МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ МАШИНО-ЛЮДИННОГО ІНТЕРФЕЙСУ БОРТОВОЇ СИСТЕМИ РОЗПІЗНАВАННЯ НАЗЕМНИХ ОБ'ЄКТІВ

Здобувач освіти гр. ІН.м-01н Є.О. Гладких

Науковий керівник,
кандидат технічних наук, доцент,
доцент кафедри комп’ютерних наук І.В. Шелехов

Завідувач кафедри
доктор технічних наук, професор А.С. Довбиш

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *122 «Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
" _____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Гладких Євгеній Олександрович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проектування машино-людинного інтерфейсу бортової системи розпізнавання наземних об'єктів*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Інформаційний огляд 2) Вибір програмних засобів 3) Практична реалізація*

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми та постановка задачі*		
2.	*Вибір алгоритму*		
3.	*Інформаційне та програмне забезпечення системи*		
4.	*Оформлення пояснювальної записки до дипломної роботи*		

Здобувач вищої освіти _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 69 стор., 23 рис., 2 додатка, 41 джерело.

Об’єкт дослідження — процес розпізнавання наземних об’єктів, що використовуються в машино-людинному інтерфейсі бортових систем БПЛА.

Мета роботи — розробка інформаційної технології оптичного розпізнавання об’єктів зображення в віртуальному просторі.

Методи дослідження — методи обробки зображення, методи розпізнавання образів.

Результати — розроблено інформаційне, програмне та алгоритмічне забезпечення системи інтелектуального аналізу даних з зображень з метою виявлення розпізнавання віртуальних об’єктів. В роботі використовується середовище Unity для швидкої реалізації віртуального оточення та поведінки дрону, також було використано фреймворк Darknet для реалізації нейромережі класу YOLO. Розробленене рішення реалізована мовами програмування C та C#.

СИСТЕМА РОЗПІЗНАВАННЯ ОБРАЗІВ, РОЗПІЗНАВАННЯ,
РОЗПІЗНАВАННЯ НАЗЕМНИХ ОБ’ЄКТІВ

ЗМІСТ

ВСТУП 6
1 ІНФОРМАЦІЙНИЙ ОГЛЯД 8
1.1 Сучасні бортові системи безпілотних літальний апаратів 8
1.2 Аналіз предметної області 10
1.3 Актуальність безпілотних літальних апаратів 11
1.4 Людинно-машиний інтерфейс 13
1.4.1 Рівень вимог до взаємодії 14
1.4.2 Вимоги до інформації 16
1.5 Постановка задачі 17
2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ 19
2.1 Основні визначення 19
2.2 Огляд методу виявлення 21
2.2.1 Архітектура мережі 24
2.2.2 Механізм тренування та втрат 25
2.3 Порівняння систем виявлення 27
3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 31
3.1 Тренувальний процес 31
3.2 Збір датасету 31
3.2.1 Аугментування робочих даних 33
3.2.2 Розбиття тренувальних даних 34
3.2.3 Фреймворк Darknet 35
3.2.4 Тренування 39
3.3 Програмна реалізація 42
3.3.1 Вибір програмного середовища 43
3.3.2 Реалізація Дрона 43
3.3.3 Сервіс Altrous 44

3.4 Тестування взаємодії оператору та дрону 45
ВИСНОВКИ 48
СПИСОК ЛІТЕРАТУРИ 49
ДОДАТОК 53

ВСТУП

Сьогодні безпілотна авіація активно розвивається та інтегрується до авіаційної системи загалом. Розроблення та використання безпілотних авіаційних систем проводиться в наш час майже в усіх країнах світу. Порядок технічного обслуговування безпілотних літальних апаратів (БПЛА) визначається метою польоту, правил польотів, районами польоту та іншими функціональними характеристиками. Одним з головних правил виконання польотів для будь-якої авіаційної техніки (як пілотованої, так і безпілотної) є обов’язкове проведення передпольотної підготовки, включаючи перевірку засобів зв’язку і функцій управління, обробку необхідної інформації, а також отримання, за потребою, відповідних дозволів від органів управління повітряним рухом.

За оцінкою американського експерта Джона Уордена, до 2025 р. майже 90% літальних апаратів будуть безпілотними, і лише 10% — безпілотними, а пілоти будуть «золотим резервом» для вирішення найважливіших і найскладніших завдань[1]. Те саме стосується і цивільних БПЛА щодо їх розвитку, це має низку важливих переваг. По-перше, відсутність екіпажу на борту унеможливлює ризик загибелі. По-друге, завдяки своїм розмірам БПЛА можуть виконувати маневри, недоступні для пілотованих апаратів. Також, тривалі безпілотні польоти унеможливлюють потенційний фактор втоми.

Безпілотні літальні апарати мають ряд додаткових переваг: низька вартість експлуатації, хороше маскування, гнучкість і стійкість, простота і доступність технологій порівняно з пілотованими літальними апаратами, а також БПЛА можуть використовуватися в тих випадках, коли використання пілотованих літальних апаратів недоцільне, чи ризиковано [2]. Вони добре показують свою ефективність у групових польотах при спостереженні за лісовими пожежами, передачі зв'язку, пошуково-рятувальних роботах у сільському господарстві, при обробці зернових культур та переміщенні вантажів значно вищі, ніж за одиночних польотів БПЛА.

Технічний прогрес дозволяє сучасним дронам використовувати камери з різним полем зору, що є дуже перспективним для різних комерційних цілей, таких як аерофотозйомка, спостереження тощо. Для масового розгортання дронів та подальшого зниження їх споживання необхідно використовувати дрони з інтелектуальним машинним зором та автопілотом. При використанні аерофотозйомки виявлення та відстеження об'єктів важливі для захоплення ключових об'єктів сцени. Виявлення та відстеження об'єктів – класичні проблеми комп'ютерного зору. Однак із дронами проблем більше через кут огляду зверху вниз і тротлінг у реальному часі. Крім того, складною проблемою є велика вага та обмежена площа вбудованого обладнання, що не дозволяє дронам запускати алгоритми з інтенсивними обчисленнями, такі як глибоке навчання, з обмеженими апаратними ресурсами.

Оператор це спеціаліст дистанційно управляє сучасними літальними апаратами. Умови роботи оператора БПЛА можуть бути різними залежно від сфери діяльності. Оператор дрона, що знаходиться в зоні прямої видимості, може стояти зовні з пультом дистанційного керування та давати команду дрону захопити весь квартал. Військові оператори працюють поза полем зору, перебувають на авіабазі і стежать за обстановкою навколо апарату, що знаходиться за десятки тисяч кілометрів від бази [8].

В рамках даної роботи буде описано створення інтерактивного симулятора управління дроном під назвою "ThirdEye" на програмному двигуні Unity, та нейромережі класу YOLO. Даний продукт буде створений на базі ОС Windows, але також в подальшому може бути портований на інші ОС.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Сучасні бортові системи безпілотних літальний апаратів

В останні роки спостерігається зростаючий інтерес до автономних БПЛА та їх додатків, таких як розвідка та спостереження, пошук та порятунок, а також перевірка інфраструктури[3, 4]. Візуальне виявлення об'єктів є фундаментальним компонентом таких програм БПЛА та має вирішальне значення для розробки повністю автономних систем.

Дрони надають неоціненну допомогу у наданні допомоги при стихійних лих. Оснащені тепловізійними камерами дрони можуть допомогти відслідковувати та визначати місцезнаходження тих, хто вижив. Вони також можуть допомогти працівникам підрахувати кількість переміщених осіб, які потребують притулку.

Дрони використовуються в багатьох експериментах суспільної безпеки в деяких частинах Великобританії. Пілотна програма Fleetlights досліджує, як дрони можуть допомогти пішоходам пересуватися темними дорогами, пропонуючи «персональні вуличні ліхтарі» для зниження ризику нещасних випадків [9].

Прогрес у дослідженнях виявлення об'єктів дронами був повільним, що поступово стало одним із вузьких місць, що обмежують розвиток БПЛА. Від рівня точності та виявлення об'єктів у режимі реального часу залежатиме, чи закінчиться місія дрону знищенням літака або його поведінкою. Алгоритм виявлення об'єктів за допомогою дрону обмежений електричною потужністю, дальністю дії та навколишнім середовищем та має деякі проблеми:

- Нестабільність швидко летючого безпілотного літального апарату означає, що аерофотознімки схильні до оптичних артефактів, таких як розмиття і шум. Крім того, від цих рухомих цілей можна отримати менше інформації про характеристики, дрон може виявляти один і той же об'єкт кілька разів і може виявляти мету помилково [5];

- Об'єкти, що виявляються, зазвичай мають невеликі розміри на зображеннях. Це означає, що коли БПЛА фотографує з великої висоти, дрібні цілі легко не помітити [19];
- Безперервний рух БПЛА та зміни зовнішнього середовища (наприклад, світло, хмари, туман, дощ тощо) призводять до раптових змін характеристик мети на зображенні, збільшуючи таким чином складність вилучення додаткових ознак [6];

Алгоритм виявлення об'єктів за допомогою дронів повинен швидко і точно виявляти цілі, що рухаються, тому алгоритм повинен відповідати вимогам обчислень в реальному часі [7].

Оскільки мета зазвичай здається маленькою на знімках з дроном, риси обличчя часто розмиті, і їх легко сплутати з іншими предметами. Крім того, надто багато фону на зображенні може призвести до завеликої кількості негативних вибірок у процесі навчання, що впливає на точність виявлення. На підставі цих спостережень робота спрямована на підвищення ефективності та точності системи виявлення дронів з урахуванням перелічених вище проблем. Існують моделі виявлення об'єктів, що базуються на ідеї додаткової інфраструктури, яка може зменшити розмір обчислень IoU (Intersection over Union) [20].

Багато досліджень дронів намагалися виявляти і відслідковувати певні типи об'єктів, такі як транспортні засоби [12], людей, включаючи рухомих пішоходів [10], та орієнтири для автономної навігації та приземлення [11] у режимі реального часу. Проте дослідники планують виявити кілька об'єктів. хоча багатоцільове виявлення очевидно важливе для багатьох програм БПЛА [13]. Основні причини такого розриву між потребами додатків та технічними можливостями пов'язані з двома практичними, але важливими обмеженнями:

1) алгоритми розпізнавання об'єктів часто необхідно налаштовувати вручну для певних типів об'єктів та контекстів;

2) складно створювати і зберігати різні моделі цільових об'єктів, особливо коли об'єкти різняться на вигляд;

3) виявлення об'єктів у реальному часі вимагає великої обчислювальної потужності навіть виявлення одного об'єкта, а про кілька цілей.

Однак перша з цих проблем затьмарюється новими революційними технологіями комп'ютерного зору, які добре працюють у різних об'єктах. Більшість цих методів засновані на «глибокому навчанні» згорткових нейронних мереж та забезпечують значне підвищення продуктивності при вирішенні різних завдань розпізнавання [14, 15]. Основна ідея полягає в тому, щоб вивчати об'єктні моделі з необроблених даних пікселів замість використання функцій, що настроюються вручну, як у традиційних підходах до розпізнавання.

Для глибокого навчання моделі часто потрібні великі навчальні набори даних, але ця проблема також вирішена за допомогою нових великомасштабних позначених наборів даних, таких як ImageNet [16]. На жаль, ці нові методи потребують безпрецедентних обчислювальних зусиль; кількість параметрів в об'єктній моделі зазвичай обчислюється мільйонами або мільярдами, для чого потрібні гігабайти пам'яті, а для навчання та розпізнавання з використанням об'єктних моделей потрібні високопродуктивні графічні процесори (GPU).

Використання нових технологій на недорогих та легких безпілотниках неможливе через розмір, вагу та вимоги до потужності цих пристроїв, тому актуальність оптимізації та покращення енергоефективності є одним із головних проблем БПЛА.

1.2 Аналіз предметної області

Автоматизація робочих місць - мегатренд, що стосується більшості ринків. Підприємства всіх видів знайомі з автоматизацією та роботизованою автоматизацією процесів на робочому місці. У деяких звітах йдеться про те, що до 2030 року технології автоматизації можуть зайняти до 800 мільйонів робочих місць.

Дрони не є винятком, їх використовують для повсякденних операцій у різних секторах. Але на відміну від деяких нових технологій, дрони зараз мають більш доповнюючі стосунки з людьми.

Замість того, щоб замінювати людську діяльність, дрони підвищують та покращують загальний інтелект. Існує два способи побачити, як дрони та люди працюють разом. Один проходить поточний етап розширення, а другий дивиться в майбутнє, у час більшої автоматизації.

Поліпшення включатимуть штучний інтелект для обходу перешкод і геозони, а це означає, що дрони можуть бути запрограмовані так, щоб вони обходили заздалегідь встановлені просторові координати, такі як обмежений повітряний простір навколо аеропортів або військових об'єктів. Наприклад Amazon та інші роздрібні торговці проводять випробування, щоб побачити, чи можуть дрони доставки замінити водіїв фургонів для виконання «останньої милі» доставки [17].

Таким чином, наступний переломний момент для дронів, настане з автономними дронами. Вони можуть бачити і літати розумно. Це відкриває можливість для БПЛА грати важливішу роль при меншій взаємодії з людиною.

1.3 Актуальність безпілотних літальних апаратів

У міру збільшення автономності дронів ролі, які грають люди, також змінюватимуться. Майбутні роботи включатимуть ремонт, технічне обслуговування, програмування та пілотування дронів. Людська праця буде спрямована на створення цінності за допомогою збору даних за допомогою дронів, таких як аналіз та моделювання. Переміщення, а не заміна, швидше за все, буде результатом майбутньої екосистеми дронів.

На цьому наступному етапі є перешкоди; Повністю автономного пілотування поки що немає. У нормативному ландшафті, як і раніше, важко орієнтуватися, і належить подолати серйозні соціальні та культурні проблеми.

Але хоча їх розробка вимагає часу, у ландшафті, покращеному дронами, є багато можливостей.

Високий попит на БПЛА дав поштовх розвитку вітчизняних розробок. Зараз ми спостерігаємо значну кількість розробників та ентузіастів, які працюють над створенням та виробництвом БПЛА, призначених насамперед для військових. Про них уже заговорили деякі держпідприємства. Так, ДП «Антонов» заявило про можливе поновлення розробки та подальшого виробництва безпілотників [18].

Використання БПЛА спільно зі стаціонарними або мобільними комплексами ТЗВ дозволяє значно підвищити ефективність охорони протяжних та розосереджених об'єктів, у тому числі тих, що підлягають державній охороні відповідно до Закону України «Про державну охорону органів державної влади та посадових осіб». При цьому БПЛА можуть стати однією із систем зберігання системи на годину масових відвідувань, що має бути особливо важливим для безпеки держави.

Системи БПЛА, оснащені сучасними технічними засобами, вимагають мінімальної підготовки наземного персоналу, вирішуючи широке коло завдань у будь-яких погодних умовах. Вони прості у використанні, мобільні, швидко розгортаються. Можна стверджувати, що високі характеристики комплексів БПЛА знижують експлуатаційні витрати, вимоги до персоналу, підвищують інформативність та повноту інформації для ефективного та оперативного виконання завдань підрозділами ДДБ України, а також зняти деякі фізичні та психофізіологічні обмеження можливостей військовослужбовців.

Переваги та специфічні характеристики комплексів БПЛА вказують на їх роль як високомобільного та ефективного компонента технічних засобів захисту, здатного не лише в короткі терміни зібрати достовірну розвідувальну інформацію, а й поставити завдання щодо наведення на злочинця чи будь-якої іншої загрози дати своєчасне та ефективне рішення щодо їх усунення або

ухвалення рішення про дії охорони без зволікання. Ці характеристики визначають зростання ролі комплексів з БПЛА, які можуть застосовуватися у тактичному та оперативному напрямку (у перспективі – у стратегічному напрямку) у рамках виконання завдань державної безпеки військовослужбовцями різних підрозділів УДО).

Технології БПЛА щороку піднімаються на новий рівень. Повністю автономних систем ще не існує, але в даний момент ми можемо оптимізувати і здійснити ефективну взаємодію оператора і дрону за рахунок більш прогресивних рішень. За рахунок підвищення популяризації і доступності БПЛА, попит на нові технології взаємодії оператора і дрону, розпізнання, управління буде тільки зростати.

1.4 Людинно-машиний інтерфейс

Ефективна взаємодія машин із людиною є однією з основних проблем при проектуванні складних оборонних чи цивільних систем. Наприклад, наступальні місії особливо підходять для застосування технології БПЛА та надзвичайно вимогливі з точки зору робочого навантаження оператора та необхідного рівня автономності. За допомогою моделювання «людина-в-контурі» в режимі реального часу в цьому дослідженні вивчається взаємодія між оператором і БПЛА, при цьому особлива увага приділяється вимогам до різних рівнів автономності, характеристик поведінки пула, інтерфейсу «людина-машина» і каналам передачі даних для управління. кілька БПЛА з одномісного пілотованого бойового літака.

Отримавши необхідний рівень абстракції з погляду призначення та ініціювання будь-якого завдання, це дослідження спрямоване на оптимізацію когнітивних характеристик людино-машинного інтерфейсу та забезпечення БПЛА адекватним рівнем автономності та інтелекту. Для операторів людино-машинний інтерфейс є основною точкою взаємодії та засіб передачі знань між системою та ментальною моделлю людини. Оптимізований людино-машинний

інтерфейс має вирішальне значення для ефективності людини та хорошої ситуаційної обізнаності. Також важливо визначити, яка інформація потрібна оператору на окремих етапах кожної місії, перш ніж вирішувати, як подати цю інформацію.

При розробці макету людино-машинного інтерфейсу важливо використовувати відповідні візуальні підказки, такі як просторове розташування, угруповання, дизайн значків та дизайн категорій, щоб допомогти оператору розпізнавати інформацію та події. Крім того, людино-машинний інтерфейс забезпечує основний спосіб управління рівнем автономності БПЛА, визначаючи ступінь, в якому підрозділ або група БПЛА можуть приймати рішення.

1.4.1 Рівень вимог до взаємодії

Рівень взаємодії, необхідний дією, залежить від розподілу відповідальності за рішення, яке залежить від правил заручення та розвідки БПЛА. Досліджено вплив цих залежностей на робоче навантаження оператора у повітряному середовищі. Визначення оптимального балансу прийняття рішень між оператором і БПЛА вимагає розуміння правил заручин, поведінки покупців, безліч характеристик рішень. правила заручення диктують прийняті межі похибки, дійсні типи цілей, максимальні рівні втрат і те, звідки мають надаватися повноваження на атаку.

Немає чіткого розмежування між обов'язками оператора та дрону, але зі зростанням впевненості у здатності БПЛА приймати обґрунтовані рішення, на БПЛА покладається максимальна відповідальність. Якщо дрон повинен бути задіяний у наступальній місії, він повинен бути в змозі передати свій статус та цілі уповноваженому органу, який, у свою чергу, повинен бути в змозі підтвердити, перенаправити або припинити місію у будь-який час, якщо він має високий рівень самостійності.

З точки зору машинно-людського дизайну, для цього може знадобитися розміщення відео в реальному часі, кнопки для прийняття та відхилення запитів

БПЛА, а також зміна плану місії БПЛА в польоті. На рис. 1.1 показано три рівні людського пізнання: вміння, правила та знання [21]. Вміння викликають автоматичні стосунки один на один, тобто їзду на велосипеді. Правила є процедурними, тобто навчання в надзвичайних ситуаціях або базова тактика. Знання покладаються на довідкову інформацію та досвід.

Рисунок 1.1 – Рівні людського пізнання

З точки зору дизайну інтерфейсу людини-машини, оператор повинен мати можливість взаємодіяти з цими інтелектуальними агентами, що вимагатиме розміщення дисплея стану БПЛА з багатьма командними кнопками завдання, такими як область пошуку або відновлення на базу.

Рисунок 1.2 – Умовний розподіл відповідальності

Умовна схема, показана на рисунку 1.2, перекладає ці рівні в контекст БПЛА [22]. Такі завдання, як відстеження шляхових точок, можна автоматизувати в рамках БПЛА. Такі завдання, як пошук місцевості та знищення цілі, є складнішими і вимагають поєднання підходів, заснованих на правилах і знаннях, для реалізації в БПЛА. Стратегічні рішення можуть залишатися компетенцією оператора, але з часом і зрілістю вони можуть стати обов'язком БПЛА.

З точки зору оператора, БПЛА об’єднані в пул, який інтелектуально виконує вказану основну місію. Оператор може виконувати завдання БПЛА на кількох рівнях. На найнижчому рівні оператор може вказати БПЛА та його основні дії, такі як маневрування за певною схемою, на певному положенні та висоті.

На вищому рівні оператор може вказати завдання верхнього рівня і дозволити системі організувати решту, наприклад, спостерігати за певною метою. На найвищому рівні оператор може вказати тип і місце розташування місії та дозволити агентам свободу планувати та виконувати її, спілкуючись з оператором лише після невдачі або в заздалегідь узгодженій точці критичного рішення.

З точки зору дизайну інтерфейсу людини-машини, оператор повинен мати можливість взаємодіяти з цими інтелектуальними агентами, що вимагатиме розміщення дисплея стану БПЛА з багатьма командними кнопками завдання, такими як область пошуку або відновлення на базу.

1.4.2 Вимоги до інформації

Інформаційні вимоги оператора залежать від призначеного рівня автономності БПЛА, який залежить від правил заручення. Було досліджено потребу в змінних рівнях автономності та вплив, який вони мають на робоче навантаження оператора та ситуаційну обізнаність.

У випадку з БПЛА автономність вимагає певної форми інтелекту. Було запропоновано багато схем, що описують змінні рівні автономності [23], але вони, здається, розроблені як дескриптори можливостей системи і є занадто грубими та складними для безпосереднього впровадження в практичний інтерфейс, як це свідчать з багатьох симуляційних випробувань. Таким чином, у дослідженні UAVOM було прийнято систему рівня автономії Pilot Authorization and Control of Tasks (PACT), яка дозволяє контролювати елементи завдань місії [24].

менеджер інтерфейсу завдань було впроваджено в чотири етапи, з другим етапом в останньому випробуванні. Перший етап дозволив оператору визначити стандартний набір загальних рівнів автономії завдання та набір рівнів автономії для місії та завдання під час планування перед місією. Другий етап дозволяє оператору змінювати рівні автономії під час місії, але тільки між максимальними і стандартними налаштуваннями, визначеними перед місією. Очікується, що останній етап дозволить динамічно оновлювати рівні автономії для окремих завдань за допомогою введення оператора, а завершальний етап дозволить менеджеру інтерфейсу завдань динамічно їх розподіляти. З точки зору дизайну машинно-людинний, включення менеджер інтерфейсу завдань і PACT вимагає розміщення дисплея повідомлень, методу зміни рівнів автономності та методу відображення поточного стану кожного БПЛА.

1.5 Постановка задачі

Метою роботи є розробка інформаційної технології машино-людинного інтерфейсу бортової системи розпізнавання наземних об'єктів. Для досягнення цієї мети нам потрібно виконати такі завдання:

1. Формування вхідного математичного опису бортової системи розпізнавання наземних об'єктів.

2. Вибір типу та структури класифікаторів зображень наземних об'єктів, що використовуються в машино-людинному інтерфейсі бортових систем розпізнавання.
3. Вибір математичних моделей формування класифікаторів зображень наземних об'єктів та критеріїв оцінювання їх якості.
4. Розробка та програмна реалізація алгоритмів оптимізації функціональних параметрів бортової системи розпізнавання наземних об'єктів.
5. Перевірка працездатності розробленого машино-людинного інтерфейсу бортової системи розпізнавання наземних об'єктів.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Основні визначення

Люди дивляться на зображення і миттєво знають, які об'єкти перебувають у зображенні, де вони і як взаємодіють. Зорова система людини швидка і точна, що дозволяє виконувати складні завдання, наприклад водіння, не замислюючись. Швидкі та точні алгоритми виявлення об'єктів дозволять комп'ютерам керувати автомобілями без спеціалізованих датчиків, допоміжні пристрої передають інформацію в режимі реального часу та надають інформацію про сцену для користувачів та розкривають потенціал для роботизованих систем загального призначення.

Сучасні системи виявлення змінюють призначення класифікаторів виконання виявлення. Для виявлення об'єкта ці системи беруть класифікатор для цього об'єкта та оцінюють його у різних місцях та масштабах на тестовому зображенні. Системи як деформовані деталі моделі (DPM) використовують підхід із розсувним вікном, де класифікатор виконується в рівномірно розташованих місцях по всьому зображенню. [25].

Більш нові підходи, такі як R-CNN, використовують методи підказки області, щоб спочатку створити потенційні рамки на зображенні, а потім запустити класифікатор для цих запропонованих прямокутників. Після класифікації використовується постобробка для уточнення рамок, що обмежують, усунення виявлення дублікатів і відновлення оцінки кадру на основі інших об'єктів у сцені. [26]. Ці складні конвеєри повільні і їх важко оптимізувати, оскільки кожен з них окремий компонент необхідно навчати окремо.

Щоб покращити та спростити систему, можно переформулювати виявлення об'єктів як єдине завдання регресії, від пікселів зображення до координат кадру та ймовірностей класів. Ця система називається “you only look once” (YOLO) . Її мета передбачити, які об’єкти присутні і де вони знаходяться.

Рисунок 2.1 – Система виявлення YOLO.

YOLO дуже проста (рис. 2.1) згорткова мережа одночасно передбачає кілька рамок, що обмежують, і ймовірності класів для цих рамок. YOLO тренується на повних кадрах і оптимізує ефективність виявлення. Ця уніфікована модель має низку переваг перед традиційними методами виявлення об'єктів.

По-перше, YOLO дуже швидка. Оскільки структуру визначено як проблему регресії, складний конвеєр не потрібний. Нейронна мережа працює із новим зображенням під час тестування, щоб забезпечити виявлення. Наприклад, якщо використовувати сучасних графічних процесор з базовою мережею, що працює зі швидкістю понад 50 кадрів за секунду без пакетної обробки. Також є спрощена мережа, за рахунок чого вона швидше та працює зі швидкістю понад 150 кадрів за секунду. Це означає, що ресурсів достатньо для обробки потокового відео в режимі реального часу із затримкою середньо менше 25 мілісекунд. Крім того, YOLO забезпечує точність більш ніж удвічі вище, ніж у інших систем реального часу.

По-друге, YOLO глобально думає про зображення, роблячи прогнози. На відміну від методу ковзного вікна та методу регіональних пропозицій, YOLO бачить все зображення під час навчання та тестування, тому явно кодує контекстну інформацію про класи, а також їхній зовнішній вигляд. Швидкий R-CNN, найкращий метод виявлення [27], неправильно розуміє фонові плями на

зображенні об'єктів, тому що не бачить більшого контексту. На YOLO припадає менше половини фонових помилок, порівняно з Fast R-CNN.

По-третє, YOLO вивчає загальні уявлення об'єктів. Під час навчання на природних зображеннях та тестуванні на ілюстраціях YOLO значно перевершує найкращі методи виявлення, такі як DPM та R-CNN. Оскільки YOLO є дуже загальним, ймовірність того, що він зламається під час використання нових доменів або несподіваних вхідних даних, менша.

YOLO завжди відстає від найпередовіших систем виявлення точності. Хоча може швидко ідентифікувати об'єкти на зображеннях, деякі об'єкти, особливо маленькі, виявити складно.

2.2 Огляд методу виявлення

Об'єднання окремих компонентів виявлення об'єктів у єдину нейронну мережу. Мережа YOLO використовує функції всього зображення для надання кожної рамки, що обмежує. Він також передбачає всі обмежувальні рамки для всіх класів зображення одночасно. Це означає, що мережа глобально говорить про повне зображення та про всі об'єкти на зображенні. Дизайн YOLO забезпечує наскрізне навчання та швидкість у реальному часі, зберігаючи при цьому високу середню точність.

Система розбиває вхідне зображення (рис. 2.2) на сітку S × S. Якщо центр об’єкта потрапляє в клітинку сітки, ця клітинка сітки відповідає за виявлення цього об'єкта.

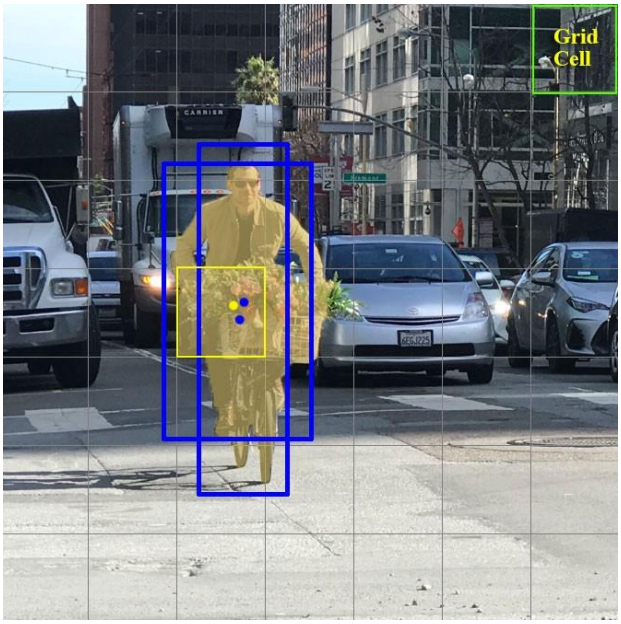


Рисунок 2.2 – Розбиття зображення

Кожна комірка сітки має на увазі *B* обмежуючих рамок і рівень впевненості для цієї рамки. Ці показники впевненості відображають, як впевнена модель полягає в тому, що рамка містить об'єкт і також наскільки точно віна думає, що рамка прогнозує.

$$\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} \tag{2.1}$$

Якщо об'єкт не існує в цьому осередку, показники довіри повинні дорівнювати нулю. Інакше ми хочемо, щоб показник впевненості (2.1) дорівнював перетину союзу (IOU) (рис. 2.3) між наданим полем та основною істиною. Кожна обмежувальна рамка складається з 5 передбачень: x, y, w, h і впевненість. Координати (x, y) представляють центр поля відносно меж осередку сітки. Ширина і висота прогнозуються відносно всього зображення. Нарешті,

прогноз довіри являє собою IOU між прогнозованою рамкою і будь-яким основним рамками істини.

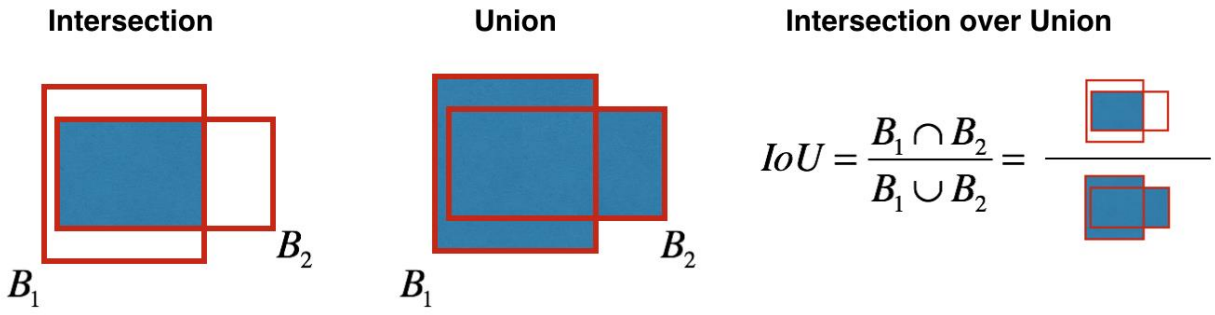


Рисунок 2.3 – Прогноз довіри IOU

Кожна клітина сітки також прогнозує C умовних ймовірностей класу, Pr(Class | Об'єкт). Ці ймовірності обумовлені коміркою сітки, що містить об’єкт. Прогнозується лише один набір ймовірностей класів на клітинку сітки, незалежно від кількість рамок B.

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (2.2)$$

Під час тестування ми множимо умовні ймовірності класів і прогнози довіри окремої рамки, що дає нам бали впевненості для кожного класу рамки. Ці бали кодують обидві ймовірності цього класу з’являється в рамці та наскільки добре передбачена рамка відповідає об'єкту.

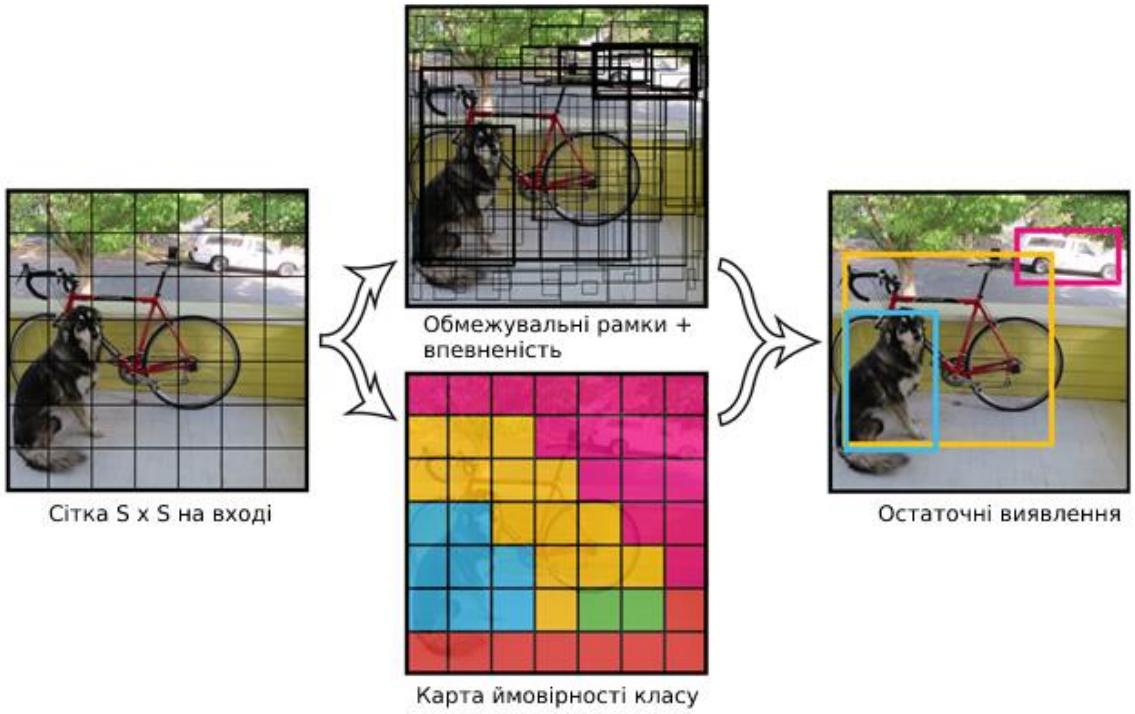


Рисунок 2.4 – Модель

Отже YOLO система моделює(рис. 2.4) виявлення як проблему регресії. Вона ділить зображення на сітку S × S і для кожного клітинка сітки передбачає B обмежувальні рамки, впевненість для цих квадратів, і ймовірності класу C. Ці передбачення закодовані як S × S × (B ∗ 5 + C) тензор.

2.2.1 Архітектура мережі

Архітектура YOLO натхнена GoogLeNet модель для класифікації зображень [28]. У цій мережі 24 згорткові шари, за якими слідують 2 повністю пов’язані шари. Замість початкових модулів, які використовує GoogLeNet, YOLO просто використовуйте редукційні шари 1 × 1, а потім згорткові шари 3 × 3, подібно до Lin et al [29]. Повна мережа є показано на малюнку 3.

Також існує швидка версія YOLO, розроблена для того, щоб розширити межі швидкого виявлення об’єктів. Fast YOLO використовує нейронну мережу з

меншою кількістю згорткових шарів (9 замість 24) і меншою кількістю фільтрів у цих шарах. Крім розміру мережі, всі параметри навчання та тестування однакові між YOLO та Fast YOLO.

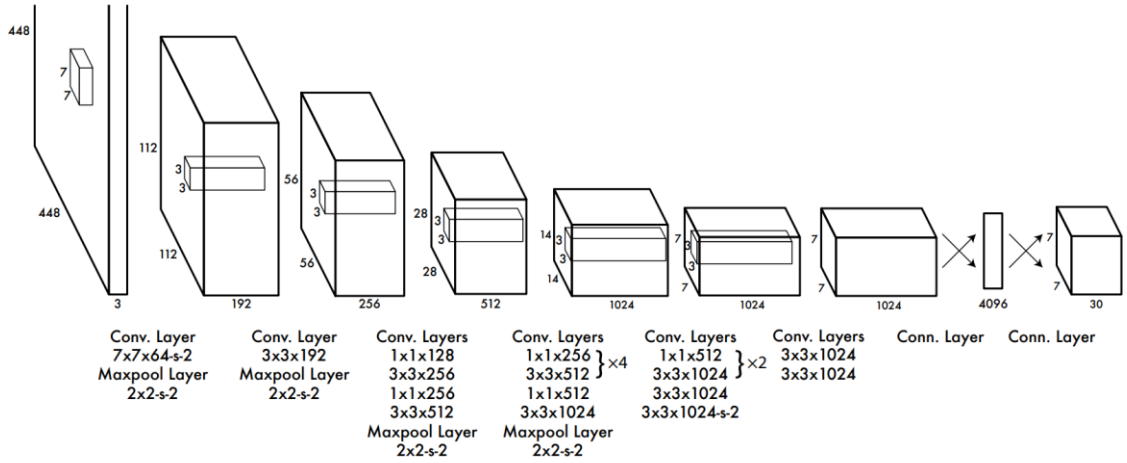


Рисунок 2.5 – Архітектура мережі

YOLO мережа виявлення має 24 згорткових шари, за якими слідують 2 повністю пов’язані шари. Чергування згорткових шарів розміром 1 × 1 зменшує простір елементів порівняно з попередніми шарами. Попередньо навчаємо згорткові шари задачі класифікації ImageNet на половині роздільної здатності (вхідне зображення 224 × 224), а потім подвоюються роздільну здатність для виявлення.

Початкові згорткові шари мережі витягують ознаки із зображення, тоді як повністю зв’язані шари прогнозують вихідні ймовірності та координати.

2.2.2 Механізм тренування та втрат

Функція втрат по-різному обробляє різні рамками. Кожна просторова комірка у вихідних шарах мережі передбачає кілька блоків (3 в YOLO-V3, 5 у попередніх версіях) — усі центровані в цій комірці за допомогою механізму, який називається прив’язками. Функція втрата YOLO для кожного прогнозу боксу складається з таких термінів:

1. Втрата координат — через передбачення рамки, яка не повністю покриває об’єкт;
2. Втрата об'єктності — через неправильний прогноз IoU об'єкта box-object;
3. Втрата класифікації — через відхилення від прогнозування «1» для правильних класів і «0» для всіх інших класів для об’єкта в цьому полі;
4. Особлива втрата, яку ми розглянемо в двох розділах нижче.

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(x_i - x_i'\right)^2 + \left(y_i - y_i'\right)^2 \right] \\
&+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{w_i'}\right)^2 + \left(\sqrt{h_i} - \sqrt{h_i'}\right)^2 \right] \\
&+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - C_i'\right)^2 \\
&+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - C_i'\right)^2 \\
&+\sum_{i=0}^{S^2} 1_{i}^{noobj} \sum_{c \in classes} \left(p_i(c) - p_i'(c)\right)^2
\end{aligned} \tag{2.3}$$

Функція втрати YOLO (2.3) лямбда - це коефіцієнти втрат. Перші 3 рядки – це втрата, спричинена «найкращими рамками» (рамками, які найкраще захоплюють об’єкти GT у кожній просторовій комірці), а 4-й – через блоки, які не захоплюють об’єкти. У YOLO V2 і V3 прямі прогнози ширини і висоти, а також квадратний корінь були замінені передбаченням залишкового масштабу, щоб зробити аргумент втрати пропорційним відносно відносної, а не абсолютної помилки масштабу.

Якість передбачення рамки вимірюється його IoU (Перетин над об'єднанням) з об'єктом, який він намагається передбачити (точніше — з його базовим рамками істинності). Значення IoU коливаються від 0 (рамка повністю пропускає об’єкт) до 1,0 (ідеальна підгонка).

Для кожної просторової комірки, для кожного передбачення блоку з центром у цій комірці, функція втрат знаходить коробку з найкращим IoU з об’єктом, центрованим у цій комірці. Цей механізм відмінності між найкращими боксами та всіма іншими рамками лежить в основі втрати YOLO.

Найкращі рамки та лише вони зазнають втрати координат (через не зовсім ідеальне прилягання до об’єкта) та втрати класифікації (через помилки класифікації). Це штовхає параметри мережі, пов’язані з цими рамками, для покращення масштабу та розташування рамки, а також класифікації. Ці рамки також призводять до втрати довіри, що буде негайно пояснено. Усі інші рамки призводять лише до втрати довіри. Зауважте, що функція втрат штрафує помилку класифікації лише в тому випадку, якщо в цій комірці сітки присутній об’єкт (звідси ймовірність умовного класу, про яку йшлося раніше). Він також штрафує за помилку координат обмежувальної рамки лише в тому випадку, якщо цей прогноз «відповідає» за основне поле істини (тобто має найвищий IOU серед будь-яких прогнозів у цій комірці сітки).

Щоб уникнути переобладнання, використовується виключення та великі дані збільшення. Випадає шар із швидкістю = 0.5 після першого зв'язаний шар запобігає коадаптації між шарами [30]. Для збільшення даних ми вводимо випадкове масштабування і переклади до 25% розміру оригінального зображення. ми також випадковим чином регулюйте експозицію та насиченість зображення з коефіцієнтом до 1.5 у колірному просторі HSV.

2.3 Порівняння систем виявлення

Виявлення об'єктів – основна проблема комп'ютерного зору. Конвеєри виявлення зазвичай починаються з отримання надійного набору функцій з вхідних зображень (Haar [31], SIFT [32], HOG [33]). Потім класифікатори [34] або локалізатори [35] використовуються для ідентифікації об'єкти в просторі ознак. Ці класифікатори або локалізатори запускаються або у вигляді ковзаючих вікон по всьому зображенню, або на деякій підмножині областей зображення [36].

Порівнюючи систему виявлення YOLO з кількома найпопулярнішими системами виявлення, мають ключові схожості та відмінності.

Моделі деталей, що деформуються (DPM). Моделі деталей, що деформуються використовують підхід ковзного вікна для виявлення об'єктів [37]. DPM використовує розрізнений конвеєр для вилучення статичних функцій, класифікувати регіони, передбачити обмежувальні рамки для високої оцінки регіони тощо. Система замінює всі ці розрізнені частини з єдиною згортковою нейронною мережею. Мережа одночасно виконує виділення ознак, передбачення обмежувальної рамки, немаксимальне здавлення та контекстне міркування. Замість статичних функцій мережа тренує функції і оптимізує їх для завдання виявлення. Наші уніфікована архітектура веде до швидшої та точнішої моделі ніж DPM.

R-CNN. R-CNN та його варіанти використовують пропозиції регіонів замість розсувних вікон для пошуку об’єктів на зображеннях. Вибірковий пошук [38] генерує потенційні обмежувальні рамки, згортка мережа витягує ознаки, SVM оцінює рамки, лінійна модель налаштовує обмежувальні рамки, а немаксимальне здавлення усуває повторні виявлення. Кожен етап цього складного конвеєру повинен бути точно налаштований самостійно і отримана система дуже повільна, займаючи більше 40 секунд на зображення під час тестування [27].

YOLO має певну схожість з R-CNN. Кожна сітка клітинка пропонує потенційні обмежувальні рамки та оцінює їх рамки з використанням згорткових ознак. Проте наша система накладає просторові обмеження на пропозиції осередку сітки, які допомагає пом’якшити багаторазове виявлення одного і того ж об’єкта. YOLO система також пропонує набагато менше обмежувальних квадратів, лише 98 за зображення в порівнянні з приблизно 2000 із вибіркового пошуку. Нарешті, наша система поєднує ці окремі компоненти в єдину, спільно оптимізовану модель.

Інші швидкі детектори. Fast та Faster R-CNN сфокусовані пришвидшити структуру R-CNN шляхом спільного використання обчислень і використання нейронних мереж для пропозиції регіонів вибіркового пошуку [29, 27]. Хоча вони пропонують швидкість і покращення точності в порівнянні з R-CNN, обидва все ще не досягнуті продуктивності в режимі реального часу.

Багато дослідницьких зусиль зосереджені на прискоренні DPM конвеєру [39]. Вони прискорюють обчислення HOG, використовуючи каскади, та передавати основні обчислення на графічні процесори. Однак, лише 30 Гц DPM насправді працює в режимі реального часу.

Замість того, щоб намагатися оптимізувати окремі компоненти великий конвеєр виявлення, YOLO викидає конвеєр повністю та спрощує дизайн проекту. Можуть бути детектори для окремих класів, таких як обличчя або люди дуже оптимізовані, оскільки їм доводиться мати справу з набагато менше варіація. YOLO - це детектор загального призначення вчиться виявляти різноманітні предмети одночасно.

Deep MultiBox. На відміну від R-CNN, навчає згорткову нейронну мережу для прогнозування регіонів інтересу [40] замість використання вибіркового пошуку. MultiBox також може виконувати виявлення одного об’єкта, замінюючи впевненість передбачення з прогнозуванням одного класу. Однак MultiBox не може виконувати загальне виявлення об’єктів і все ще є справедливим фрагмент у більшому конвеєрі виявлення, що вимагає подальшої класифікації латок зображення. І YOLO, і MultiBox використовують згортковість для прогнозування обмежувальних рамок у зображенні, але YOLO — це повна система виявлення.

OverFeat. Навчіть згорткову нейронну мережу для виконання локалізації та адаптуйте цей локалізатор для виконання виявлення [35]. OverFeat ефективно виконує виявлення ковзаючих вікон, але це все ще розрізнена система. OverFeat оптимізує для локалізації, а не для виявлення. Як і DPM, локалізатор бачить локальну інформацію лише тоді, коли зробити прогноз. OverFeat не може

міркувати про глобальне контекст і, отже, вимагає значної постобробки для отримання когерентних виявлення.

MultiGrasp. Наша робота за дизайном схожа на роботу виявлення захоплення [41]. Наш сітковий підхід до Прогноз обмежувальної рамки заснований на системі MultiGrasp для регресії до схоплень. Однак виявлення захоплення - це багато простіше завдання, ніж виявлення об'єктів. Потрібно лише MultiGrasp щоб передбачити єдину область для охоплення для зображення, що містить один об'єкт. При цьому не потрібно оцінювати розмір, розташування, або межі об’єкта або передбачити його клас, знайдіть лише а регіон, придатний для захоплення. YOLO передбачає обидва обмеження рамки та ймовірності класів для кількох об’єктів кількох класів на зображенні.

YOLO уніфікована модель для виявлення об’єктів, вона проста у побудові та її можна навчати безпосередньо на повних зображеннях. На відміну від підходів на основі класифікатора, YOLO навчається на функції втрат, яка безпосередньо відповідає продуктивності виявлення, і вся модель навчається спільно. Fast YOLO є найшвидшим універсальним детектором об'єктів у літературі, а YOLO просуває найсучасніші можливості виявлення об'єктів у реальному часі. YOLO також добре поширюється на нових доменах, що робить його ідеальним для програм, які покладаються на швидке та надійне виявлення об’єктів.

3 Інформаційне та програмне забезпечення

3.1 Тренувальний процес

Як і з будь-яким завданням глибокого навчання, першим найважливішим завданням є підготовка набору даних. Набір даних, який обробляється комп'ютером як єдине ціле – це датасет. Це означає, що набір даних містить багато окремих фрагментів даних, але його можна використовувати для навчання алгоритму з метою пошуку передбачуваних шаблонів у всьому наборі даних. Також зібрані дані мають бути однорідними та зрозумілими для машини, яка не бачить дані так само, як люди.

3.2 Збір датасету

Реальність показує, що робота з наборами даних є найбільш трудомісткою частиною будь-якого проекту машинного навчання, яка іноді займає до 70% часу в цілому. Крім того, для створення високоякісного набору даних потрібні досвідчені, навчені фахівці, які знають, що робити з фактичними даними, які можна зібрати.

Достатній обсяг даних дозволяє аналізувати тенденції та приховані закономірності та приймати рішення на основі створеного вами набору даних. Однак, хоча це може виглядати досить просто, робота з даними є складнішою, оскільки вона вимагає, перш за все, належної обробки даних, які ви маєте, від цілей використання набору даних до підготовки вихідних даних для того, щоб вони були насправді придатний до використання.

Існує декілька підходів для збору датасета:

1. Власноручні підбір. Метод включає в себе пошук цільових зображення (даних) та ручна анотація даних зображення.
2. Використання готових датасетів. Існує велика кількість готових датасетів. Це суттєво прискорює процес підбору, агрегувати цільові класи набагато зручніше.

3. Генерація синтетичних даних. Більш складний процес створення датасета, що включає пошук уявлень, налаштування оточення, налаштування матеріалів, освітленості, трансформацій. Його перевагою є створення будь-якої кількості навчальних даних з автоматичними анотаціями. Також цей процес включає аугментацію, за рахунок змін властивостей уявлень, що також підвищує якість датасета.

Рисунок 3.1 – Приклад датасету MS COCO

Для вирішення завдання розпізнавання було використано готовий датасет (рис. 3.1) MS COCO. Common Objects in Context (COCO) — це база даних, яка має на меті уможливити майбутні дослідження для виявлення об'єктів, сегментації екземплярів, підписів зображень та локалізації ключових точок людей.

Рисунок 3.2. Unity Perception, приклад синтетичних об’єктів

Також для більшої різноманітності було частково використано синтетичну генерацію. З цим завданням чудово справляється плагін Unity Perception. Плагін

Perception надає набір інструментів для створення великомасштабних наборів даних(рис. 3.2) для навчання та перевірки комп'ютерного зору. Наразі він зосереджується на кількох варіантах використання на основі камери і врешті-решт поширяться на інші форми датчиків і завдань машинного навчання.

3.2.1 Аугментування робочих даних

Аугументування або розширення зображення слід вибирати на основі проблеми домену, яка вирішується. Зокрема, аерофотодані часто мають ряд характеристик, які дозволяють виконати подібний вибір збільшення зображення. Повітряні дані – це будь-які дані, отримані зверху, дивлячись вниз. Зазвичай це включає знімки з дронів і супутників.

Препроцецссінг зображення до навчання, має кілька ключових переваг.

1. Збільшується відтворюваність моделі. Можна виявити, що ваша модель працює краще на яскравих зображеннях, ніж на темних зображеннях, тому вам слід зібрати більше навчальних даних при слабкому освітленні.
2. Зменшується час навчання. Аугментації – це операції з обмеженим процесором. Коли тренування на своєму графічному процесорі та проводите розширення на льоту, графічний процесор часто чекає, поки процесор надасть розширені дані в кожну епоху.
3. Знижуються витрати на навчання. Оскільки аугментації є операціями з обмеженим процесором, графічний процесор часто чекає, щоб отримати зображення для навчання. Це витрачений час в очікуванні.

Для єдиної структуризації та збільшення кількості робочих даних було використано сервіс Roboflow. Він безкоштовний, дозволяє структурувати різні дані до одного формату. Ще завдяки цьому сервісу ми можемо зробити аугументацію, а саме зробити дублікати робочих зображень. Сервіс підтримує 23 техніки аугментації (рис. 3.3) , наприклад: поворот, обрізання, масштабування, контроль яскравістю, віддзеркалювання, зашумлення та інші.

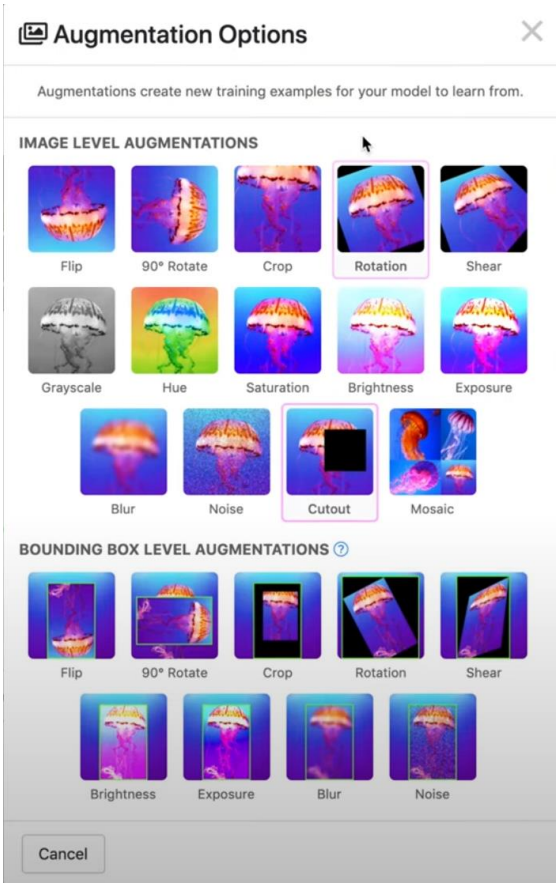


Рисунок 3.3 – Результати застування техніки аугментації

3.2.2 Розбиття тренувальних даних

Навчання вимагає, щоб алгоритм переглядав навчальні дані кілька разів, а це означає, що модель буде піддаватися одних і тих самих шаблонів, якщо вона працює над тим самим набором даних.

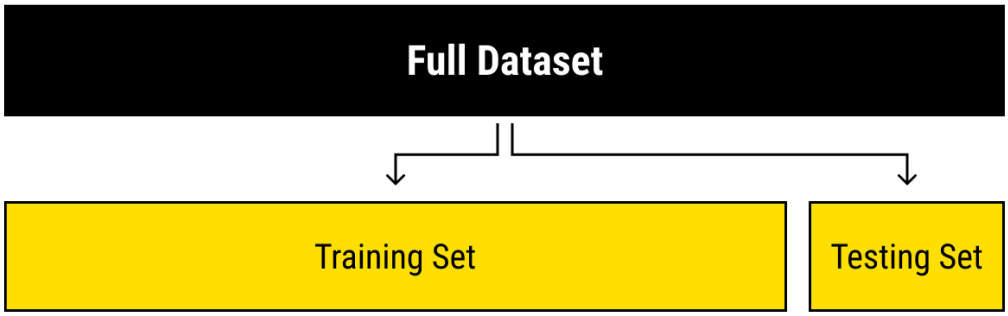


Рисунок 3.4 – Розбиття датасету

Щоб уникнути цього, потрібено мати розділений набір даних, який допоможе алгоритму бачити різні шаблони. Але в той же час ви не хочете залучати свій набір даних тестування (рис. 3.4) до закінчення навчання, оскільки він вам потрібен для різних цілей. Тобто тренувальний датасет ділиться:

1. Навчальний набір: це частина даних, на якій ми навчаємо модель. Залежно від обсягу даних, які ви маєте, ви можете випадковим чином вибрати від 70% до 90% даних для навчання.
2. Тестовий набір: це частина даних, на якій ми тестуємо нашу модель. Зазвичай це 10-30% даних. Жодне зображення не повинно бути частиною як навчального, так і тестового набору.

3.2.3 Фреймворк Darknet

Для тренування використовуємо фреймворк Darknet від Джозефа Редмона. Це структура глибокого навчання, написана на C. Він дозволяє тренувати та використовувати YOLO через термінал, досить легко. Darknet швидкий, простий в установці та підтримує обчислення як на CPU, так і на GPU.

3.2.3.1 Анотація даних

Після того надали доступ до файлів міток з анотаціями в папці labels. Кожен запис рядка у файлі етикетки представляє одну обмежувальну рамку на зображенні та містить таку інформацію про поле:

```
<object-class-id> <center-x> <center-y> <width> <height>
```

Перше поле object-class-id є цілим числом, що представляє клас об’єкта. Він коливається від 0 до (кількість класів – 1). У нашому поточному випадку, оскільки у нас є лише один клас сніговика, він завжди встановлюється на 0.

Другий і третій запис, center-x і center-y відповідно є координатами x і y центру обмежуючого прямокутника, нормованими (поділеними) на ширину та висоту зображення відповідно.

Четвертий і п'ятий запис, ширина і висота відповідно є шириною і висотою обмежуючого прямокутника, знову нормованими (поділеними) на ширину і висоту зображення відповідно. Розглянемо приклад із такими позначеннями:

x – координата x (у пікселях) центру обмежувальної рамки

y – координата y (у пікселях) центру обмежувальної рамки

w – ширина (у пікселях) рамки

h – висота (у пікселях) рамки

W – ширина (у пікселях) всього зображення

H – висота (у пікселях) всього зображення

Потім ми обчислюємо значення анотації у файлах етикеток наступним чином:

центр-x = x / W

центр-y = y / H

ширина = w / W

висота = h / H

Наведені вище чотири записи є значеннями з плаваючою комою від 0 до 1.

3.2.3.2 Використання пре-тренованої моделі

Для більш успішного та продуктивного тренування власного детектору об’єктів, добре використовувати існуючі моделі, навчені на дуже великих наборах даних, навіть якщо великий набір даних може не містити об’єкт, який ми намагаєтеся виявити. Цей процес називається трансферним навчанням.

Замість того, щоб вчитися з нуля, ми використовуємо попередньо навчену модель, яка містить згорткові ваги, навчені на YOLO пре-тренувальні ваги . Використовуючи ці вагові коефіцієнти як стартові, наша мережа може навчатися швидше.

3.2.3.3 Використання конфігурацій

Поряд з файлами darknet.data і classes.names, YOLO також потребує файлу конфігурації. Він також включений до бази коду. Він заснований на

демонстраційному файлі конфігурації yolov-pre.cfg (поставляється з кодом darknet), який використовувався для навчання набору даних. Усі важливі параметри навчання зберігаються в цьому конфігураційному файлі. Зазначимо такі основні параметри:

А) Пакетний гіперпараметр. Параметр пакета вказує розмір пакету, який використовується під час навчання.

Наш навчальний набір містить кілька сотень зображень, але нерідко можна тренуватися на мільйонах зображень. Процес навчання включає ітераційне оновлення ваг нейронної мережі на основі того, скільки помилок вона робить у наборі навчальних даних.

Недоцільно (і непотрібно) використовувати відразу всі зображення в наборі для оновлення ваг. Отже, за одну ітерацію використовується невелика підмножина зображень, і ця підмножина називається розміром пакету.

Коли розмір пакету встановлено на 32, це означає, що 32 зображень використовуються за одну ітерацію для оновлення параметрів нейронної мережі.

Б) Параметр конфігурації підрозділів. Найчастіше для пакетного розміру 32 для навчання нейронної мережі може не вистачати пам'яті. Щоб вирішити проблему з переповненням використовують параметр subdivision, який дозволяє обробляти частину від розміру пакету за один раз на графічному процесорі. Це зробити тренування трохи довшим, але це необхідність. Під час тестування для пакету та підрозділу встановлено значення 1.

В) Ширина, висота, канали. Вхідні навчальні зображення спочатку змінюються до ширини х висоти перед тренуванням. Тут ми використовуємо значення за замовчуванням 416×416. Результати можуть покращитися, якщо ми збільшимо його до 608×608, але тренування також займе більше часу. channels=3 означає, що ми будемо обробляти 3-канальні вхідні зображення RGB.

Г) Імпульс і затухання. Файл конфігурації містить кілька параметрів, які контролюють, як оновлюється вага.

У попередньому розділі ми згадували, як ваги нейронної мережі оновлюються на основі невеликої партії зображень, а не всього набору даних. З цієї причини оновлення ваги досить сильно коливаються. Ось чому імпульс параметра використовується для штрафування великих змін ваги між ітераціями.

Типова нейронна мережа має мільйони ваг, і тому вони можуть легко переповнювати будь-які навчальні дані. Переобладнання просто означає, що воно буде дуже добре працювати з навчальними даними і погано з тестовими даними. Це схоже на те, що нейронна мережа запам’ятала відповідь на всі зображення в навчальному наборі, але насправді не вивчила основну концепцію. Одним із способів пом’якшити цю проблему є покарання за велике значення ваг. Параметр затухання контролює цей штрафний термін. Значення за замовчуванням працює нормально, але ви можете змінити це, якщо помітите переобладнання.

Д) Швидкість навчання, кроки, масштаб, запис. Параметр швидкість навчання контролює, наскільки агресивно ми повинні навчатися на основі поточної партії даних. Зазвичай це число від 0,01 до 0,0001.

На початку процесу навчання ми починаємо з нульовою інформацією, тому швидкість навчання має бути високою. Але оскільки нейронна мережа бачить багато даних, ваги потрібно змінювати менш агресивно. Іншими словами, швидкість навчання потрібно з часом знижувати. У файлі конфігурації це зниження швидкості навчання досягається, якщо спочатку вказати, що наша політика зниження швидкості навчання є кроковою. У наведеному вище прикладі швидкість навчання почнеться з 0,001 і залишиться постійною протягом 3800 ітерацій, а потім вона помножиться на шкали, щоб отримати нову швидкість навчання. Також є можливість вказати кілька кроків і масштабів.

Як було описано вище, швидкість навчання повинна бути високою на початку і низькою пізніше. Хоча це твердження в значній мірі вірне, емпірично було виявлено, що швидкість навчання має тенденцію збільшуватися, якщо ми маємо нижчу швидкість навчання протягом короткого періоду часу на самому

початку. Це контролюється параметром burn_in. Іноді цей період вигорання також називають періодом розігріву.

Г) Аугементація даних. Фреймворк yolo вміє аугементувати за 4 параметрами, це доповнює датасет, та дозволяє зменшити датасет

Ґ) Кількість ітерацій. Останнім параметром буде кількість ітерацій, потрібних для виконування навчального процесу.

Для багатокласових детекторів об’єктів кількість max_batches є більшою, тобто нам потрібно запустити більшу кількість пакетів (наприклад, у yolo.cfg). Для детектора об’єктів n-класу доцільно запустити навчання для щонайменше 2000*n пакетів.

3.2.4 Тренування

Після того як було створено датасет і створено конфігураційний файл, можна запустити процес навчання. Завдяки виводу в консоль можна спостерігати за процесом навчання (рис.3.5).

```
395: 0.658902, 0.738367 avg, 0.000951 rate, 5.031825 seconds, 25280 images
396: 0.418165, 0.706346 avg, 0.000961 rate, 4.935657 seconds, 25344 images
397: 0.524274, 0.688139 avg, 0.000970 rate, 5.121099 seconds, 25408 images
398: 0.600312, 0.679356 avg, 0.000980 rate, 5.148844 seconds, 25472 images
399: 0.666177, 0.678039 avg, 0.000990 rate, 5.495806 seconds, 25536 images
400: 0.540531, 0.664288 avg, 0.001000 rate, 5.182177 seconds, 25600 images
401: 0.603434, 0.658202 avg, 0.001000 rate, 5.840096 seconds, 25664 images
402: 0.480128, 0.640395 avg, 0.001000 rate, 5.756050 seconds, 25728 images
403: 0.816233, 0.657979 avg, 0.001000 rate, 6.160515 seconds, 25792 images
404: 0.630400, 0.655221 avg, 0.001000 rate, 6.171423 seconds, 25856 images
405: 0.434452, 0.633144 avg, 0.001000 rate, 6.168081 seconds, 25920 images
```

Рисунок 3.5 – Логіювання процесу тренування

Як бачимо, швидкість навчання поступово збільшується від 0 до 0,001 до 400-ї партії. Він залишиться там до 3800-ї партії, коли знову зміниться на 0,0001.

Під час навчання файл журналу містить втрати в кожній партії. Можна заперечити, щоб припинити тренування після того, як програш опустився нижче певного порогу.

Зазвичай достатньо 2000 ітерацій для кожного класу (об'єкта), але не менше кількості навчальних зображень та не менше 6000 ітерацій загалом. Але для більш точного визначення, коли слід припинити тренування, є такі:

1) Під час навчання ви можете побачити різні індикатори помилок.
   Зупинитись слід тоді, коли індикатор 0.XXXXXXX avg перестане помітно зменшуватися.
2) Коли навчання зупинено, необхідно взяти кілька останніх вагових файлів з директорії і вибрати найкращий.
   Перенавчання - ситуація в якій модель працюватиме тільки на даних із тренувального датасету.

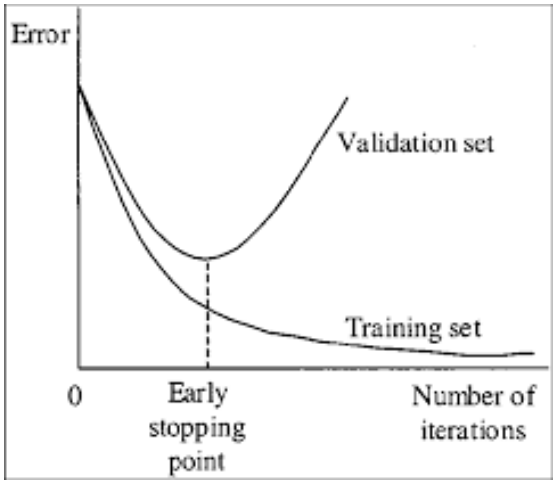


Рисунок 3.6 – Процес закінчення навчання

Порівняются останні рядки виводу для кожного файлу терезів(7000, 8000, 9000): Вибирати потрібно по найбільшому показнику mAP. (mean average precision - середня точність) або IoU (intersect over union - перетин по об'єднанні).

Також можна вивести графік mAP (red-line) поверх графіка помилок. Показник mAP буде розрахований на 4 епохи, використовуючи valid=valid.txt файл, який вказаний у файлі obj.data (1 Epoch = images_in_train_txt / batch ітерацій).

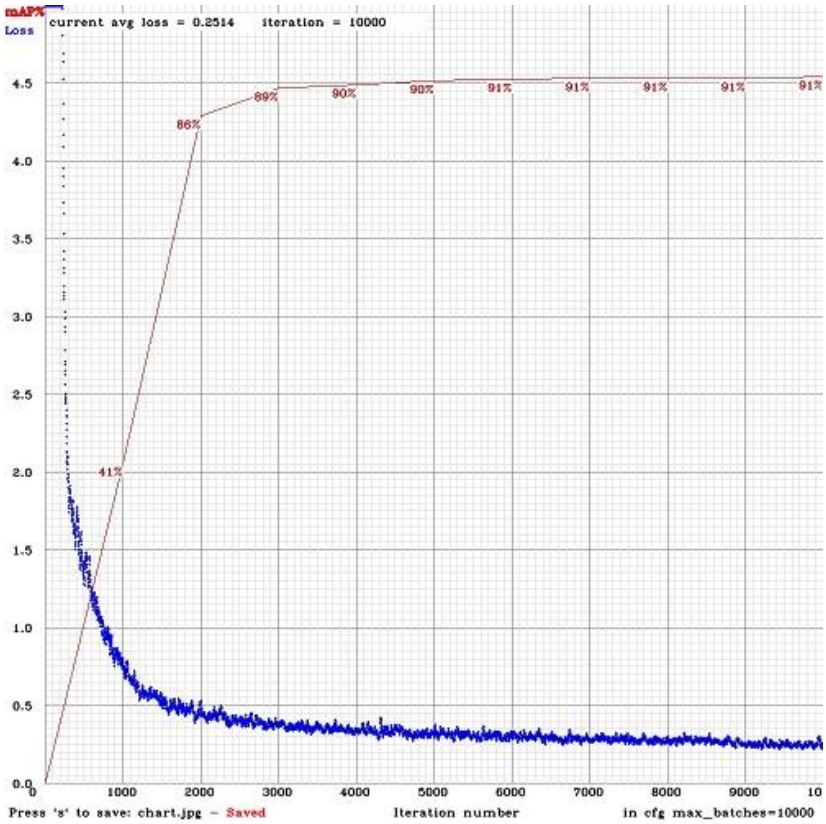


Рисунок 3.7 – Графік зміни значень критеріїв якості навчання

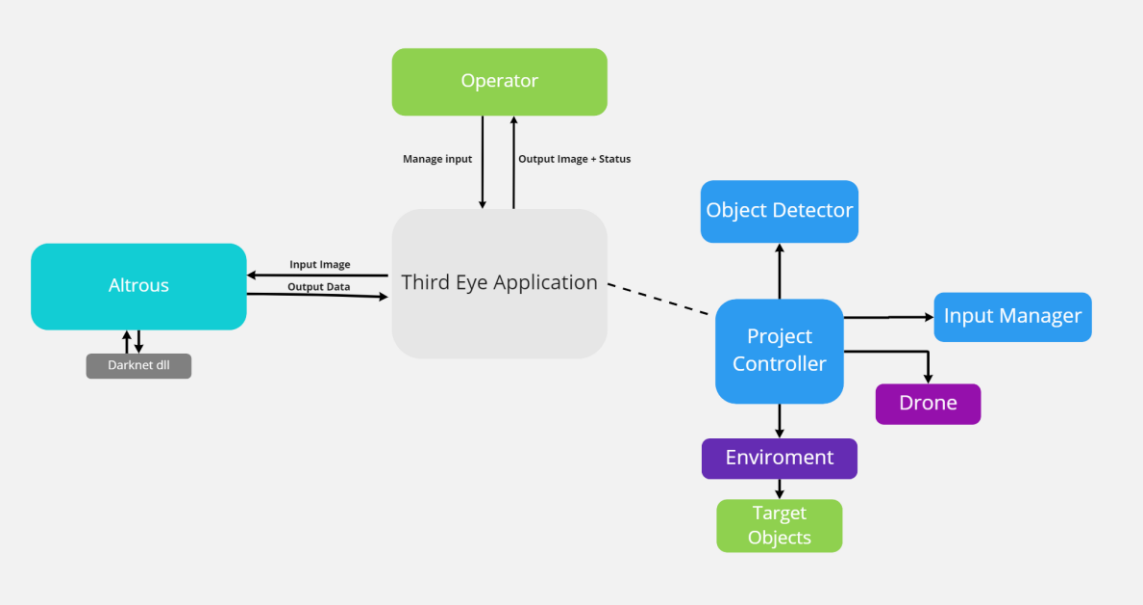
IoU (intersection over union - перетин по об'єднанню) - середнє значення IoU об'єктів та розпізнавання для певного порога = 0.24.

mAP (mean average precision) – середнє значення average precisions для кожного класу, де average precision – це середнє значення 11 точок на кривій PR для всіх можливих порогів (кожна ймовірність розпізнавання об'єкта) для однакового класу (Precision-Recall з точки зору PascalVOC , де Precision розраховується як : TP/(TP+FP), а Recall як TP/(TP+FN))

3.3 Програмна реалізація

Для вирішення завдання було розроблено просту архітектуру (діаграма 3.7) проекту. Можна виділити 3 ключові сутності:

- Оператор. Це людина яка взаємодіє з віртуальним оточенням через дрон. Людина вводить керуючі команди та отримує зображення від камери дрону з його статусом. Поверх вхідного зображення відображаються знайдені об'єкти.
- ThirdEyeApplication. Центр програми, що включає обробкою вхідних даних від оператора, симуляцією дрону і оточення, розрахунку користувальницької інформації.
- Altrous. Окремий сервіс, який дозволяє відокремити логіку нейромережі. Це дає більше гнучкості, завдяки якій можна запустити сервіс більш продуктивної обчислювальної машини.



Діаграма 3.7 – Архітектура додатка

3.3.1 Вибір програмного середовища

Як середовище для розробки програми було обрано програмний рушій Unity 2019.4.23f1. Для редагування та компілювання коду – Visual Studio 2019.

На сьогоднішній день існує безліч середовищ розробки програмного забезпечення. Вибрані середовища виявилися найбільш прогресивними та ефективними у розробці. У Unity є інструменти, з якими дуже зручно працювати, не витрачаючи багато сил на освоєння.

На відміну від багатьох ігрових двигунів, Unity має три основні переваги: наявність візуального середовища розробки, кросплатформова підтримка і модульна система компонентів. До першого чинника ставляться як засоби візуального моделювання, а й інтегроване середовище, спрямовану підвищення продуктивність праці розробників. Кросплатформеність забезпечується не тільки місцями розгортання (установка на персональний комп'ютер, мобільний пристрій, консоль тощо), але й наявністю засобів розробки (інтегроване середовище може використовуватись на Windows та Mac OS). Модульна система компонентів, з яких будуються ігрові об'єкти, коли останні є наборами функціональних елементів.

3.3.2 Реалізація Дрона

Також можна особливо виділити клас Quadcopter Controller (рисунок 3.8) Завдяки йому користувач може взаємодіяти з ігровим оточенням. Цей клас можна розділити на 3 основні частини:

- InputModule. Класс зчитування вхідних керуючих команд користувача.
- Motor. Клас імітує роботу двигуна, створює фізичну підйомну, що скручує силу.
- ComputerModule. Цей клас обчислює підйомну силу (PID) на основі введення користувача і поточного стану транспортного засобу.

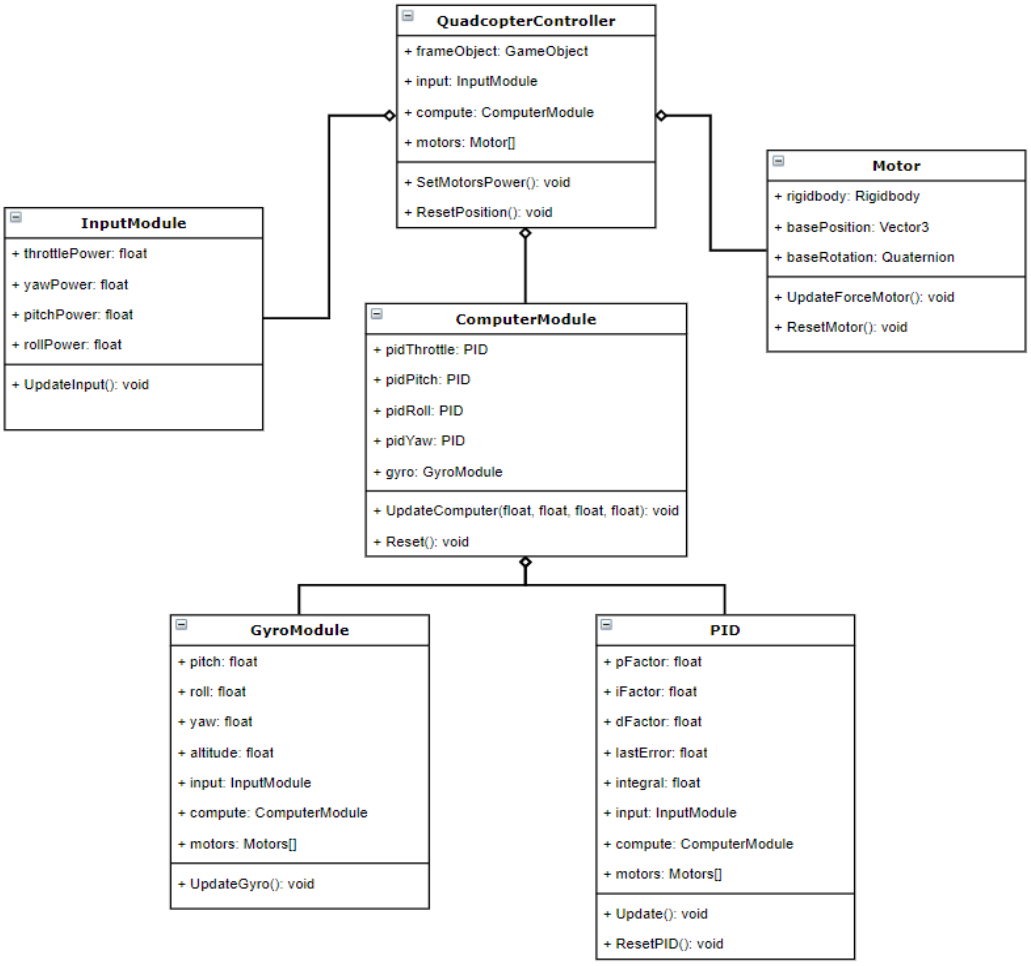


Рисунок 3.8 – Схема роботи віртуального мультикоптеру

3.3.3 Сервіс Altrous

Сучасна система виявлення об’єктів у реальному часі для C# (Visual Studio). Цей проект підтримує процесор і графічний процесор, з GPU виявлення працює набагато швидше. Основною метою цього проекту є просте використання Yolo, цей пакет доступний на nuget, і вам потрібно встановити лише два пакунки, щоб почати виявлення. У фоновому режимі ми використовуємо версію darknet для Windows Yolo. Вхідні дані відправляємо шлях зображення або масив байтів до Yolo та отримаємо положення виявлених об’єктів. Наш проект призначений для повернення типу об'єкта та -позиції як даних, що підлягають обробці.

3.4 Тестування взаємодії оператору та дрону

Перед початком аналізу сформуємо очікуваний результат. Дрон повинен маркувати об'єкти для подальшого вивчення після команди оператора.

Як цільові об'єкти розпізнання використовуємо віртуальне місто з Інтернету. Наша модель навчена опізнати 80 класів (включаючи машин і людей) заснованих на реальних фотографіях.

Дизайн міста складається із спрощених форм та кольору, тіней це утворить розпізнавання. Об'єкти всередині міста не використовувалися для навчання моделі, дрон побачить їх уперше.

Рисунок 3.9 – Вигляд дрона

Після вильоту дрону (рисунок 3.9) він відразу починає визначати об'єкти перед собою. На рисунку 2, видно як він позначає об'єкти, намагається побудувати рамку та чисельний рівень впевненості. Нейромережа впоралась з розпізнаванням, більшість об’єктів вдається правильно класифікувати.

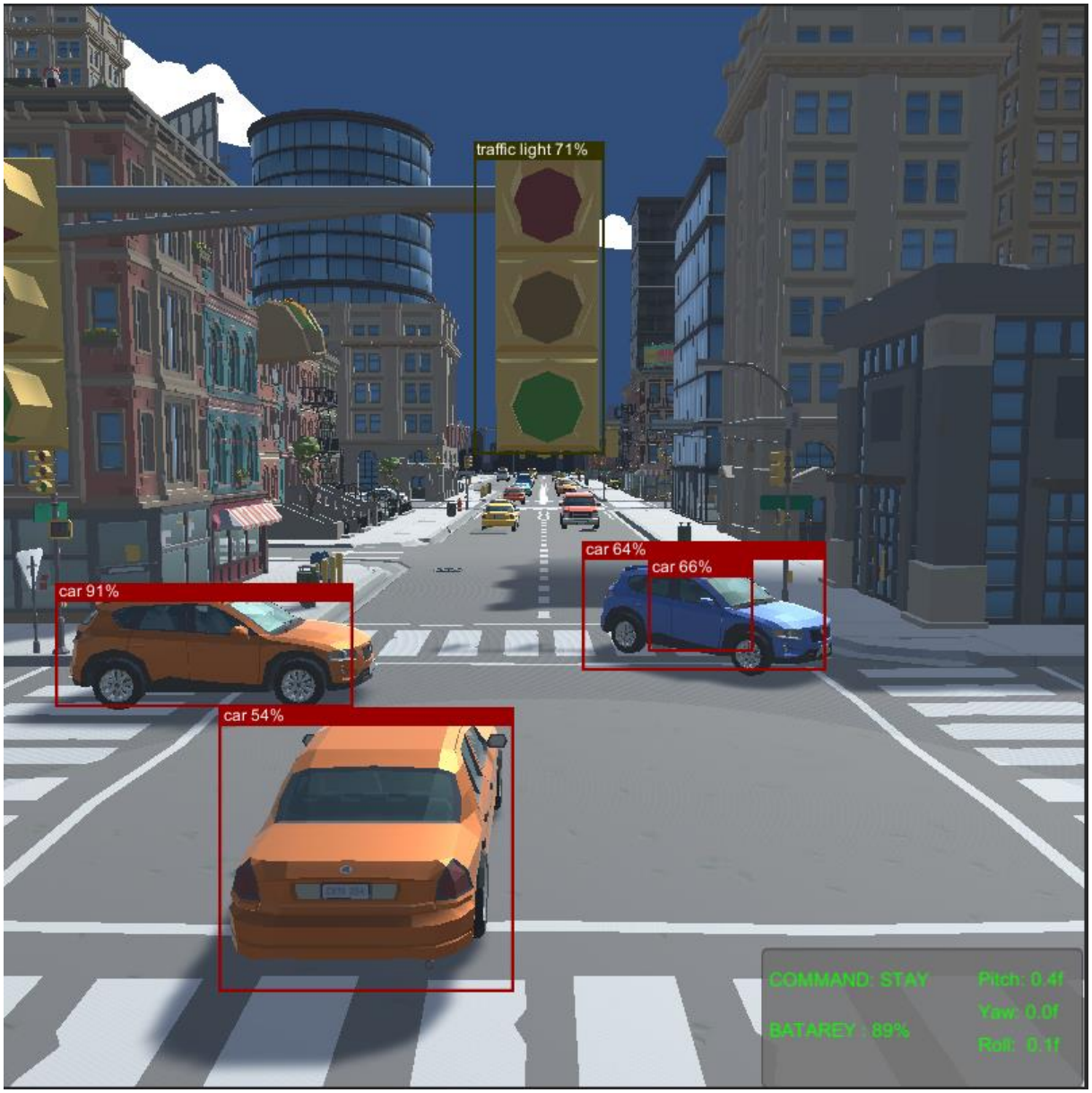


Рисунок 3.10 – Вигляд екрана оператора

З боку оператора, відаємо задачу дрону скоротити дистанцію з дальньою машиною. Вхідним даним буде клік(точка на 2д просторі) або червоний маркер яка входе в рамки упізнаній машині. Далі завдяки віртуальному лазерному дальноміру, розраховується відстань до цільового об’єкту. Дрон знає кінцеву точку, та виконує команду “Рушити”.

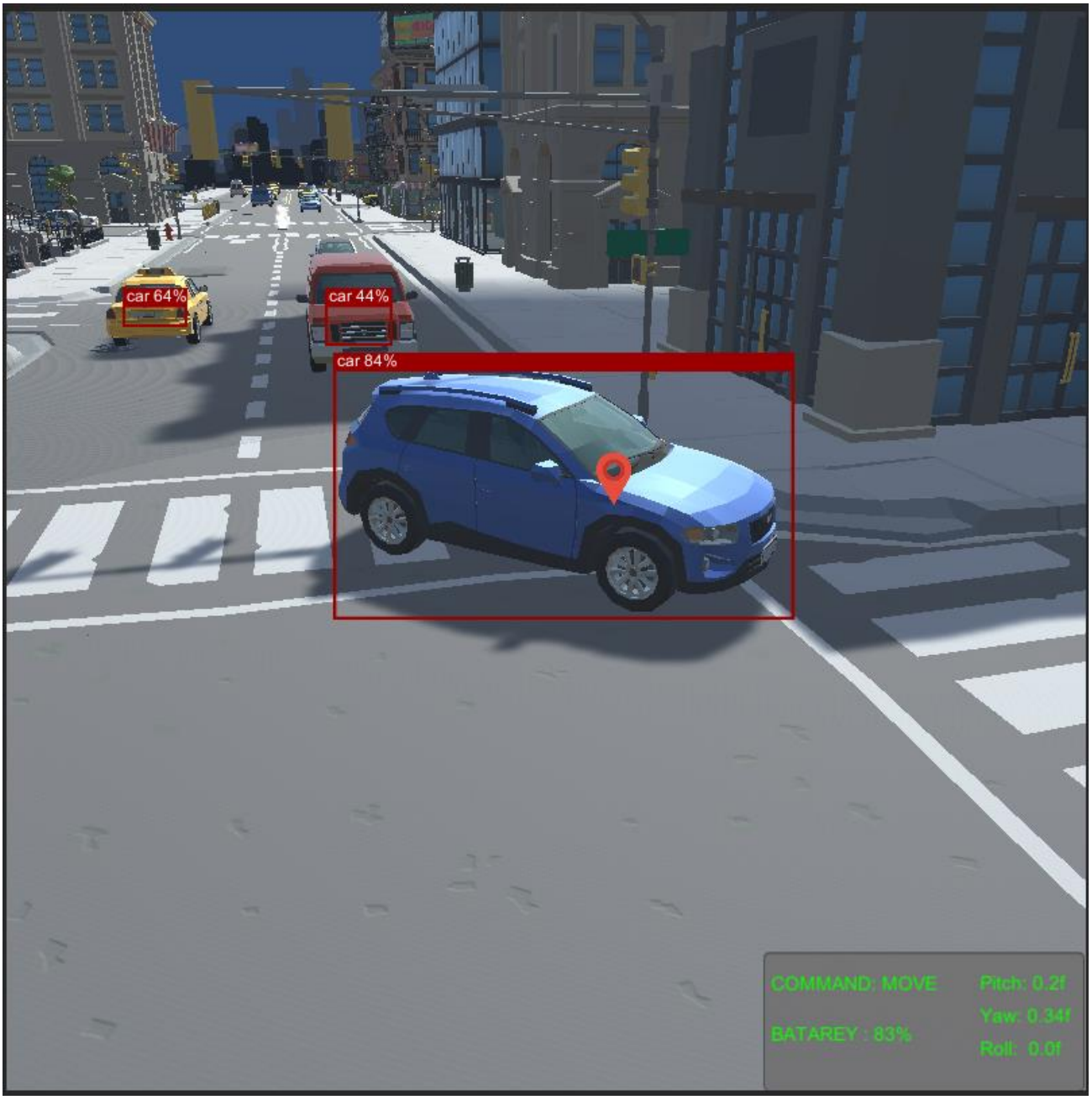


Рисунок 3.11 – Дрон зменшує дистанцію з ціллю

Як результат, ми отримали дієздатність, що відповідає прогнозованому сценарію. Так як результат відповідає прогнозованому, на мою думку, тест можна вважати вдалим.

ВИСНОВКИ

В кваліфікаційній магістерській роботі було проведено розробку інформаційної технології машино-людинного інтерфейсу бортової системи розпізнавання наземних об'єктів.

При цьому було виконано такі завдання виконати такі завдання:

1. Було детально розглянуто та порівняно різні методи та алгоритми розпізнавання.
2. Сформовано вхідний математичний опис бортової системи розпізнавання наземних об'єктів.
3. Проведено вибір типу та структури класифікатору зображень наземних об'єктів, що використовуються в машино-людинному інтерфейсі бортових систем розпізнавання.
4. Проведено вибір математичних моделей формування класифікаторів зображень наземних об'єктів та критеріїв оцінювання їх якості.
5. Проведено розробку та програмну реалізацію алгоритмів оптимізації функціональних параметрів бортової системи розпізнавання наземних об'єктів.
6. Проведено перевірку працездатності розробленого машино-людинного інтерфейсу бортової системи розпізнавання наземних об'єктів

СПИСОК ЛІТЕРАТУРИ

1. Харченко В.: “Методологія ситуаційного колективного управління пілотованими і безпілотними літальними апаратами в єдиному повітряному просторі”, 2017, 120 с
2. Reg Austin, Unmanned aircraft systems : UAVS тdesign, development and deployment / 2010 John Wiley & Sons Ltd
3. J. D. Gibson and M. Bhaskaranand, “Low-complexity video encodingfor UAV reconnaissance and surveillance,” pp. 1633-1638, 2011
4. T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair,I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, “Toward a fullyautonomous UAV”, page. 46-56, 2012
5. Ma, Y.; Wu, X.; Yu, G.; Xu, Y.; Wang, Y. Pedestrian detection and tracking from low-resolution unmanned aerial vehicle thermal imagery. Sensors 2016, 16, 446
6. Wu, Z.; Suresh, K.; Narayanan, P.; Xu, H.; Kwon, H.; Wang, Z. Delving into robust object detection from unmanned aerial vehicles: A deep nuisance disentanglement approach. 2019; page 1201–1210.
7. J. Engel, J. Sturm, and D. Cremers, “Scale-aware navigation of a low-cost quadrocopter with a monocular camera,” page 1646-1656, 2014
8. Професія: оператор безпілотних літальних апарати (БПЛА) – Режим доступу: https://proforientator.ru/publications/articles/professiya–operator–bespilotnykh–letatelnykh–apparatov–bpla.html
9. Fleetlights take streetlights to the skies – Режим доступу: https://newatlas.com/fleetlights-drone-streetlights/46499/
10. T. P. Breckon, A. Gaszczak, and J. Han, “Real-time people and vehicle detection from UAV imagery”, 2011

11. C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, “Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles,” in Proc. pp. 111-118, 2015
12. S. E. Barnes, T. P. Breckon, , M. L. Eichner, and K. Wahren, “Autonomous real-time vehicle detection from a medium-level UAV,” in Proc. 24th International Conference on Unmanned Air Vehicle Systems, pp. 29.1-29.9, 2009
13. F. S. Leira, T. A. Johansen, and T. I. Fossen, “Automatic detection, classification and tracking of objects in the ocean surface from UAVs using a thermal camera,” in Proc. pp. 1-10, 2015
14. T. Darrell, R. Girshick, J. Donahue, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” on Computer Vision and Pattern Recognition (CVPR), pp. 580-587, 2014
15. J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” in arXiv preprint arXiv:1411.4389, 2014
16. Large Scale Visual Recognition Challenge 2013 – Режим доступу: http://image-net.org/challenges LSVRC/2013/
17. Amazon is still struggling to make drone deliveries work – Режим доступу: https://www.theverge.com/2022/4/11/23020549/amazon-struggling-drone-deliveries-prime-air-bezos
18. Стратегічний ударний БПЛА: "Антонов" вирішили реанімувати амбітний проект – Режим доступу: https://defence-ua.com/news/strategichnij_udarnij_bpla_na_dp_antonov_virishili_reanimuvati_ambitnij_proekt_video-3670.html
19. Zhang, J.;Liang, X.; Li, Y.; Zhuo, L; Tian, Q. Small object detection in unmanned aerial vehicle images using feature fusion and scaling-based single shot detector with spatial context analysis. 2019, page 1758–1770

20. Zheng, Z.; Wang, P.; Liu, W.; Li, J.; Ye, R.; Ren, D. Distance-IoU loss: Faster and better learning for bounding box regression. 2013
21. Rasmussen, J.; "Information Processing and Human-Machine Interaction, An Approach to Cognitive Engineering"; 1986
22. White, A.D.; "The Human-Machine Partnership in UCAV Operations"; 17th Bristol International UAV Systems Conference; April 2002
23. Clough, B.T.; "Metrics, Schemetrics! How Do You Track a UAV's Autonomy?"; AIAA2002-3499; 1st AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference and Workshop; May 2002
24. Platts, J.T.; "Application of a Variable Autonomy Framework to the Control of Multiple Air-Launched UAVs"; Association for Unmanned Vehicle Systems International Conference, August 2002
25. S. Gidaris and N. Komodakis. Object detection via a multiregion & semantic segmentation-aware CNN model. CoRR, abs/1505.01749, 2015
26. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition (CVPR), pages 580–587, 2014
27. R. B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015
28. D. Mishkin. Models accuracy on imagenet 2012 val. https://github.com/BVLC/caffe/wiki/ Models-accuracy-on-ImageNet-2012-val. Accessed: 2015-10-2
29. S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. CoRR, abs/1504.06066, 2015
30. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012
31. C. P. Papageorgiou, M. Oren. A general framework for object detection. In Computer vision, 1998, pages 555–562, 1998

32. D. G. Lowe. Object recognition from local scale-invariant features. In Computer vision. pages 1150–1157, 1999

33. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, Computer Society Conference on, volume 1, pages 886–893, 2005

34. R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on, volume 1, pages I–900

35. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2009, page 304

36. S. Gould, T. Gao, and D. Koller. Region-based segmentation and object detection. In Advances in neural information processing systems, pages 655–663, 2009

37. P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models, 2010

38. J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. International journal of computer vision, 104(2):154–171, 2013.

39. T. Dean, M. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, J. Yagnik, et al. Fast, accurate detection of 100,000 object classes on a single machine. In Computer Vision and Pattern Recognition (CVPR), pages 1814–1821, 2013

40. D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In Computer Vision and Pattern Recognition (CVPR), pages 2155–2162, 2014

41. J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. CoRR, abs/1412.3128, 2014.

ДОДАТОК

ClientManager.cs

```
public class ClientManager : IDisposable
    {
        public event EventHandler<DetectionEventArgs> RaiseDetectionEvent;
        DetectionEventArgs detectionEventArgs; // re-use

        Channel channel;
        ClientWrapper client;
        Texture2D texture;
        YoloResult result; // re-use, reference

        Stopwatch timer;
        const int minInterval = 0; // throttle requests
        bool requestEnabled => timer.Elapsed.Milliseconds >= minInterval;

        public ClientManager(ref Texture2D texture)
        {
            channel = new Channel("127.0.0.1:50052", ChannelCredentials.Insecure);
            client = new ClientWrapper(new YoloService.YoloServiceClient(channel));

            this.texture = texture;
            result = new YoloResult();
            detectionEventArgs = new DetectionEventArgs(result);

            timer = new Stopwatch();
            timer.Start();
        }

        public void Dispose()
        {
            channel.ShutdownAsync();
        }

        public void Update()
        {
            if (client.IsIdle)
            {
                if (requestEnabled)
                {
                    timer.Restart();
                    result.Clear();
                    client.Detect(ImageConversion.EncodeToPNG(texture), result);
                }
            }
            else if (client.HasNewResponse)
            {
                UnityEngine.Debug.Log(string.Format("Detection time: {0}ms,
Roundtrip time: {1}ms",
                   result.ElapsedMilliseconds, timer.Elapsed.Milliseconds));

                timer.Restart();
                client.Reset();
                OnRaiseDetectionEvent(detectionEventArgs);
            }
        }

        void OnRaiseDetectionEvent(DetectionEventArgs e)
        {
            EventHandler<DetectionEventArgs> handler = RaiseDetectionEvent;
```

```
            if (handler != null)
            {
                handler(this, e);
            }
        }
    }

    public class DetectionEventArgs : EventArgs
    {
        public YoloResult Result { get; private set; }

        public DetectionEventArgs(YoloResult result)
        {
            Result = result;
        }
    }
```

ClientWrapper.cs

```
public class ClientWrapper
    {
        public bool IsIdle => state == State.Idle;
        public bool IsBusy => state == State.Busy;
        public bool HasNewResponse => state == State.NewResponse;

        enum State
        {
            Idle,
            Busy,
            NewResponse
        }
        State state = State.Idle;

        readonly YoloService.YoloServiceClient client;

        public ClientWrapper(YoloService.YoloServiceClient client)
        {
            this.client = client;
        }

        public void Reset()
        {
            state = State.Idle;
        }

        public async Task Detect(byte[] imageData, YoloResult result)
        {
            try
            {
                state = State.Busy;
                DetectionRequest request = new DetectionRequest { Image =
ByteString.CopyFrom(imageData) };

                using (var call = client.Detect(request))
                {
                    var responseStream = call.ResponseStream;
                    while (await responseStream.MoveNext())
                    {
                        DetectionResponse response = responseStream.Current;
                        foreach (DetectionResult r in response.YoloItems)
                        {
```

```
                            result.Add(new YoloItem(r.Type, r.Confidence, r.X, r.Y,
r.Width, r.Height));
                        }
                        result.ElapsedMilliseconds = response.ElapsedMilliseconds;
                        state = State.NewResponse;
                    }
                }
            }
            catch (RpcException e)
            {
                UnityEngine.Debug.LogError("RPC failed " + e);
                throw;
            }
        }
    }
```

YoloItem.cs

```
public class YoloItem
    {
        public string Type { get; private set; }
        public float Confidence { get; private set; }
        public RectInt Rect { get; private set; }

        public YoloItem(string type, double confidence, int x, int y, int width, int
height)
        {
            Type = type;
            Confidence = (float)confidence;
            Rect = new RectInt(x, y, width, height);
        }

        public void Expand(RectInt rect)
        {
            Rect.SetMinMax(
                Vector2Int.Min(Rect.min, rect.min),
                Vector2Int.Max(Rect.max, rect.max)
            );
        }

        public override string ToString()
        {
            return string.Format("YoloItem Type:{0} Conf:{1} Rect:{2}", Type,
Confidence, Rect);
        }
    }
```

YoloResult.cs

```
public class YoloResult
    {
        public int ElapsedMilliseconds;
        public Dictionary<string, List<YoloItem>> Dict { get; private set; }

        public YoloResult()
        {
            Dict = new Dictionary<string, List<YoloItem>>();
        }
```

```
        public List<YoloItem> ToList(float confidenceThreshold = 0)
        {
            return Dict.Values.SelectMany(x => x).Where(x => x.Confidence >=
confidenceThreshold).ToList();
        }

        public void Clear()
        {
            Dict.Clear();
        }

        public void Add(YoloItem item)
        {
            if (!Dict.ContainsKey(item.Type))
            {
                Dict.Add(item.Type, new List<YoloItem>());
            }
            Dict[item.Type].Add(item);
        }
    }
```

Label.cs

```
public class Label : MonoBehaviour
    {
        Image frame;
        RectTransform frameRect;

        Text text;
        RectTransform textRect;
        Vector2 textOffset;
        float textHeight;

        public void OnUpdate(Size size, Color color, YoloItem item)
        {
            gameObject.SetActive(true);

            frame.color = color;
            text.color = color.grayscale > 0.5f ? Color.black : Color.white;

            RectInt r = item.Rect;
            frameRect.offsetMin = new Vector2(
                r.x * size.Factor, (size.Image.y - r.height - r.y) * size.Factor);
            frameRect.offsetMax = new Vector2(
                (r.x - (size.Image.x - r.width)) * size.Factor, -r.y * size.Factor);

            text.text = item.Type + " " + Mathf.Round(item.Confidence * 100) + "%";
            textRect.anchoredPosition = new Vector2(
                (r.width * size.Factor) / 2 + textOffset.x, textOffset.y);
            textRect.sizeDelta = new Vector2(r.width * size.Factor, textHeight);
        }

        public void OnUpdate()
        {
            gameObject.SetActive(false);
        }

        void Awake()
        {
            frame = transform.GetComponent<Image>();
            frameRect = transform.GetComponent<RectTransform>();
```

```
            text = transform.GetComponentInChildren<Text>();
            textRect = text.GetComponent<RectTransform>();
            textOffset = new Vector2(textRect.anchoredPosition.x -
textRect.sizeDelta.x / 2, textRect.anchoredPosition.y);
            textHeight = textRect.sizeDelta.y;
        }
    }
```

Main.cs

```
public class Main : MonoBehaviour
    {
        [SerializeField]
        [Range(0f, 1f)]
        float confidenceThreshold = 0;

        ClientManager clientManager;
        SizeConfig sizeConfig;
        Texture2D texture;
        Monitor monitor;
        Cam cam;

        public void Initialize()
        {
            sizeConfig = GetComponent<SizeConfig>();
            sizeConfig.RaiseResizeEvent += OnScreenResize;
            Size size = sizeConfig.Initialize();

            texture = new Texture2D(size.Image.x, size.Image.y, TextureFormat.RGB24,
false);
            cam = GameObject.FindObjectOfType<Cam>();
            cam.Initialize(ref texture, size);

            monitor = GameObject.FindObjectOfType<Monitor>();
            monitor.Initialize(size,
LabelColors.CreateFromJSON(Resources.Load<TextAsset>("LabelColors").text));

            clientManager = new ClientManager(ref texture);
            clientManager.RaiseDetectionEvent += OnDetection;
        }

        void Start()
        {
            Initialize();
        }

        void Update()
        {
            clientManager.Update();
        }

        void OnDetection(object sender, DetectionEventArgs e)
        {
            monitor.UpdateLabels(e.Result.ToList(confidenceThreshold));
        }

        void OnScreenResize(object sender, ResizeEventArgs e)
        {
            texture.Resize(e.Size.Image.x, e.Size.Image.y);
            monitor.SetSize(e.Size);
```

```
            cam.SetSize(e.Size);
        }

        void OnApplicationQuit()
        {
            sizeConfig.RaiseResizeEvent -= OnScreenResize;
            clientManager.RaiseDetectionEvent -= OnDetection;
            clientManager.Dispose();
        }
    }
```

Monitor.cs

```
public class Monitor : MonoBehaviour, IResizable
    {
        Label labelTemplate;
        List<Label> labels;
        RectTransform rect;
        LabelColors labelColors;
        Size size;

        public void Initialize(Size size, LabelColors labelColors)
        {
            labelTemplate = transform.GetComponentInChildren<Label>();
            labels = new List<Label>() { labelTemplate };
            rect = GetComponent<RectTransform>();

            this.labelColors = labelColors;
            SetSize(size);
        }

        public void SetSize(Size size)
        {
            this.size = size;
            rect.sizeDelta = new Vector2(Screen.width, Screen.height);
        }

        public void UpdateLabels(List<YoloItem> list)
        {
            int diff = list.Count - labels.Count;
            if (diff > 0)
            {
                CreateMissingLabels(diff);
            }

            for (int i = 0; i < labels.Count; i++)
            {
                if (i < list.Count)
                {
                    labels[i].OnUpdate(size, labelColors.GetColor(list[i].Type),
list[i]);
                }
                else
                {
                    labels[i].OnUpdate();
                }
            }
        }

        void CreateMissingLabels(int n)
        {
```

```
            for (int i = 0; i < n; i++)
            {
                labels.Add(Instantiate(labelTemplate,
transform).GetComponent<Label>());
            }
        }
    }
```

SizeConfig.cs

```
public class SizeConfig : MonoBehaviour
    {
        [HideInInspector]
        public event EventHandler<ResizeEventArgs> RaiseResizeEvent;

        [SerializeField]
        int imageWidth = 416;

        Size size;

        public Size Initialize()
        {
            size = new Size(Screen.width, Screen.height, imageWidth);
            return size;
        }

        void Update()
        {
            if (size.Screen.x != Screen.width || size.Screen.y != Screen.height)
            {
                size = new Size(Screen.width, Screen.height, imageWidth);
                OnRaiseResizeEvent(new ResizeEventArgs(size));
            }
        }

        void OnRaiseResizeEvent(ResizeEventArgs e)
        {
            EventHandler<ResizeEventArgs> handler = RaiseResizeEvent;
            if (handler != null)
            {
                handler(this, e);
            }
        }
    }

    public class Size
    {
        public Vector2 Screen { get; private set; }
        public Vector2Int Image { get; private set; }
        public float Factor { get; private set; }

        public Size(float screenWidth, float screenHeight, int imageWidth)
        {
            Screen = new Vector2(screenWidth, screenHeight);
            Factor = screenWidth / (float)imageWidth;
            Image = new Vector2Int(imageWidth, Mathf.RoundToInt(screenHeight /
Factor));
        }

        public override string ToString()
        {
```

```
            return string.Format("Size Screen:{0} Image:{1} Factor:{2}", Screen,
Image, Factor);
        }
    }

    public class ResizeEventArgs : EventArgs
    {
        public Size Size { get; private set; }

        public ResizeEventArgs(Size size)
        {
            Size = size;
        }
    }

    public interface IResizable
    {
        void SetSize(Size size);
    }
```

ДОДАТОК Б

YoloWrapper.cs

```
public class YoloWrapper : IDisposable
    {
        public const int MaxObjects = 1000;
        private const string YoloLibraryCpu = @"yolo_cpp_dll_cpu.dll";
        private const string YoloLibraryGpu = @"yolo_cpp_dll_gpu.dll";

        private readonly Dictionary<int, string> _objectType = new Dictionary<int,
string>();
        private readonly ImageAnalyzer _imageAnalyzer = new ImageAnalyzer();

        public DetectionSystem DetectionSystem { get; private set; } =
DetectionSystem.Unknown;
        public EnvironmentReport EnvironmentReport { get; private set; }

        #region DllImport Cpu

        [DllImport(YoloLibraryCpu, EntryPoint = "init")]
        private static extern int InitializeYoloCpu(string configurationFilename,
string weightsFilename, int gpu);

        [DllImport(YoloLibraryCpu, EntryPoint = "detect_image")]
        internal static extern int DetectImageCpu(string filename, ref BboxContainer
container);

        [DllImport(YoloLibraryCpu, EntryPoint = "detect_mat")]
        internal static extern int DetectImageCpu(IntPtr pArray, int nSize, ref
BboxContainer container);

        [DllImport(YoloLibraryCpu, EntryPoint = "dispose")]
        internal static extern int DisposeYoloCpu();

        #endregion

        #region DllImport Gpu

        [DllImport(YoloLibraryGpu, EntryPoint = "init")]
        internal static extern int InitializeYoloGpu(string configurationFilename,
string weightsFilename, int gpu);
```

```
        [DllImport(YoloLibraryGpu, EntryPoint = "detect_image")]
        internal static extern int DetectImageGpu(string filename, ref BboxContainer
container);

        [DllImport(YoloLibraryGpu, EntryPoint = "detect_mat")]
        internal static extern int DetectImageGpu(IntPtr pArray, int nSize, ref
BboxContainer container);

        [DllImport(YoloLibraryGpu, EntryPoint = "dispose")]
        internal static extern int DisposeYoloGpu();

        [DllImport(YoloLibraryGpu, EntryPoint = "get_device_count")]
        internal static extern int GetDeviceCount();

        [DllImport(YoloLibraryGpu, EntryPoint = "get_device_name")]
        internal static extern int GetDeviceName(int gpu, StringBuilder deviceName);

        #endregion

        public YoloWrapper(YoloConfiguration yoloConfiguration, bool ignoreGpu =
false)
        {
            this.Initialize(yoloConfiguration.ConfigFile,
yoloConfiguration.WeightsFile, yoloConfiguration.NamesFile, 0, ignoreGpu);
        }

        public YoloWrapper(string configurationFilename, string weightsFilename,
string namesFilename, int gpu = 0, bool ignoreGpu = false)
        {
            this.Initialize(configurationFilename, weightsFilename, namesFilename,
gpu, ignoreGpu);
        }

        public void Dispose()
        {
            switch (this.DetectionSystem)
            {
                case DetectionSystem.CPU:
                    DisposeYoloCpu();
                    break;
                case DetectionSystem.GPU:
```

```
                    DisposeYoloGpu();
                    break;
            }
        }

        private void Initialize(string configurationFilename, string
weightsFilename, string namesFilename, int gpu = 0, bool ignoreGpu = false)
        {
            if (IntPtr.Size != 8)
            {
                throw new NotSupportedException("Only 64-bit processes are
supported");
            }

            this.EnvironmentReport = this.GetEnvironmentReport();
            //if
(!this.EnvironmentReport.MicrosoftVisualCPlusPlus2017RedistributableExists)
            //{
            //    throw new DllNotFoundException("Microsoft Visual C++ 2017
Redistributable (x64)");
            //}

            this.DetectionSystem = DetectionSystem.CPU;
            if (!ignoreGpu && this.EnvironmentReport.CudaExists &&
this.EnvironmentReport.CudnnExists)
            {
                this.DetectionSystem = DetectionSystem.GPU;
            }

            switch (this.DetectionSystem)
            {
                case DetectionSystem.CPU:
                    InitializeYoloCpu(configurationFilename, weightsFilename, 0);
                    break;
                case DetectionSystem.GPU:
                    var deviceCount = GetDeviceCount();
                    if (deviceCount == 0)
                    {
                        throw new NotSupportedException("No graphic device is
available");
                    }
```

```
                    if (gpu > (deviceCount - 1))
                    {
                        throw new IndexOutOfRangeException("Graphic device index is
out of range");
                    }

                    var deviceName = new StringBuilder(); //allocate memory for
string
                    GetDeviceName(gpu, deviceName);
                    this.EnvironmentReport.GraphicDeviceName =
deviceName.ToString();

                    InitializeYoloGpu(configurationFilename, weightsFilename, gpu);
                    break;
            }

            var lines = File.ReadAllLines(namesFilename);
            for (var i = 0; i< lines.Length; i++)
            {
                this._objectType.Add(i, lines[i]);
            }
        }

        private bool IsMicrosoftVisualCPlusPlus2017Available()
        {
            //Detect if Visual C++ Redistributable for Visual Studio is installed
            //https://stackoverflow.com/questions/12206314/detect-if-visual-c-
redistributable-for-visual-studio-2012-is-installed/34209692#34209692
            var checkKeys = new string[]
            {
                @"Installer\Dependencies\,,amd64,14.0,bundle",
                @"Installer\Dependencies\VC,redist.x64,amd64,14.16,bundle",
            };

            foreach (var checkKey in checkKeys)
            {
                using (var registryKey = Registry.ClassesRoot.OpenSubKey(checkKey,
false))
                {
                    if (registryKey == null)
                    {
                        continue;
```

```
                    }

                    var displayName = registryKey.GetValue("DisplayName") as string;
                    if (string.IsNullOrEmpty(displayName))
                    {
                        continue;
                    }

                    if (displayName.StartsWith("Microsoft Visual C++ 2017
Redistributable (x64)", StringComparison.OrdinalIgnoreCase))
                    {
                        return true;
                    }
                }
            }

            return false;
        }

        private EnvironmentReport GetEnvironmentReport()
        {
            var report = new EnvironmentReport();
            report.MicrosoftVisualCPlusPlus2017RedistributableExists =
this.IsMicrosoftVisualCPlusPlus2017Available();

            if (File.Exists(@"cudnn64_7.dll"))
            {
                report.CudnnExists = true;
            }

            var envirormentVariables =
Environment.GetEnvironmentVariables(EnvironmentVariableTarget.Machine);
            if (envirormentVariables.Contains("CUDA_PATH"))
            {
                report.CudaExists = true;
            }

            return report;
        }

        public IEnumerable<YoloItem> Detect(string filepath)
        {
```

```
    if (!File.Exists(filepath))
    {
        throw new FileNotFoundException("Cannot find the file", filepath);
    }

    var container = new BboxContainer();
    var count = 0;
    switch (this.DetectionSystem)
    {
        case DetectionSystem.CPU:
            count = DetectImageCpu(filepath, ref container);
            break;
        case DetectionSystem.GPU:
            count = DetectImageGpu(filepath, ref container);
            break;
    }

    if (count == -1)
    {
        throw new NotImplementedException("C++ dll compiled incorrectly");
    }

    return this.Convert(container);
}

public IEnumerable<YoloItem> Detect(byte[] imageData)
{
    if (!this._imageAnalyzer.IsValidImageFormat(imageData))
    {
        throw new Exception("Invalid image data, wrong image format");
    }

    var container = new BboxContainer();
    var size = Marshal.SizeOf(imageData[0]) * imageData.Length;
    var pnt = Marshal.AllocHGlobal(size);

    try
    {
        // Copy the array to unmanaged memory.
        Marshal.Copy(imageData, 0, pnt, imageData.Length);
        var count = 0;
        switch (this.DetectionSystem)
```

```
                {
                    case DetectionSystem.CPU:
                        count = DetectImageCpu(pnt, imageData.Length, ref
container);
                        break;
                    case DetectionSystem.GPU:
                        count = DetectImageGpu(pnt, imageData.Length, ref
container);
                        break;
                }

                if (count == -1)
                {
                    throw new NotImplementedException("C++ dll compiled
incorrectly");
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception);
                return null;
            }
            finally
            {
                // Free the unmanaged memory.
                Marshal.FreeHGlobal(pnt);
            }

            return this.Convert(container);
        }

        private IEnumerable<YoloItem> Convert(BboxContainer container)
        {
            var yoloItems = new List<YoloItem>();
            foreach (var item in container.candidates.Where(o => o.h > 0 || o.w >
0))
            {
                var objectType = this._objectType[(int)item.obj_id];
                var yoloItem = new YoloItem() { X = (int)item.x, Y = (int)item.y,
Height = (int)item.h, Width = (int)item.w, Confidence = item.prob, Type = objectType
};
                yoloItems.Add(yoloItem);
```

```
            }


            return yoloItems;
        }
    }
```

ImageAnalyzer.cs

```
public class ImageAnalyzer
    {
        private Dictionary<string, byte[]> _imageFormats = new Dictionary<string,
byte[]>();

        public ImageAnalyzer()
        {
            var bmp = Encoding.ASCII.GetBytes("BM");  //BMP
            var png = new byte[] { 137, 80, 78, 71 }; //PNG
            var jpeg = new byte[] { 255, 216, 255 };  //JPEG

            this._imageFormats.Add("bmp", bmp);
            this._imageFormats.Add("png", png);
            this._imageFormats.Add("jpeg", jpeg);
        }

        public bool IsValidImageFormat(byte[] imageData)
        {
            if (imageData.Length <= 3)
            {
                return false;
            }

            foreach(var imageFormat in this._imageFormats)
            {
                if
(imageData.Take(imageFormat.Value.Length).SequenceEqual(imageFormat.Value))
                {
                    return true;
                }
            }

            return false;
        }
    }
```

YoloItem.cs

```
    public class YoloItem
    {
        public string Type { get; set; }
        public double Confidence { get; set; }
        public int X { get; set; }
        public int Y { get; set; }
        public int Width { get; set; }
        public int Height { get; set; }
```

```
        public Point Center()
        {
            return new Point(this.X + this.Width / 2, this.Y + this.Height / 2);
        }
    }
```

Program.cs

```
    class Program
    {
        static void Main(string[] args)
        {
            const int Port = 50052;
            YoloServiceImpl impl = new YoloServiceImpl();

            Server server = new Server
            {
                Services = { YoloService.BindService(impl) },
                Ports = { new ServerPort("localhost", Port,
ServerCredentials.Insecure) }
            };
            server.Start();

            Console.WriteLine("YoloService server listening on port " + Port);
            Console.WriteLine("Press any key to stop the server...");
            Console.ReadKey();

            impl.Dispose();
            server.ShutdownAsync().Wait();
        }
    }
```