення.

Далі буде створено та змонтовано шари. Наприклад, навчальний набір використовує метод *shuffle*, який перемішує та передає параметр «1000» для перемішування даних за допомогою буфера розміром 1000 елементів, щоб дані можна було випадково перетасувати під час навчання. Нарешті, дані упаковуються в пакети розміру *batch_size* за допомогою методу *batch* і, крім того, з попередньо вибраними пакетами за допомогою методу попередньої вибору, щоб вони працювали шляхом попереднього завантаження пакета, що його обробляє модель, яка в основному корисно значно прискорити процес навчання, не створюючи затримки при завантаженні даних.

Для валідаційного набору також створюється батч, але без перемішування, оскільки для валідації важливо використовувати дані в їх початковому порядку (рис. 3.7).

```
train_steps = int(np.ceil(train_nums / batch_size))
val_steps = int(np.ceil(val_nums / batch_size))

# Зчитування даних
train_dataset = read_records(train_record_file, resize_height, resize_width, normalization_type='normalization')
val_dataset = read_records(val_record_file, resize_height, resize_width, normalization_type='normalization')

# Повторення даних і формування батчів
train_dataset = train_dataset.repeat().shuffle(buffer_size=1000).batch(batch_size).prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE)
val_dataset = val_dataset.repeat().batch(batch_size).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

Рисунок 3.7 – Формування батчів

Далі відбувається побудова моделі на основі InceptionV3 для задачі категоризації зображень. Спочатку створюється базова модель InceptionV3, яка завантажує попередньо навчені ваги з набору даних ImageNet; при цьому верхній шар моделі (класифікатор) не використовується, оскільки буде доданий власний класифікатор. Для цього вказується параметр *include_top=False*. Також встановлюється *input_shape=(resize_height, resize_width, 3)*, що визначає розмір вхідних зображень після зміни їх розміру, а параметр *weights='imagenet'* забезпечує використання попередньо навчених ваг. Для збереження властивостей попереднього навчання базової моделі всі її шари заморожуються, встановлюючи *base_model.trainable = False*.

Після цього додається класифікатор. Спочатку використовується *GlobalAveragePooling2D()*, який зменшує просторові розміри зображення до одного значення для кожного каналу, а потім додаються кілька повнозв'язних шарів. Перший з них – *Dense(1024, activation='relu')* – має 1024 нейрони і функцію активації *ReLU* для додавання нелінійності. Наступний шар — це *Dropout(0.5*), який допомагає запобігти перенавчанню, випадково вимикаючи 50 % нейронів. Останнім шаром є *Dense(5, activation='softmax')*, що має 10 виходів для 10 класів і використовує функцію активації *softmax*, яка перетворює виходи в ймовірності для кожного класу.

Модель компілюється з оптимізатором *Adam* і функцією втрат *sparse_categorical_crossentropy*, яка підходить для задач багатокласової

класифікації з цілими мітками класів (рис. 3.8). Як метрику використовуємо точність *(accuracy)*.

```
base_model = tf.keras.applications.InceptionV3(
    input_shape=(resize_height, resize_width, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False  # Заморожуємо основну модель

# Додаємо класифікатор
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')  # 10 кл
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Рисунок 3.8 – Побудова моделі InceptionV3

Далі здійснюється тренування моделі. За допомогою методу *fit()* модель навчається на тренувальному наборі даних, який подається через змінну *train_dataset*. Кількість епох для тренування вказується через параметр *epochs=epochs*, що визначає кількість разів, коли модель буде проходити через усі дані. Параметр *validation_data=val_dataset* дозволяє використовувати валідаційний набір для перевірки точності моделі після кожної епохи, що допомагає контролювати процес перенавчання.

Після завершення тренування модель зберігається за допомогою методу *save()*. Збережена модель зберігається у вказаній директорії *models/inception_v3/* під назвою *inception_v3_model.keras*, що дозволяє у подальшому завантажити її для подальших тестів або використання. Після цього виводиться повідомлення про успішне збереження моделі (рис. 3.9–3.10).

```
history = model.fit(
    train_dataset,
    epochs=epochs,
    steps_per_epoch=train_steps,
    validation_data=val_dataset,
    validation_steps=val_steps
)

# Збереження моделі
model.save("models/inception_v3/inception_v3_model.keras")
print("Model saved to inception_v3_model.keras")
```

Рисунок 3.9 – Тренування моделі та збереження моделі

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 182s 5s/step - accuracy: 0.7809 - loss: 0.8246 - val_accuracy: 0.8534 - val_loss: 0.3699
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 130s 4s/step - accuracy: 0.9960 - loss: 0.0118 - val_accuracy: 0.9799 - val_loss: 0.0634
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 1.0000 - loss: 2.2612e-04 - val_accuracy: 0.8858 - val_loss: 0.2076
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 130s 4s/step - accuracy: 0.9974 - loss: 0.0034 - val_accuracy: 0.7639 - val_loss: 0.6921
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 128s 4s/step - accuracy: 0.9980 - loss: 0.0078 - val_accuracy: 0.9660 - val_loss: 0.0986
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 130s 4s/step - accuracy: 0.9997 - loss: 7.5205e-04 - val_accuracy: 0.9120 - val_loss: 0.1724
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 128s 4s/step - accuracy: 1.0000 - loss: 2.8039e-04 - val_accuracy: 0.8704 - val_loss: 0.2259
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 1.0000 - loss: 5.9905e-04 - val_accuracy: 0.8410 - val_loss: 0.3002
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 127s 4s/step - accuracy: 1.0000 - loss: 2.5336e-05 - val_accuracy: 0.8349 - val_loss: 0.3147
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 129s 4s/step - accuracy: 1.0000 - loss: 5.2212e-04 - val_accuracy: 0.9907 - val_loss: 0.0385
Model saved to inception_v3_model.keras

Process finished with exit code 0
```

Рисунок 3.10 – Тренування моделі

Згідно з результатами тренування, можна зробити такі висновки:

Модель демонструє значний прогрес у тренуванні: початкова точність на тренувальному наборі становила 78 %, а до останньої епохи вона досягла 100 %. Це свідчить, що модель повністю адаптувалася до тренувальних даних.

Валідаційна точність стартувала з 85 % у першій епосі й досягла 99,07 % в останній. Це свідчить про відмінне узгодження моделі з тестовими даними, але також є ризик, що модель може бути перенавченою (особливо з урахуванням точності 100 % на тренувальних даних).

Значення втрат (loss) на тренуванні зменшилося майже до нуля, що відповідає високій точності. Однак, втрати на валідаційному наборі (val_loss) мали незначні коливання, досягнувши найнижчого значення (0,0385) до кінця тренування.

Незважаючи на коливання валідаційної точності в середині тренування, вона знову стабілізувалася й досягла максимального значення. Це свідчить про здатність моделі правильно категоризувати дані, не втрачаючи узагальнюючих властивостей.

Модель була успішно збережена, що дозволяє її використовувати для прогнозування на нових даних.

Повний код винесений в додаток А3.

3.4 Імпорт моделі GoogLenet InceptionNet V3 на Raspberry Pi 3 Model B

Модель GoogLeNet InceptionNet V3 – це глибока нейронна мережа, розроблена спеціально для класифікації зображень і заснована на архітектурі Inception, яка забезпечує високу ефективність аналізу складних даних. Цей розділ містить процес імпортування вже навченої моделі InceptionNet V3 до одноплатного комп’ютера Raspberry Pi 3 Model B для подальшого використання в категоризації зображень.

Початок роботи з моделлю вимагає налаштувань Raspberry Pi 3 Model B. Потрібно встановити Python 3.11+ з усіма потрібними бібліотеками. Модель у форматі TensorFlow Lite (TFLite) необхідно попередньо підготувати та зберегти, щоб вона ефективно працювала на обмежених апаратних ресурсах.

Код на рис. 3.11 виконує конвертацію моделі з формату Keras у TensorFlow Lite. Функція *tf.keras.models.load_model* завантажує модель із формату Keras. Далі використовується *tf.lite.TFLiteConverter.from_keras_model*, щоб створити конвертер для перетворення моделі у формат TFLite. За допомогою *converter.convert()* модель конвертується, а збереження результату у файл *model.tflite* здійснюється через контекстний менеджер *with open()*. В кінці виводиться повідомлення про успішну конвертацію.

```
import tensorflow as tf

# Завантажуємо збережену модель
model = tf.keras.models.load_model("models/inception_v3/inception_v3_model.keras")

# Конвертуємо модель у TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Зберігаємо TFLite-модель
with open("models/inception_v3/model.tflite", "wb") as f:
    f.write(tflite_model)

print("Модель конвертовано у формат TFLite.")
```

Рисунок 3.11 – Конвертація моделі *keras* в модель *tflite*.

Наступний скрип буде відразу в себе включати декілька частин. Він буде працювати з камерою та водночас категоризувати об’єкти, які потрапляють в об’єктив.

У першій частині коду є налаштування параметрів для доступу до GitHub репозиторію, куди буде надсилатись інформація щодо об’єктів та рекомендації до їх порятунку (рис. 3.12). Змінні *GITHUB_TOKEN* і *GITHUB_REPO* використовуються для аутентифікації та вказівки, в який репозиторій завантажувати файли.

```
GITHUB_TOKEN = "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"
GITHUB_REPO = "gfdtkl23/tensorflow_v2_models_nets"
```

Рисунок 3.12 – Налаштування GitHub

Токен використовується для аутентифікації запитів до GitHub API, а ім'я репозиторію вказує, в який саме репозиторій будуть завантажуватися файли. Модель TensorFlow Lite завантажується з вказаного шляху(рис. 3.13).

```
MODEL_PATH = r"model.tflite"
if not os.path.exists(MODEL_PATH):
    raise FileNotFoundError(f"Файл моделі {MODEL_PATH} не знайдено!")

interpreter = tf.lite.Interpreter(model_path=MODEL_PATH)
interpreter.allocate_tensors()
print("Модель успішно завантажена (TensorFlow Lite)!")
```

Рисунок 3.13 – Завантаження моделі TensorFlow Lite

Якщо файл не знайдено, виникає помилка. Після цього створюється *interpreter*, що дозволяє працювати з моделлю TensorFlow Lite та здійснювати обчислення. Виклик *allocate_tensors()* виконує необхідні операції для підготовки інтерпретатора до роботи.

Ця функція дозволяє завантажити файл на GitHub. Файл кодується в *base64* перед тим, як бути надісланим на сервер (рис. 3.14). Відповідно до статусу відповіді від сервера, виводиться повідомлення про успіх чи помилку.

```python
def upload_file_to_github(file_path, github_path):
    url = f"https://api.github.com/repos/{GITHUB_REPO}/contents/{github_path}"
    with open(file_path, "rb") as file:
        encoded_content = base64.b64encode(file.read()).decode("utf-8")
    data = {
        "message": f"Завантажено: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}",
        "content": encoded_content,
    }
    headers = {"Authorization": f"Bearer {GITHUB_TOKEN}"}
    response = requests.put(url, json=data, headers=headers)
    if response.status_code == 201:
        print(f"Файл {file_path} успішно завантажено як {github_path}.")
    else:
        print(f"Помилка при завантаженні {file_path}: {response.status_code} - {response.text}")
```

Рисунок 3.14 – Завантаження файлів на GitHub

В основному циклі зчитується кадр з камери, обробляється і передається в модель для прогнозування (рис. 3.15).

```python
while True:
    ret, frame = cap.read()
    if not ret:
        print("Помилка зчитування кадру з камери.")
        break
    input_frame = preprocess_frame(frame)
    interpreter.set_tensor(input_details[0]['index'], input_frame.astype(np.float32))
    interpreter.invoke()
    predictions = interpreter.get_tensor(output_details[0]['index'])
    predicted_class = np.argmax(predictions, axis=1)[0]
    class_probability = predictions[0][predicted_class] * 100  # Відсоток для передбаченого класу
    label = class_labels[predicted_class]
    cv2.putText(frame, f"Class: {label} ({class_probability:.2f}%)", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (0, 255, 0), 2)
    cv2.imshow("Розпізнавання", frame)
    if class_probability >= 75.0:
        image_name = f"{label}_{class_probability:.2f}.jpg"
        cv2.imwrite(image_name, frame)
        print(f"Збережено зображення: {image_name}")

        folder_path = create_recommendation_folder(predicted_class, image_name)
        upload_folder_to_github(folder_path)
        print("Рекомендації завантажено. Завершення аналізу.")
        break
    # Вихід із програми (при натисканні 'q')
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Рисунок 3.15 – Основний цикл

Після цього результат передбачення виводиться на екран. Якщо ймовірність передбаченого класу перевищує 75 %, то програма зберігає зображення та виконує подальші кроки для завантаження файлів. Повний код винесено в додаток А 4.

Висновок до розділу 3

Проєкт на основі TensorFlow з використанням архітектури InceptionV3 успішно реалізує систему для розпізнавання людей і тварин на Raspberry Pi. Підготовка даних у форматі TFRecord та застосування технік попередньої обробки, таких як зміна розміру зображень, забезпечили ефективне навчання моделі навіть на обмежених ресурсах.

Результати тренування показали високу точність категоризації, досягнувши майже 100 % на тренувальних даних і понад 99 % на валідаційних. Це вказує на ефективність застосованої архітектури для задач розпізнавання, де важлива точність навіть за складних умов, таких як низька якість зображень або різноманіття об'єктів.

Конвертування моделі в формат TensorFlow Lite дозволило адаптувати її для роботи в умовах обмежених ресурсів Raspberry Pi, що є важливим кроком для практичного застосування в реальних сценаріях. Після конвертації модель успішно інтегрувалась у систему, де вона використовує камери для реального часу розпізнавання та відправлення даних на сервер для подальшої обробки.

Це дозволяє створити автономну систему, здатну виконувати розпізнавання і надавати рекомендації щодо порятунку людей і тварин, що є важливим для різноманітних застосувань, включаючи гуманітарні місії та екологічні проєкти.

4. ТЕСТУВАННЯ АПАРАТНО-ПРОГРАМНОЇ СИСТЕМИ ДЛЯ МІСІЙ ПОШУКУ ТА ПОРЯТУНКУ З ВИКОРИСТАННЯМ БПЛА

Апаратні та програмні системи є критично важливими для пошуково-рятувальних операцій у сучасних операціях, особливо у важкодоступних або небезпечних для життя районах. Ймовірно, це використання БПЛА, здатних виконувати швидкі та ефективні пошукові місії на великих територіях. Але, на жаль, на сьогоднішній день існують певні обмеження щодо їх реального використання, особливо в реальних умовах.

Тому в поточному вигляді система не використовує БПЛА в пошукових місіях; натомість він зосереджується на застосуванні звичайних методів обробки зображень із та категоризує об’єкти для виконання поставлених завдань у межах пошуково-рятувальних операцій. Це дозволить отримати точні показання або ідентифікувати потенційний об’єкт, наприклад людину чи тварину, на знімках, зроблених камерою.

4.1 Прогнозування моделі Inception V3

У рамках тестування моделі з було відібрано 9 випадкових зображень: по 3 фотографії для кожного класу. Прогнозування дозволить оцінити ефективність моделі в реальних умовах і перевірити її здатність правильно категоризувати тестові об'єкти.

Для тестування було реалізовано скрипт *predict.py*, функції якого наведені на скріншотах нижче.

Функція *read_image* служить для читання та передобробки зображень (рис. 4.1). Вона використовує: *cv2.imread(image_path)* для зчитуваня зображення з вказаного шляху, *cv2.cvtColor(img, cv2.COLOR_BGR2RGB)* для конвертуваня зображення з формату BGR (який використовує OpenCV) у RGB (який використовується в TensorFlow), *cv2.resize(img, (resize_width, resize_height))* для зміни розмір зображення до заданих норм, та *img / 255.0* для нормалізації пікселів зображення до діапазону [0; 1].

```python
def read_image(image_path, resize_height, resize_width, normalization=True):
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Cannot read image: {image_path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (resize_width, resize_height))
    if normalization:
        img = img / 255.0
    return img
```

Рисунок 4.1 – Функція для читання та передобробки зображень

Функція *load_labels* завантажує етикетки класів із текстового файлу, розділеного табуляцією. Кожен елемент файлу представляє назву класу. Якщо файл не існує або виникає помилка під час читання, викидається повідомлення про помилку (рис. 4.2).

```python
def load_labels(labels_filename):
    if not os.path.exists(labels_filename):
        raise FileNotFoundError(f"Labels file not found: {labels_filename}")
    try:
        labels = np.loadtxt(labels_filename, dtype=str, delimiter='\t')
    except Exception as e:
        raise ValueError(f"Error reading labels file: {e}")
    return labels
```

Рисунок 4.2 – Функція для завантаження *labels*-файлу

Функція *predict* завантаження моделі TFLite, якщо модель не знайдена, викидає помилку; якщо модель знайдена, її інтерпретатор ініціалізується для виконання прогнозів (рис. 4.3). Після цього виконується перевірка розмірності виходу моделі, перевіряється, чи кількість класів, яку прогнозує модель, відповідає кількості класів, заданих у параметрі. Далі функція отримує списки зображень для прогнозування, шукає зображення у вказаній директорії.

```
predict(model_path, image_dir, labels_filename, labels_nums, resize_height, resize_width)
labels = load_labels(labels_filename)
if not os.path.exists(model_path):
    raise FileNotFoundError(f"TFLite model file not found: {model_path}")
interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
if output_details[0]['shape'][1] != labels_nums:
    raise ValueError(f"Model output classes ({output_details[0]['shape'][1]}) "
                     f"do not match the provided label numbers ({labels_nums}).")
images_list = glob.glob(os.path.join(image_dir, '*.jpg'))
if not images_list:
    print(f"No images found in directory: {image_dir}")
    return
for image_path in images_list:
    img = read_image(image_path, resize_height, resize_width)
    img = np.expand_dims(img, axis=0).astype(np.float32)
    start_time = time.time()
    interpreter.set_tensor(input_details[0]['index'], img)
    interpreter.invoke()
    pre_score = interpreter.get_tensor(output_details[0]['index'])[0]
    pre_label = np.argmax(pre_score)
    max_score = pre_score[pre_label]
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"{image_path} is: pre labels: {pre_label}, name: {labels[pre_label]}, score: {
    print(f"Time taken for prediction: {elapsed_time:.4f} seconds ({elapsed_time_ms} ms)"
```

Рисунок 4.3 – Функція для прогнозування моделі

Потім виконується прогнозування для кожного зображення та виводиться клас з найвищою ймовірністю та час виконання прогнозу (рис. 4.4). Повний код прогнозування було винесено в додаток А 5.

```
File  Edit  Tabs  Help
csbot@raspberrypi:~ $ source tftest1/bin/activate
(tftest1) csbot@raspberrypi:~ $ cd tftest1/inception/
(tftest1) csbot@raspberrypi:~/tftest1/inception $ python3 predict.py
Error in cpuinfo: prctl(PR_SVE_GET_VL) failed
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
test_image/images7.jpg is: pre labels: 1, name: houses, score: 1.0000
Time taken for prediction: 1.1729 seconds (1172.853 ms)
test_image/images8.jpg is: pre labels: 0, name: animal, score: 1.0000
Time taken for prediction: 1.2028 seconds (1202.791 ms)
test_image/images4.jpg is: pre labels: 1, name: houses, score: 1.0000
Time taken for prediction: 1.1663 seconds (1166.294 ms)
test_image/images1.jpg is: pre labels: 1, name: houses, score: 1.0000
Time taken for prediction: 1.1692 seconds (1169.159 ms)
test_image/images5.jpg is: pre labels: 0, name: animal, score: 1.0000
Time taken for prediction: 1.1685 seconds (1168.469 ms)
test_image/images9.jpg is: pre labels: 2, name: human, score: 0.9943
Time taken for prediction: 1.1668 seconds (1166.785 ms)
test_image/images2.jpg is: pre labels: 0, name: animal, score: 1.0000
Time taken for prediction: 1.1655 seconds (1165.497 ms)
test_image/images3.jpg is: pre labels: 2, name: human, score: 0.9990
Time taken for prediction: 1.1679 seconds (1167.935 ms)
test_image/images6.jpg is: pre labels: 2, name: human, score: 1.0000
Time taken for prediction: 1.2095 seconds (1209.547 ms)
```

Рисунок 4.4 – Результати прогнозування моделі на Raspberry Pi

Прогнозування моделі на зображеннях показує категорії для кожного фото, включаючи ймовірності для кожної категорії, що вказують на впевненість моделі у своєму прогнозі. Наприклад, якщо модель категоризує зображення як "houses" і дає ймовірність 1,0000 – це означає, що вона дуже впевнена у своєму прогнозі. В деяких випадках ймовірність знижується, наприклад, при категоризації "human", що може бути пов'язано з тим, що на фото є об'єкти, які знаходяться на великій відстані або якість зображення занизька, через що модель не так впевнена в прогнозі. Час, витрачений на прогнозування, варіюється для кожного зображення, що може залежати від розміру зображення або складності сцени. В цілому, модель добре працює з об'єктами, що знаходяться ближче до камери, але може мати нижчу впевненість, коли об'єкти розташовані на великій відстані або є менш чіткими.

Час, за який на Raspberry Pi розпізнається об’єкт, коливається від 1,16 с до 1,2 с (рис. 4.5).

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
test_image\images1.jpg is: pre labels: 1, name: houses, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
test_image\images2.jpg is: pre labels: 0, name: animal, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
test_image\images3.jpg is: pre labels: 2, name: human, score: 0.9990
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step
test_image\images4.jpg is: pre labels: 1, name: houses, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
test_image\images5.jpg is: pre labels: 0, name: animal, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
test_image\images6.jpg is: pre labels: 2, name: human, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step
test_image\images7.jpg is: pre labels: 1, name: houses, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
test_image\images8.jpg is: pre labels: 0, name: animal, score: 1.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
test_image\images9.jpg is: pre labels: 2, name: human, score: 0.9943
```

Рисунок 4.5 – Результати прогнозування моделі на ноутбуці з процесором i5-7200U та обсягом ОП 20 Гбайт

Прогнозування на ПК для кожного зображення відбувалось за дуже короткий час, з переважною більшістю прогнозів, виконаних за 100–125 мс. Для кожного зображення модель виводила ймовірність належності до відповідних категорій. З 10 об’єктів модель розпізнала 8 з ймовірністю в 100 %, два інших – з ймовірністю в 99 %. Ці результати демонструють ефективність моделі для реального часу в рамках обробки зображень.

Цей процес вказує на можливість використання моделі в реальних умовах з високою швидкістю та точністю для визначення об'єктів на зображеннях.

4.2 Порівняння моделі з іншими схожими дослідженнями

Для порівняння побудованої моделі з дослідженнями «Object Detection Models on Raspberry Pi» та «A Deep Trash Classification Model on Raspberry Pi 4» було створено таблицю з ключовими метриками: точністю, часом прогнозування, кількістю епох та використаними датасетами [20–21].

Таблиця 4.1 – Порівняння з іншими схожими моделями

Параметр	**TensorFlow Lite на InceptionV3**	**Object Detection Models on Raspberry Pi**	**A Deep Trash Classification Model on Raspberry Pi 4**
Модель	TensorFlow Lite на InceptionV3	SSDLite MobileNetV2, YOLOv3-tiny	ResNet-50
Пристрій	Raspberry Pi 3 Model B	Raspberry Pi 3 Model B+	Raspberry Pi 4
Датасет	The Oxford-IIIT Pet Dataset, Houses Dataset, Human Detection Dataset	COCO Dataset	Власний
Вхідна форма	229 × 229	224 × 224	224 × 224
Точність	~99 %	~82 % (YOLOv3-tiny), ~87 % (SSDLite MobileNetV2)	~96 %

Параметр	TensorFlow Lite на InceptionV3	Object Detection Models on Raspberry Pi	A Deep Trash Classification Model on Raspberry Pi 4
Час прогнозування (1 зобр.)	~1,20 с	~0,9 с (YOLOv3-tiny), ~0,6 с (SSDLite MobileNetV2)	Не вказано
Кількість епох	10	Не вказано	5

Моделі на основі архітектури TensorFlow Lite та InceptionV3 показали найвищу точність (приблизно 99 %) серед запропонованих рішень, що підкреслює її ефективність для категоризації різних типів об’єктів. Час передбачення на одне зображення становить близько 1,20 секунди, що повільніше порівняно з іншими моделями, такими як SSDLite MobileNetV2 (приблизно 0,6 секунди) і YOLOv3-tiny (приблизно 0,9 секунди).

Набір даних, використаний у дослідженні, охоплює кілька класів об’єктів, що демонструє широку універсальність завдань, що вирішуються. Це різко контрастує з вузькоспеціалізованими підходами інших моделей, таких як моделі глибокої класифікації «A Deep Trash Classification», які зосереджуються на обмежених наборах даних.

Кількість епох навчання в TensorFlow Lite становить 10, що більше порівняно з 5 епохами в моделі класифікації, що може вплинути на підвищення точності, але збільшить час навчання. Недоліком є те, що Raspberry Pi 3 Model B потребує трохи більше ресурсів, що робить цю модель неідеальною для обладнання з обмеженими ресурсами.

4.3 Тестування в реальному часі

У цьому підрозділі буде розглянуто процес обробки даних з камери у реальному часі для виконання прогнозів за допомогою моделі машинного навчання. Зображення, отримані з камери, будуть передаватися на пристрій для обробки, після чого результати прогнозування надсилатимуться для подальшої обробки чи збереження, наприклад, на GitHub або іншому сховищі. Це дозволить інтегрувати систему в реальний процес роботи та забезпечити автоматичний збір результатів для аналізу та моніторингу.

Під час роботи модель в реальному часі визначає категорії об’єкту, якщо співпадіння з потрібними категоріями більше 75 % створюється тека в назві якої заначено дату та час коли об’єкт був знайдений та до якого класу він відноситься. В теку зберігається фото об’єкту та файл з рекомендаціями щодо порятунку, приблизний зміст рекомендацій нижче. Фото при збереженні отримує назву категорії до якої було віднесено об’єкт та відсоток співпадіння (рис. 4.6).

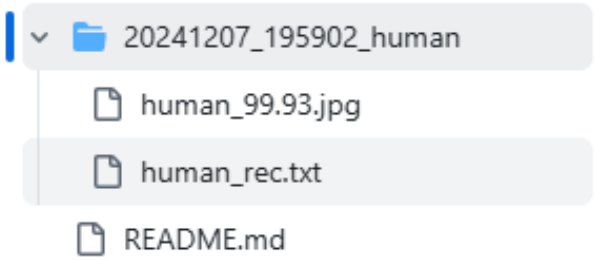


Рисунок 4.6 – Збережена тека на репозиторію

Перелік видів об’єктів, для яких проводиться категоризація:

1) тварина:

– якщо поранену тварину виявлено в місці евакуації (наприклад, у лісовій зоні чи покинутому будинку), для транспортування слід використовувати спеціальні носилки (якщо тварина велика або поранена). Для дрібних тварин доступні переноски;

– до надання першої допомоги та консультування щодо подальших дій залучаються ветеринари;

– якщо поранену тварину необхідно транспортувати через обмежений простір (наприклад, через сміття), слід використовувати невелике обладнання або переносний транспорт. При наявності кількох тварин необхідно організувати мобільні групи;

2) людина:

– на випадок поранення або задухи повинні бути доступні джгути, пов’язки та методи відновлення прохідності дихальних шляхів (наприклад, ендотрахеальна інтубація) [23].Слід забезпечити належну організацію надання першої допомоги;

– за потреби залучаються машини швидкої допомоги, а у разі серйозних травм – вертольоти для швидкої транспортування до медичних закладів;

– важливі також знеболюючі, антибіотики, кровоспинна транексамова кислота, вітаміни або препарати, що підвищують опірність організму. У разі масової евакуації необхідно буде організувати систему збору та розподілу рідин і крові;

– для оцінки тяжкості травми та надання подальшої допомоги рекомендовано мобільні медичні пункти;

3) будинок :

– якщо постраждалий знаходиться в покинутому будинку або іншій будівлі, важливо оцінити руйнування конструкції або пошкодження, які можуть ускладнити евакуацію;

– необхідно організувати пошуково-рятувальні групи для проведення ретельного обстеження всіх місць;

– якщо будівля має багато поверхів або вузькі проходи, використовуйте легкі носилки для транспортування потерпілих, а також використовуйте спеціальні інструменти, щоб пробити стіни, двері та вікна, щоб забезпечити безпечний вихід;

– якщо кількість постраждалих велике, необхідно направити додаткові медичні бригади та задіяти мобільні госпіталі, які можна розгорнути поблизу або віддалено;

– якщо наземний транспорт недоступний, для транспортування важкопоранених можна використовувати вертоліт. Якщо це неможливо, можна використовувати спеціальне обладнання, яке може проходити через руїни.

Приклад виконання категоризації об’єкта та формування кейсу рекомендацій щодо виконання місії порятунку наведено на рис. 4.7.

Також важливі поради щодо технічних засобів:

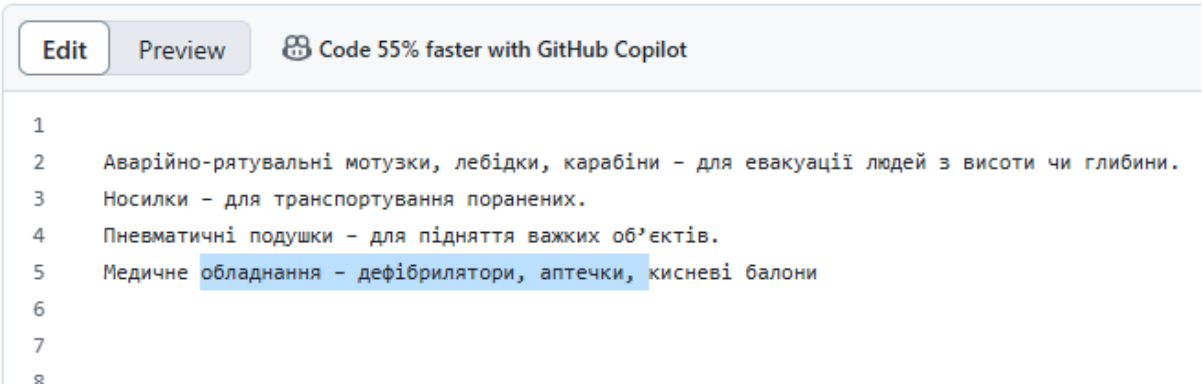
– якщо важко пройти через завали або інші природні або штучні перешкоди, забезпечити евакуацію технічними засобами (вантажівками, бронетехнікою або гелікоптерами);

– провітрювання та опалення приміщень (якщо є ймовірність перебування потерпілого в закритому приміщенні);

– ці вказівки є лише вказівками для кожного типу події. У реальній ситуації важливі деталі можуть відрізнятися залежно від конкретного характеру операції, складності доступу до жертви та кількості ресурсів.

Тека з файлами за допомогою токену відразу після створення завантажується до GitHub-репозиторію [22].

а)



б)

Рисунок 4.7 – Розміщення проєкту на репозиторію GitHub: а – Фото піддослідного об’єкту за категорією «людина»; б – файл рекомендацій

Зберігання результатів на GitHub було вибрано, тому що це полегшує контроль версій і зручний доступ до результатів прогнозування. GitHub дозволяє зберігати дані, відстежувати зміни та ділитися інформацією з іншими користувачами, що важливо для спільної роботи та аналізу результатів. Також він є безкоштовним.

Рекомендації, які надаються наразі є поверхневими. Для подальшого вдосконалення слід опитати рятувальників щодо того, яке обладнання вони використовують на місці події та які транспортні засоби доступні для евакуації постраждалих. Це дозволить точніше адаптувати модель до реальних умов і адаптувати її до конкретних потреб надзвичайних ситуацій.

4.4 Недоліки поточної реалізації та можливості для поліпшення

У поточній версії системи є деякі важливі обмеження, які можуть вплинути на ефективність її роботи в реальних умовах пошуково-рятувальних робіт. Одним із головних обмежень є використання Raspberry Pi, який має обмежені ресурси для виконання складних завдань машинного навчання. Час передбачення на Raspberry Pi становить приблизно 1,16–1,20 секунди, що, мабуть, занадто довго для реального часу. Щоб скоротити цей час, ви можете розглянути можливість використання потужнішої версії Raspberry Pi, як-от Raspberry Pi 4 із більшою кількістю оперативної пам’яті, або використання спеціалізованого чипа машинного навчання, такого як Google Coral Edge TPU, який може значно пришвидшити прогнозування процесу. і забезпечити необхідну швидкість обробки без втрати точності, це дозволить збільшити продуктивність системи без необхідності переходу на дорогі сервери або інші більш потужні платформи.

Крім того, модель може сплутати об’єкти зі схожими характеристиками, наприклад котів і собак або дорослих і дітей. Модель InceptionV3 може втратити точність класифікації зі збільшенням відстані від об’єкта. Це особливо важливо, коли предмети знаходяться далеко. Це може призвести до неправильної класифікації. Щоб вирішити цю проблему, можна застосувати методи доповнення даних для створення більш різноманітних зображень, що дозволить моделі краще розрізняти подібні об’єкти. Ви також можете використовувати чудові набори даних місцевості, щоб зобразити об’єкти під різними кутами та з різних відстаней.

Ці методи значно підвищать точність моделі InceptionV3, зробивши її більш ефективною для практичних пошуково-рятувальних місій.

Висновок до розділу 4

Тестування апаратних і програмних систем з використанням дронів для пошуково-рятувальних місій виявило кілька важливих результатів щодо ефективності системи в реальних умовах. Хоча система на даний момент не використовує дрони, виявилося, що вона здатна категоризувати об’єкти з високою точністю за допомогою традиційних методів обробки зображень. Випробування моделі Inception V3 показали високу точність у режимі реального часу (приблизно 99 %), що є значним досягненням для категоризації об’єктів на основі камери (особливо людей і тварин), навіть у складних місцях, таких як великі відстані або низька якість зображення. вірно за умов.

Модель показує хороші результати на потужному апаратному забезпеченні (ПК), де час прогнозування на зображення становить близько 100–125 мс. Однак, для Raspberry Pi прогнозований час становить близько 1,16–1,2 с, що може бути обмеженням системи для використання в реальному часі в польових умовах. Але в той же час, обробка зображень і категоризація об'єктів досить швидкі, що дозволяє інтегрувати цю модель в реальні рятувальні операції.

Було вирішено додавати результати на GitHub, це зручний і безкоштовний спосіб зберігання інформації. Такий підхід дозволяє зберегти знайдені об’єкти та рекомендації щодо порятунку, забезпечуючи доступність та зручність для подальшого аналізу та використання.

Перспективою розвитку роботи слід вважати вдосконалення поточних рекомендацій системи щодо порятунку згідно з прийнятими нормативними документами. З цієї причини також доцільно провести опитування рятувальників і визначити обладнання та транспортні засоби, які вони використовують, щоб можна було точніше налаштувати модель та зробити її більш корисною в реальних ситуаціях.

ВИСНОВКИ

Під час виконання кваліфікаційної магістерської роботи проведено ретельне дослідження та розробка алгоритмів для БПЛА, призначеними для виконання пошуково-рятувальних завдань. Сучасні реалії покладають на рятувальні служби все більш складні завдання, що вимагають швидкого реагування у важкодоступних районах, стихійних або техногенних катастрофах. У цьому контексті дрони стали незамінними помічниками зі своїми унікальними можливостями. Система, розроблена в рамках цієї роботи, спрямована на суттєве підвищення ефективності таких операцій шляхом впровадження найсучасніших алгоритмів машинного навчання, адаптованих до потреб пошукових завдань.

На початковому етапі роботи було проведено детальний аналіз сучасних моделей і платформ, що використовуються в пошуково-рятувальних операціях. Особливу увагу приділено дронам DJI Matrice 300 RTK, Parrot Anafi USA та Autel Robotics EVO II Dual, які є передовими прикладами апаратних рішень у цій сфері. Проведений аналіз дозволяє виділити їх основні переваги, серед яких точність навігації, висока автономність і здатність працювати в умовах поганої видимості.

Ключовим завданням цієї роботи є створення ефективної системи для безпілотників, здатної автоматично визначати та категоризувати об’єкти в реальному часі. Для цього було обрано сучасний підхід до розробки на основі нейронної мережі GoogLeNet InceptionNet V3 та бібліотеки TensorFlow Lite. Використання останньої стало стратегічно важливим рішенням, оскільки дозволяє адаптувати високоточні моделі для роботи на платформах з обмеженими обчислювальними ресурсами. Це забезпечує найкращий баланс між точністю та продуктивністю, що є критичним для автономних систем.

Розроблена система включає апаратний компонент на основі мікрокомп’ютера Raspberry Pi 3 Model B, який забезпечує обчислювальну потужність, необхідну для виконання завдань комп’ютерного зору. Поєднання

цього одноплатного комп’ютера з камерою високої роздільної здатності створює потужну платформу для реалізації алгоритмів розпізнавання об’єктів. Завдяки підключенню модулів Wi-Fi і Bluetooth система отримує можливість дистанційного керування, що значно розширює її функціональність в реальних умовах експлуатації.

Одним із важливих аспектів цієї роботи є навчання моделі на унікальному наборі даних, який охоплює основні категорії, знайдені під час пошуково-рятувальних місій: люди, тварини, будинки. Під час цього процесу були створені інструменти для генерації наборів даних та їх оптимізації у форматі TFRecord. Такий підхід забезпечує ефективну організацію даних і високу швидкість навчання моделі, що вкрай важливо для забезпечення точності в задачах розпізнавання та надає великі можливості для розширення моделі в майбутньому .

Система, нажаль, не тестувалася в умовах, максимально наближених до реальних пошуково-рятувальних операцій. Але система показує здатність швидко ідентифікувати об'єкти навіть у складних умовах, таких як слабке освітлення. Алгоритм забезпечує точність віднесення об’єкту до певної категорії (людина, тварина або неживий об’єкт), що перевищує існуючі базові рішення на ринку.

Окрім пошуково-рятувальних операцій, результати цієї роботи мають потенціал для широкого застосування в таких сферах, як моніторинг навколишнього середовища, перевірки інфраструктури, оборонна промисловість і навіть сільське господарство. Це підкреслює загальність та актуальність розробленого рішення, яке можна адаптувати до різних потреб.

Отже, результати роботи підтверджують ефективність розробленого методу у вдосконаленні алгоритмів БПЛА. Система не тільки забезпечує технологічний прорив у сфері автоматизації пошуково-рятувальних операцій, але й відкриває нові можливості для розвитку сучасних технологій у багатьох інших напрямках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Лосіцький П. М., Журавська І. М. Розробка та вдосконалення алгоритмів управління БПЛА для місій пошуку та порятунку. *Ольвійський форум-2024*: *Стратегії країн Причорноморського регіону в геополітичному просторі*. Миколаїв: ЧНУ ім. Петра Могили, 2024. С. 168–170.

2. Mano S., Sreehari V. M. Ambulance drone for rescue – A perspective on conceptual design, Life detection systems, and prototype development. *In book: Internet of Drones*. 1st Edition. CRC Press, 2023. Chapter 6. 12 p. DOI: 10.1201/9781003252085.

3. Best Search and Rescue (SAR) Drones. URL: https://www.jouav.com/search-and-rescue-drone#:~:text=What%20are%20SAR%20drones%3F,injured%20people%20in%20emergency%20situations. (Last accessed: 05.10.2024).

4. Drone for rescue at sea and system for rescue at sea using thereof URL: https://patents.google.com/patent/KR101801348B1/en. (Last accessed: 05.10.2024).

5. Star-X VTOL-6520HP 160kg MTOW Long range >1100km 50L Fuel tank Commerical Drone for Rescue, Satcom Relay (Global Warehouse) URL: https://www.austars-model.com/starx-vtol6520hp-160kg-mtow-long-range-1100km-50l-fuel-tank-commerical-drone-for-rescue-satcom-relay_g25552.html (Last accessed: 05.10.2024).

6. Дослідження Жанети Пржиходової присвячені створенню концепції рятувального дрона. URL: https://webdev.fa.cvut.cz/galerie/diplomove-prace/2015-1-prihodova-osobni-zachranarsky-dron-uav 57544d5b501653748408fe2a/diplomova-prace.pdf (дата звернення: 05.10.2024).

7. Квадрокоптер DJI Matrice 300 RTK. URL: https://flytechnology.ua/ru/kvadrokopter-dji-matrice-300-rtk (дата звернення: 05.10.2024).

8. Parrot Anafi USA. URL: https://www.parrot.com/us/parrot-drones-trusted-proven-globally (Last accessed: 05.10.2024).

9. Autel Robotics EVO II Dual. URL: https://shop.autelrobotics.com/pages/evo-ii-dual-specification (Last accessed: 05.10.2024).

10. You Only Look Once (YOLO): What is it? URL: https://datascientest.com/en/you-only-look-once-yolo-what-is-it (Last accessed: 05.10.2024).

11. Single Shot MultiBox Detector (SSD) URL: https://medium.com/@jesse419419/single-shot-multibox-detector-ssd-explained-by-a-j-dda10ba42a29. (Last accessed: 05.10.2024).

12. GoogLeNet Explained: The Inception Model that Won ImageNet URL: https://viso.ai/deep-learning/googlenet-explained-the-inception-model-that-won-imagenet/ (Last accessed: 05.10.2024).

13. Gageik N., Reul C., Montenegro S. Autonomous quadrocopter for search, count and localization of objects. *In book: Recent Advances in Robotic Systems.* Ch. 1. INTECH, Sept. 2016. 26 p. DOI: 10.5772/63568.

14. Bloesch M., Omari S., Hutter M., Siegwart R. Robust visual inertial odometry using a direct EKF-based approach. *In: IEEE/RSJ Int. Conf. Intell. Robot. Syst.* 2015. DOI: 10.1109/IROS.2015.7353389.

15. Smart RTH (Return-to-Home). URL: https://store.dji.com/content/how-to-use-the-djis-return-to-home (Last accessed: 05.10.2024).

16. Naidoo Yo., Stopforth R., Bright G. Development of an UAV for search & rescue applications. 2011. DOI: 10.1109/AFRCON.2011.6072032.

17. Blackwelder P. S. UAV Flight Dynamics. *International Journal of Aeronautical Science & Aerospace Research.* 2015. P. 81–85. DOI: 10.19070/2470-4415-150009.

18. Chityala R., Pudipeddi S. Segmentation. *In book: Image Processing and Acquisition using Python*. 2nd edition. Boca Raton: Chapman & Hall/CRC Press, 2020. P. 167–194. DOI: 10.1201/9780429243370-8.

19. Рятувальні операції. Дрони летять на допомогу. URL: https://www.dronarium.com.ua/uk/uslugi/spasatelnye-operacii/ (дата звернення: 05.10.2024).

20. Real time object detection on a Raspberry Pi URL: https://www.diva-portal.org/smash/get/diva2:1361039/FULLTEXT01.pdf (Last accessed: 05.10.2024).

21. Thien Khai Tran, Kha Tu Huynh, Dac-Nhuong Le, Muhammad Arif, Hoa Minh Dinh. A Deep Trash Classification Model on Raspberry Pi 4. 2023 P. 13. DOI: 10.32604/iasc.2023.029078.

22. GitHub-репозиторій для збереження даних URL: https://github.com/gfdtk123/tensorflow_v2_models_nets/tree/main (дата звернення: 05.10.2024)

23. Global Medical Knowledge Alliance (GMKA). URL: https://gmka.org/uk/zahody-resustsytatsiyi-ta-hirurgichna-dopomoga-v-neprystosovanyh-umovah-arsc-sccc (дата звернення: 05.10.2024)

ДОДАТОК А

Код програми

Додаток А.1. Код програми *create_labels_files.py*

```
import os
import os.path
def write_txt(content, filename, mode='w'):
    with open(filename, mode) as f:
        for line in content:
            str_line = ""
            for col, data in enumerate(line):
                if not col == len(line) - 1:
                    # Розділяє значення пробілами
                    str_line = str_line + str(data) + " "
                else:
                    # Для останнього елемента додає новий рядок
                    str_line = str_line + str(data) + "\n"
            f.write(str_line)
def get_files_list(dir):
    files_list = []
    for parent, dirnames, filenames in os.walk(dir):
        for filename in filenames:
            # Отримуємо ім'я поточної папки
            curr_file = parent.split(os.sep)[-1]
            if curr_file == 'animal':
                label = 0
            elif curr_file == 'cattle':
                label = 1
            elif curr_file == 'horse':
                label = 2
            elif curr_file == 'houses':
                label = 3
            elif curr_file == 'human':
                label = 4
            files_list.append([os.path.join(curr_file, filename), label])
    return files_list
if __name__ == '__main__':
    train_dir = 'dataset/train'  # Тека з навчальними даними
    train_txt = 'dataset/train.txt'  # Шлях до файлу для збереження
    train_data = get_files_list(train_dir)  # Отримуємо список файлів
    write_txt(train_data, train_txt, mode='w')  # Зберігаємо у txt
    val_dir = 'dataset/val'  # Тека з валідаційними даними
    val_txt = 'dataset/val.txt'  # Шлях до файлу для збереження
    val_data = get_files_list(val_dir)  # Отримуємо список файлів
    write_txt(val_data, val_txt, mode='w')  # Зберігаємо у txt
```

Додаток А.2 Код програми *create_tf_record.py*

```
import tensorflow as tf
import numpy as np
import os
import cv2
import random
import matplotlib.pyplot as plt

# === ДОПОМІЖНІ ФУНКЦІЇ ===
def _int64_feature(value):
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
def _bytes_feature(value):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
def load_labels_file(filename, shuffle=False):
    """Завантажує список зображень і міток із текстового файлу."""
    images, labels = [], []
    with open(filename) as f:
        lines = f.readlines()
        if shuffle:
            random.shuffle(lines)
        for line in lines:
            path, label = line.strip().split(' ')
            images.append(path)
            labels.append(int(label))
    return images, labels
def read_image(filename, resize_height, resize_width, normalization=False):
    """Читає зображення, змінює розмір і, за потреби, нормалізує."""
    bgr_image = cv2.imread(filename)
    if bgr_image is None:
        raise ValueError(f"Cannot read image: {filename}")
    rgb_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2RGB)
    if resize_height > 0 and resize_width > 0:
        rgb_image = cv2.resize(rgb_image, (resize_width, resize_height))
    if normalization:
        rgb_image = rgb_image / 255.0
    return np.asarray(rgb_image)
def create_records(image_dir, label_file, output_file, resize_height, resize_width, shuffle=True,
log=10):
    """Створює TFRecord із зображень та міток."""
    images, labels = load_labels_file(label_file, shuffle)
    writer = tf.io.TFRecordWriter(output_file)

    for i, (image_name, label) in enumerate(zip(images, labels)):
        image_path = os.path.join(image_dir, image_name)
        if not os.path.exists(image_path):
            print(f"Warning: Image not found - {image_path}")
            continue
        image = read_image(image_path, resize_height, resize_width)
        image_raw = image.tobytes()
```

```
        feature = {
            'image_raw': _bytes_feature(image_raw),
            'height': _int64_feature(image.shape[0]),
            'width': _int64_feature(image.shape[1]),
            'depth': _int64_feature(image.shape[2]),
            'label': _int64_feature(label)
        }
        example = tf.train.Example(features=tf.train.Features(feature=feature))
        writer.write(example.SerializeToString())
        if i % log == 0 or i == len(images) - 1:
            print(f"Processed {i + 1}/{len(images)} images")
    writer.close()
    print(f"TFRecord file created: {output_file}")
def read_records(file_path, resize_height, resize_width, normalization_type=None):
    """Читує дані з TFRecord і перетворює їх у tf.data.Dataset."""
    def parse_example(serialized_example):
        features = tf.io.parse_single_example(
            serialized_example,
            features={
                'image_raw': tf.io.FixedLenFeature([], tf.string),
                'label': tf.io.FixedLenFeature([], tf.int64)
            }
        )
        image = tf.io.decode_raw(features['image_raw'], tf.uint8)
        image = tf.reshape(image, [resize_height, resize_width, 3])
        image = tf.cast(image, tf.float32)
        if normalization_type == 'normalization':
            image /= 255.0
        elif normalization_type == 'standardization':
            image = (image - 127.5) / 127.5
        label = features['label']
        return image, label
    dataset = tf.data.TFRecordDataset(file_path)
    dataset = dataset.map(parse_example)
    return dataset
def disp_records(record_file, resize_height, resize_width, show_nums=3):
    """Відображає кілька зображень із TFRecord для перевірки."""
    dataset = read_records(record_file, resize_height, resize_width,
normalization_type='normalization')
    for i, (image, label) in enumerate(dataset.take(show_nums)):
        plt.imshow(image.numpy())
        plt.title(f"Label: {label.numpy()}")
        plt.axis('off')
        plt.show()
def get_example_nums(tfrecord_file):
    dataset = tf.data.TFRecordDataset(tfrecord_file)
    return sum(1 for _ in dataset)
if __name__ == '__main__':
    # Налаштування
    resize_height = 229
    resize_width = 229
```

```
    shuffle = True
    # Шляхи до даних
    train_image_dir = "dataset/train"
    train_label_file = "dataset/train.txt"
    train_record_file = "dataset/record/train.tfrecords"
    val_image_dir = "dataset/val"
    val_label_file = "dataset/val.txt"
    val_record_file = "dataset/record/val.tfrecords"
    # Створення TFRecord-файлів
    print("Creating training TFRecord...")
    create_records(train_image_dir, train_label_file, train_record_file, resize_height,
resize_width, shuffle)
    print("Creating validation TFRecord...")
    create_records(val_image_dir, val_label_file, val_record_file, resize_height, resize_width,
shuffle)
    # Відображення даних для перевірки
    print("Displaying samples from training TFRecord...")
    disp_records(train_record_file, resize_height, resize_width)
```

Додаток А.3 Код програми тренування моделі

```
import tensorflow as tf
from create_tf_record import *  # Імпортуємо функції
import numpy as n2
# === Налаштування ===
resize_height = 229
resize_width = 229
batch_size = 32
epochs = 10
# Шляхи до TFRecord-файлів
train_record_file = "dataset/record/train.tfrecords"
val_record_file = "dataset/record/val.tfrecords"
# === Тренувальна функція ===
def train(train_record_file, val_record_file, resize_height, resize_width, batch_size, epochs):
  print(f"Tensorflow version:{tf.__version__}")
 # Перевірка кількості прикладів
   train_nums = get_example_nums(train_record_file)
   val_nums = get_example_nums(val_record_file)
   print(f"Train nums: {train_nums}, Val nums: {val_nums}")
   if train_nums == 0 or val_nums == 0:
      raise ValueError("Train or validation dataset is empty. Check your TFRecord files.")
   # Розрахунок steps_per_epoch
   train_steps = int(np.ceil(train_nums / batch_size))
   val_steps = int(np.ceil(val_nums / batch_size))
   # Зчитування даних
   train_dataset = read_records(train_record_file, resize_height, resize_width,
normalization_type='normalization')
   val_dataset = read_records(val_record_file, resize_height, resize_width,
normalization_type='normalization')
   # Повторення даних і формування батчів
   train_dataset = train_dataset.repeat().shuffle(buffer_size=1000).batch(batch_size).prefetch(
      buffer_size=tf.data.experimental.AUTOTUNE)
   val_dataset =
val_dataset.repeat().batch(batch_size).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
   # Побудова моделі InceptionV3
   base_model = tf.keras.applications.InceptionV3(
      input_shape=(resize_height, resize_width, 3),
      include_top=False,
      weights='imagenet'
   )
   base_model.trainable = False  # Заморожуємо основну модель
   # Додаємо класифікатор
   model = tf.keras.Sequential([
      base_model,
      tf.keras.layers.GlobalAveragePooling2D(),
      tf.keras.layers.Dense(1024, activation='relu'),
```

```
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(5, activation='softmax')  # 5 класів
    ])
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    # Тренування моделі
    history = model.fit(
        train_dataset,
        epochs=epochs,
        steps_per_epoch=train_steps,
        validation_data=val_dataset,
        validation_steps=val_steps
    )
    # Збереження моделі
    model.save("models/inception_v3/inception_v3_model.keras")
    print("Model saved to inception_v3_model.keras")
# === Запуск тренування ===
if __name__ == "__main__":
    train(
        train_record_file=train_record_file,
        val_record_file=val_record_file,
        resize_height=resize_height,
        resize_width=resize_width,
        batch_size=batch_size,
        epochs=epochs
    )
```

Додаток А.4 Код програми на Rispberry Pi

```
import tensorflow as tf
import numpy as np
import cv2
import os
from datetime import datetime
import base64
import requests

# Налаштування GitHub
GITHUB_TOKEN = "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"
GITHUB_REPO = "gfdtk123/tensorflow_v2_models_nets"
# Завантаження моделі TensorFlow Lite

MODEL_PATH = r"model.tflite"
if not os.path.exists(MODEL_PATH):
    raise FileNotFoundError(f"Файл моделі {MODEL_PATH} не знайдено!")
interpreter = tf.lite.Interpreter(model_path=MODEL_PATH)
interpreter.allocate_tensors()
print("Модель успішно завантажена (TensorFlow Lite)!")

# Завантаження класів із файлу
LABELS_PATH = "label.txt"
if not os.path.exists(LABELS_PATH):
    raise FileNotFoundError(f"Файл класів {LABELS_PATH} не знайдено!")
class_labels = np.loadtxt(LABELS_PATH, str, delimiter="\t")
print(f"Класи завантажено: {class_labels}")

# Функція для обробки кадрів
def preprocess_frame(frame, target_size=(229, 229)):
    resized_frame = cv2.resize(frame, target_size)
    normalized_frame = resized_frame / 255.0  # Нормалізація
    return np.expand_dims(normalized_frame, axis=0)

# Функція для створення папки з рекомендаціями
def create_recommendation_folder(predicted_class, image_name):
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    folder_name = f"recommendations/{timestamp}_{class_labels[predicted_class]}"
    os.makedirs(folder_name, exist_ok=True)

    # Додавання рекомендаційного файлу до папки
    recommendation_file = f"recommendations/{class_labels[predicted_class]}_rec.txt"
    if os.path.exists(recommendation_file):
        destination_file = os.path.join(folder_name, os.path.basename(recommendation_file))
        with open(recommendation_file, "r") as src, open(destination_file, "w") as dest:
            dest.write(src.read())
        print(f"Рекомендація скопійована до: {destination_file}")
    else:
        print(f"Рекомендаційний файл {recommendation_file} не знайдено.")
```

```
    # Переміщення збереженого зображення до папки
    os.rename(image_name, os.path.join(folder_name, image_name))
    return folder_name

# Функція для завантаження файлу на GitHub
def upload_file_to_github(file_path, github_path):
    url = f"https://api.github.com/repos/{GITHUB_REPO}/contents/{github_path}"
    with open(file_path, "rb") as file:
        encoded_content = base64.b64encode(file.read()).decode("utf-8")
    data = {
        "message": f"Завантажено: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}",
        "content": encoded_content,
    }
    headers = {"Authorization": f"Bearer {GITHUB_TOKEN}"}
    response = requests.put(url, json=data, headers=headers)
    if response.status_code == 201:
        print(f"Файл {file_path} успішно завантажено як {github_path}.")
    else:
        print(f"Помилка при завантаженні {file_path}: {response.status_code} - {response.text}")
# Функція для завантаження папки на GitHub
def upload_folder_to_github(folder_path):
    for root, _, files in os.walk(folder_path):
        for file in files:
            relative_path = os.path.relpath(os.path.join(root, file), folder_path)
            github_path = f"{os.path.basename(folder_path)}/{relative_path}"
            upload_file_to_github(os.path.join(root, file), github_path)
# Підключення до камери
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise Exception("Не вдалося отримати доступ до камери.")
# Отримуємо тензори моделі
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
try:
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Помилка зчитування кадру з камери.")
            break
        # Обробка кадру
        input_frame = preprocess_frame(frame)
        interpreter.set_tensor(input_details[0]['index'], input_frame.astype(np.float32))
        interpreter.invoke()
        # Отримання результату
        predictions = interpreter.get_tensor(output_details[0]['index'])
        predicted_class = np.argmax(predictions, axis=1)[0]
        class_probability = predictions[0][predicted_class] * 100  # Відсоток для передбаченого
класу
        # Відображення результатів
        label = class_labels[predicted_class]
```

```
        cv2.putText(frame, f"Class: {label} ({class_probability:.2f}%)", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 255, 0), 2)
        cv2.imshow("Розпізнавання", frame)
        # Зупинка аналізу при точності >= 75%
        if class_probability >= 75.0:
            image_name = f"{label}_{class_probability:.2f}.jpg"
            cv2.imwrite(image_name, frame)
            print(f"Збережено зображення: {image_name}")
            folder_path = create_recommendation_folder(predicted_class, image_name)
            upload_folder_to_github(folder_path)
            print("Рекомендації завантажено. Завершення аналізу.")
            break
        # Вихід із програми (при натисканні 'q')
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    cap.release()
    cv2.destroyAllWindows()
```

Додаток А.5 Код прогнозування моделі

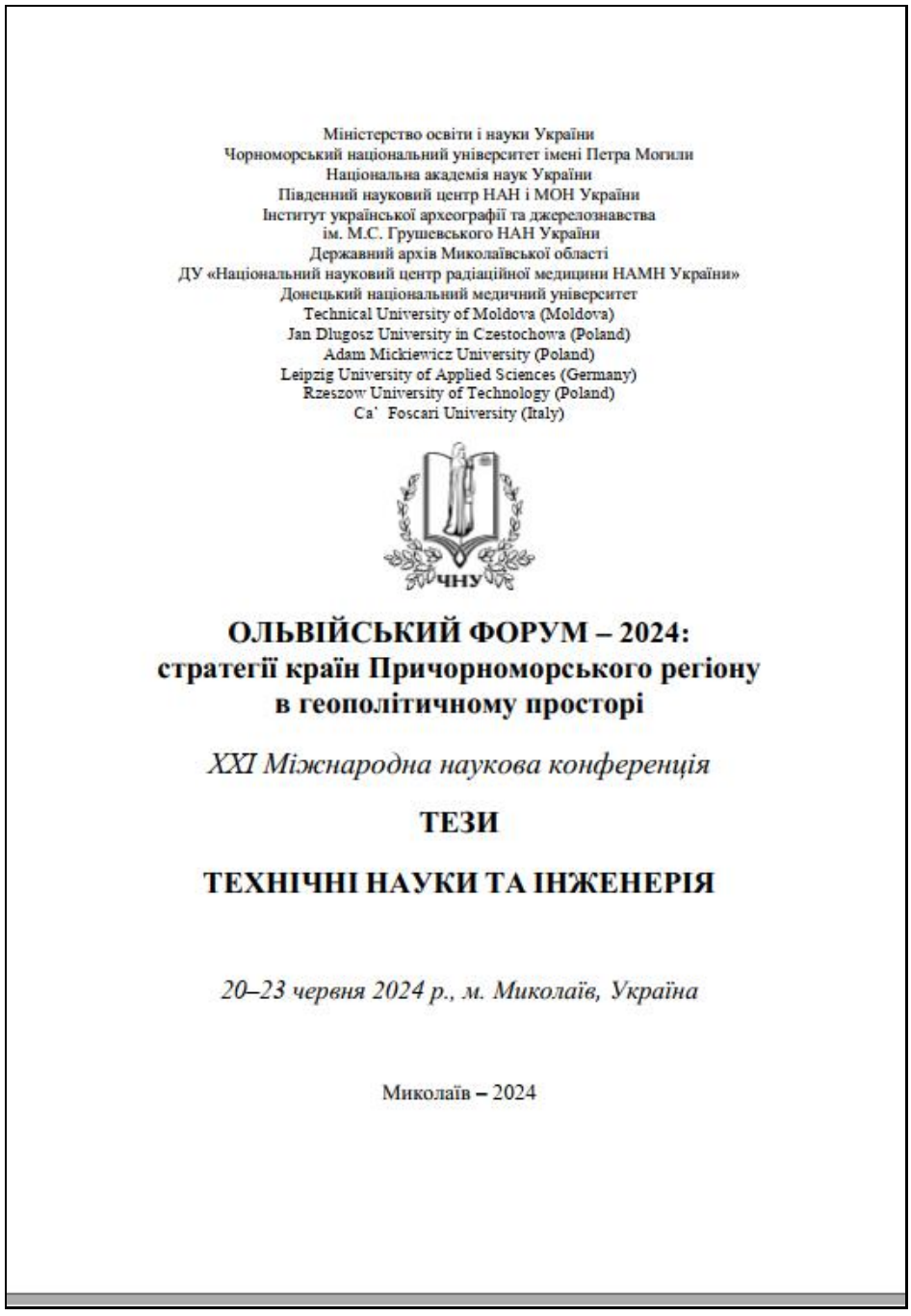
```
import tensorflow as tf
import numpy as np
import os
import glob
import cv2
import time  # Імпортуємо модуль для вимірювання часу
# Функція для читання та передобробки зображень
def read_image(image_path, resize_height, resize_width, normalization=True):
    """Завантажує зображення, змінює його розмір і нормалізує."""
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Cannot read image: {image_path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (resize_width, resize_height))
    if normalization:
        img = img / 255.0  # Нормалізуємо до [0, 1]
    return img
# Функція для завантаження етикеток класів
def load_labels(labels_filename):
    """Завантажує етикетки класів з текстового файлу."""
    if not os.path.exists(labels_filename):
        raise FileNotFoundError(f"Labels file not found: {labels_filename}")
    try:
        labels = np.loadtxt(labels_filename, dtype=str, delimiter='\t')
    except Exception as e:
        raise ValueError(f"Error reading labels file: {e}")
    return labels
# Функція для прогнозування
def predict(model_path, image_dir, labels_filename, labels_nums, resize_height, resize_width):
    """Завантажує модель TFLite та прогнозує класи для зображень."""
    # Завантаження етикеток класів
    labels = load_labels(labels_filename)
    # Завантаження моделі TFLite
    if not os.path.exists(model_path):
        raise FileNotFoundError(f"TFLite model file not found: {model_path}")
    interpreter = tf.lite.Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    # Отримуємо індекси тензорів для введення і виведення
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    # Перевірка розмірності класів
    if output_details[0]['shape'][1] != labels_nums:
        raise ValueError(f"Model output classes ({output_details[0]['shape'][1]}) "
                         f"do not match the provided label numbers ({labels_nums}).")
    # Отримуємо список зображень для прогнозування
    images_list = glob.glob(os.path.join(image_dir, '*.jpg'))
    if not images_list:
```

```
        print(f"No images found in directory: {image_dir}")
        return
    # Прогнозування для кожного зображення
    for image_path in images_list:
        img = read_image(image_path, resize_height, resize_width)
        img = np.expand_dims(img, axis=0).astype(np.float32)  # Додаємо batch dimension і
приводимо до float32
        # Починаємо вимірювати час
        start_time = time.time()
        # Встановлюємо значення тензора введення
        interpreter.set_tensor(input_details[0]['index'], img)
        # Запускаємо модель
        interpreter.invoke()
        # Отримуємо прогноз
        pre_score = interpreter.get_tensor(output_details[0]['index'])[0]
        pre_label = np.argmax(pre_score)  # Клас із найбільшою ймовірністю
        max_score = pre_score[pre_label]
        # Вимірюємо час виконання
        end_time = time.time()
        elapsed_time = end_time - start_time
        # Округляємо час до мілісекунд
        elapsed_time_ms = round(elapsed_time * 1000, 3)
        # Виведення результатів
        print(f"{image_path} is: pre labels: {pre_label}, name: {labels[pre_label]}, score:
{max_score:.4f}")
        print(f"Time taken for prediction: {elapsed_time:.4f} seconds ({elapsed_time_ms} ms)")
if __name__ == '__main__':
    class_nums = 5  # Кількість класів
    image_dir = 'test_image'  # Директорія з тестовими зображеннями
    labels_filename = 'label.txt'  # Шлях до файлу з етикетками
    model_path = 'model.tflite'  # Шлях до TFLite моделі
    resize_height = 229  # Висота зображення
    resize_width = 229  # Ширина зображення
    # Запуск функції прогнозування
    predict(model_path, image_dir, labels_filename, class_nums, resize_height, resize_width)
```

ДОДАТОК Б

Апробація роботи



Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Національна академія наук України
Південний науковий центр НАН і МОН України
Інститут української археографії та джерелознавства
ім. М.С. Грушевського НАН України
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Донецький національний медичний університет
Technical University of Moldova (Moldova)
Jan Dlugosz University in Czestochowa (Poland)
Adam Mickiewicz University (Poland)
Leipzig University of Applied Sciences (Germany)
Rzeszow University of Technology (Poland)
Ca' Foscari University (Italy)

ЧНУ

**ОЛЬВІЙСЬКИЙ ФОРУМ – 2024:
стратегії країн Причорноморського регіону
в геополітичному просторі**

XXI Міжнародна наукова конференція

ТЕЗИ

ТЕХНІЧНІ НАУКИ ТА ІНЖЕНЕРІЯ

20–23 червня 2024 р., м. Миколаїв, Україна

Миколаїв – 2024

пам’ять мікрокомп’ютера. Крім того, TinyML дозволяє інтегрувати у вбудовані пристрої штучний інтелект на основі керованих даними алгоритмів.

Список використаних джерел

1. Du M., Chen W., Liu K., Wang L., Hu Y., Mao Y., Sun X., Luo Y., Shi J., Shao K., Huang H., Ye D. The global burden of leukemia and its attributable factors in 204 countries and territories: Findings from the Global Burden of Disease 2019 study and projections to 2030. *Journal of oncology*. 2022, 1612702. DOI: 10.1155/2022/1612702.
2. Мікрокомп’ютер Raspberry Pi Zero W. *Евоком.юа*. URL: https://evo.net.ua/raspberry-pi-zero-w/ (дата звернення: 22.04.2024).
3. Spring Cloud Stream App Starters Reference Guide. *Spring*. 2020. URL:https://docs.spring.io/spring-cloud-stream-app-starters/docs/
4. current/reference/htmlsingle/#spring-cloud-stream-modules-tensorflow-processor (Last accessed: 25.04.2024).
5. Simeu B. N. Convolutional Neural Network using TensorFlow. *Medium*. 2023. URL: https://medium.com/@BriceNicodem/convolutional-neural-network-using-tensorflow-3d5b5d3a38a5 (Last accessed: 27.04.2024).

УДК 004.896:629.7.07

Лосіцький П. М.,
студент 5-го курсу,
Журавська І. М.,
д-р техн. наук, професор,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

РОЗРОБКА ТА ВДОСКОНАЛЕННЯ АЛГОРИТМІВ УПРАВЛІННЯ БПЛА ДЛЯ МІСІЙ ПОШУКУ ТА ПОРЯТУНКУ

На сьогодинній день існує кілька алгоритмів, які використовуються для керування безпілотними літальними апаратами (БПЛА або дрон) для пошуково-рятувальних завдань. Серед них можна відзначити алгоритми на основі класичних методів, такі як ПІД-регулятори (англ. *Proportional–Integral–Derivative Controller – PID*) та алгоритми планування шляху, а також алгоритми на основі штучного інтелекту (ШІ), такі як нейронні мережі та генетичні алгоритми.

168