МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія розпізнавання та трекінгу об'єктів для систем захисту від безпілотних літальних апаратів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Москаленко В. В.**

Студента групи ІНм – 91н **Паращенко В. А.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою

_____

_____

“_____” _____

20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Паращенку Владиславу Анатолійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія розпізнавання та трекінгу об'єктів для систем захисту від безпілотних літальних апаратів*

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) Огляд основних концепцій та підходів детектування та трекінгу об'єктів; 2) постановка задачі і формування завдань для реалізації; 3) огляд і порівняння основних кроків реалізації; 4) вибір інструментарію та програмних засобів для реалізації програмного продукту; 4) проектування та розробка прототипу системи для детектування та трекінгу безпілотних літаючих об'єктів; 5) аналіз результатів.

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. Постановка задачі*		
2.	*Аналіз існуючих технологій для детектування та трекінгу*		
3.	*Застосування згоркових мереж в задачах детектування та трекінгу об'єктів*		
4.	*Розробка інформаційного та програмного забезпечення для система захисту від безпілотних літаючих об'єктів*		
5.	*Оформлення пояснювальної записки до дипломної роботи.*		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 68 с., 46 рис., 4 табл., 20 літературних джерел, 2 додатки.

Об'єкт дослідження — Інформаційна технологія розпізнавання та трекінгу об'єктів для систем захисту від безпілотних літальних апаратів

Мета роботи — створити програмний комплекс для детектування та трекінгу об’єктів для систем захисту від безпілотних літаючих об’єктів.

Методи дослідження — згорткові нейронні мережі в задачах трекінгу та детектування об’єктів

Результати — досліджено можливості застосування згорткових нейронних мереж в задачах детектування та трекінгу для систем захисту від безпілотних літаючих об’єктів. Для перевірки гіпотез було обрано дві моделі Single shot detection та Faster R-CNN. Дані моделі були навчені на попередньо підібраному датасеті з зображень безпілотних літаючих об’єктів. Навчені моделі було протестовано на якість та шкидкість розпізнавання. Було практично підтверджено можливості застосування згорткових мереж в системах захисту від безпілотних літаючих об’єктів. Також була досліджена техніка аугментації для оптимізації якості детектування. Отримані результати підтверджують, що згорткові мережі можуть бути застосовані в якості детекторів живого часу для систем захисту від безпілотних літаючих об’єктів.

PYTHON, OPEN CV, SSD, R-CNN, COMPUTER VISION, CONVULATIONAL NETWORK, TENSORFLOW

ЗМІСТ

ВСТУП

Безпілотні літаючі апарати(БПЛА) стрімко розвиваються та знаходять застосування у багатьох сферах. Завдяки розвитку технології GPS навігації стала можливість застосування таких апаратів багатьох сферах. Наприклад, дрони знайшли застосування у сільському господарстві, їх застосовають для внесення добрив та хімікатів, систем охорони та контролю врожаю[1]. Такі компанії, як Uber та Amazon давно розробляють системи доставки з дронами в якості кур'єрів. Ці компанії вже активно випробовують системи такі системи для доставки. Також дрони ідеально підходять для систем моніторингу, вже зараз дронів використовуються для контролю за кордонами країн, цілісністю електромереж і трубопроводів, та навіть для оцінок і прогнозування заторів.

Як і багато споживчих технологій безпілотні апарати спочатку розроблялися ще у військових цілях. Ще у 1849 році безпілотні аеростати використовувалися для повітряного бомбардування Венеції. З подальшим винайденням електроенергії та радіо зв'язку безпілотні системи застосовувалися в Першій та Другій світовій війні для військових ударів, під час холодної війни для шпигунства та продовжують розроблятися та застосовуватися і у наш час. Саме завдяки високій дальності польотів, відсутності систем забезпечення життєдіяльності пасажирів та значному розміру корисного навантаження технології БПЛА є одним з пріоритетних напрямків розробки сучасного військового обладнання.

Нажаль розвиток технологій безпілотної авіації має і негативні наслідки. Безпілотні літаючи апарати почали застосовуватися в терористичних цілях, доставці незаконних вантажів та шпигунстві. Зростання доступності побудови чи покупки безпілотних літаючих апаратів провокує те, що такі апарати все частіше опиняються в руках зловмисників. Наприклад, дрони були використані для обмеження та призупинки роботи аеропорту в Лондоні[2], також дрони регулярно дрони застовуються для доставки незаконних вантажів у в'язниці[3]. Хоча споживчі дрони і значно відстають від військових розробок ціна та доступність провокує використання цих апаратів у злочинних цілях,

окрім того системи захисту звичайних дронів доволі прості, тож часто трапляються випадки незаконного використання та викрадення чужих дронів. Саме тому багато держав активно займається розробкою систем для контролю та протидії безпілотникам, а також посилюють контроль у правовому полі[4][5].

Споживчі дрони зазвичай мають невеликі розміри, низький рівень шуму та високу маневреність, що дозволяє залишатися їм доволі непомітними. Окрім цього зазвичай оснащуються декількома камерами, що допомагає управляти дроном не знаходячись в області бачення. Завдяки розвинутій системі камер з’явилася технологія FPV(First person view), що дозволяє управляти дроном з видом від першої особи, подібно управлінню в відеоіграх. Наявність компактних та високоякісних камер, потужні передавачі, які забезпечують передачу даних на десятки кілометрів, а також системи автоматичного управління перетворюють БПЛА на небезпечні автономні системи від яких необхідно захищатися.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Сучасний стан та тенденції розвитку систем захисту від безпілотних апаратів

З часу виникнення безпілотних літаючих апаратів розвиваються і технології захисту від апаратів. Існує доволі багато технологій захисту, найбільш популярнішими методами нейтралізації є:

- Акустичні
- Лазерні
- Мікрохвильові
- Метод звичайної сітки
- Системи перехоплення управління
- Дрони-перехоплювачі
- Системи радіо-електронної боротьби(РЕБ)

Акустична установка працює за принципом впливу на дрон звуком резонансної частоти з внутрішнім гіроскопом пристрою. Гіроскоп це компонент, який дозволяє контролювати положення в просторі для. Підбір резонансної частоти провокує неправильні показання гіроскопу, що призводить до аварії. Недоліком цього методу є те що, він працює лише на відносно недорогих апаратах, які не мають магнітометра, котрий буде видававати правильні горозонтальні координати незважаючи на акустичний вплив. Також радіус дії такоої установки обмежений приблизно 50 метрами при потужності звуку в 150 Дб.

Розробкою лазерних систем займаються ВМФ США. Їх система дозволяє нейтралізувати дрони в радіусі до 2 км від установки. Основою такою установки є твердотільний лазер, який працює в інфракрасному діапазоні. Потужність такого лазера досягая 30 кВт, час нейтралізації цілі до 2 секунд. Установка може працювати як в режимі знешкодження сенсорів цілі, так і у високо енергетичному режимі, коли ціль повністю знешкоджується.

Мікрохвильові системи працюють трохи за іншим принципом. На відміну від прямого фізичного впливу з використанням лазера, тут знешкодження цілі досягається через знищення внутрішніх електронних компонентів дрона під впливом електромагнітного випромінювання. Перевагою таких систем є можливість нейтралізації відразу декількох цілей, які знаходяться поруч.

Застосування звичайної сітки є найбільш простим на інтуїтивним методом знешкодження дрону. Потрібно лише детектувати дрон та вистрілити в нього сіткою враховуючи траєкторію руху дрону. Така система може бути як повністю ручною, так і автоматизованою. Розробкою автоматизованих систем займається компанія OpenWorks Engineering, яка має розробки систем котрі можуть автоматизовано нейтралізувати дронів у радіусі до метрів від установки. Їх розробки можуть працювати як в автономному режимі, так і в режимі ручного управління.

Дрони-перехоплювачі являють собою БПЛА, які націлені на знешкодження інших дронів. Такі апарати зазвичай мають більш міцний корпус та двигуни порівняно зі звичайними моделями. Такий дрон може бути обладнаний сіткою для знешкодження несанкціонованого дрону. Розробкою подібних систем займаються спеціалісти з Японію.

Системи РЕБ вже зараз є на озброєнні багатьох армій світу. Такі засоби дозволяють переривати зв’язок між дроном та оператором за допомогою шуму, який передається на частотах, які популярні для управління громадськими дронами. В залежності від класу апарату він може по різному реагувати на втрату сигналу від оператору. Деякі апарати намагаються повернутися до координат зльоту використовуючи автономне управління та системи GPS та ГЛОНАСС. Для того, щоб запобігти поверненню апарату до рук зловмисника необхідно не лише блокувати сигнали на частотах управління, а також і сигнали навігаційних систем. Розвиток систем радіо-електронної боротьби привів до виникнення систем переносних систем у

формфакторі гвинтівок. Недоліком цього цього методу є неможливість нейтралізації дронів, які працюють на частотах на які не розрахована система РЕБ чи взагалі працюють повністю автономно, за попередньо заданою програмою.

Системи перехоплення управління подібні до систем РЕБ, та навідміну від “зашумлення” частот передачі сигналів управління такі системи намагаються передавати власні команди для БПЛА. Можна охарактеризувати 3 методи, які можуть бути застосовані до перехоплення контролю:

- Злом ключів шифрування за допомогою підбору або вразливостей реалізації того чи іншого протоколу шифрування
- Використання вразливостей ПО, наприклад, перевантаження буферу
- Використання оригінальних інтерфейсів управління для виконання стороннього коду

Навіть дорогі БПЛА побудовані на технології Wi-Fi для зв'язку з оператором літаючого засобу. Зазвичай тут застосовуються мінімальні протоколи шифрування для мінімізації затримок при управлінні. Так наприклад протокол WEP, який широко застосовуються для забезпечення безпеки мереж Wi-Fi, можливо зламати за декілька секунд. В результаті можливо виконати так звану атаку “man-in-the-middle” перехоплюючі пакети від оператора та посилаючи власні команди на управління.

Для захисту від безпілотних літаючих об’єктів необхідні не лише технології знешкодження, але і технології детектування та трекінгу таких об’єктів. Дана робота розглядає застосування технологій комп’ютерного зору в задачах трекінгу БПЛА для використання в системах захисту.

1.2. Аналіз методів детектування і трекінгу об'єктів інтересу на відео

Потреба в детектуванні об’єктів на вхідних зображеннях виникла доволі давно. На основі технологій детектування будуються системи автономного управління транспортними засобами, системи контролю якості, автоматизовані системи управління камерою для спортивних подій.

Перш за все розглянемо різницю між поняттями трекінгу та детектування на відео. Детектування це алгоритм чи нейронна мережа, яка знаходить об’єкт в кадрі відео, класифікує його і повертає в якості результату координати об’єкту та його клас. Візуально це можна представити так (див. Рисунок 1.1).

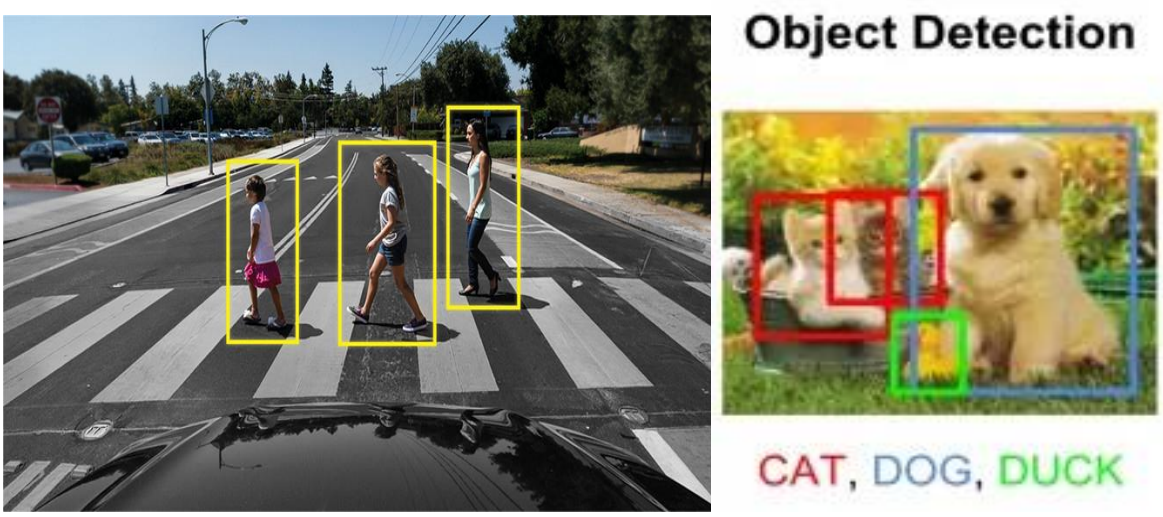


Рисунок 1.1 – приклади детектування об’єктів на зображенні

В загальному вигляді схема задачі комп’ютерного бачення виглядає наступним чином (див. Рисунок 1.2). Поточна робота розглядає задачу детектування для вирішення якої необхідно вирішити задачі класифікації об’єктів інтересу за класом та задачу локалізації областей розміщення об’єктів інтересу.

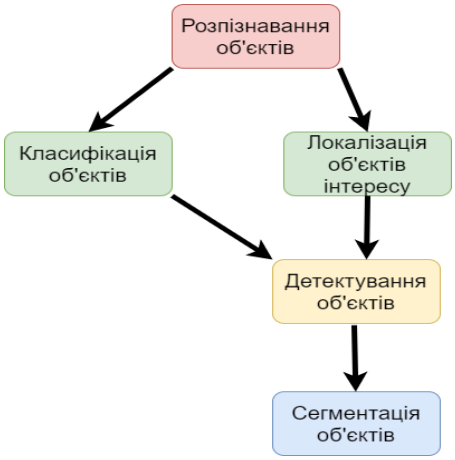


Рисунок 1.2 - Схема завдань області комп'ютерного зору

Ефективність моделі класифікації об'єктів на зображенні характеризується якість точністю розпізнавання класів, ефективність знаходження об'єктів характеризується відстанню між передбаченим розташуванням об'єкту в кадрі та його реальною позицією. Ефективність алгоритму детектування оцінюється як залежність між точністю розпізнавання класів кожного об'єкта на відео і співпадінню положенню розпізнаного об'єкта з реальним об'єктом в кадрі.

Для оцінки якості класифікації визначається показник залежності площі перетину прямокутника, тобто області інтересу, та площі отриманого прямокутника в результаті процесу детектування. Даний показник має назву Intersection over Union (IoU) та обчислюється за формулою:

$$IoU = \frac{AoO}{AoU} \tag{1.1}$$

де AoO (Area of Overlap) площа перетину справжнього об'єкту інтересу та площі об'єкту інтересу, який передбачено мережею; AoU (Area of Union) це площа об'єднання істинної області та передбаченої області.

Історію алгоритмів детектування можна розділити на дві епохи. До і після початку застосування нейронних мереж.

Перші алгоритми детектування працювали за алгоритмом ковзаючого вікна, коли по вхідному зображенню проходить область меншого розміру і перевіряється вміст цієї області на наявність в ній необхідних ознак певного класу. Якщо оцінка вища, за певний вхідний поріг, то фіксувалося детектування в цій області.

Для прикладу в 2001 році була опублікована робота в якій будується приклад першого детектору, який може працювати в режимі реального часу [6]. Класифікація областей в даній роботі виконується за допомогою знаходження суми значень пікселів в даній області. Робота розглядає побудову детектора обличчя на зображенні на основі логічних закономірностей. Наприклад, зона очей темніша за зону щік чи перенісся світліше за зону очей. За такими закономірностями можна побудувати ознаки для обличчя людини, по розмірам, положенню та яскравості певних пікселів на зображенні. Основною проблемою в застосуванні цього методу є те що навіть ковзаюче вікно розміром 24 на 24 пікселі дає нам 162 тисячі можливих ознак, що потребує значних витрат на розрахунок.

Для зменшення кількості зайвих спрацювань застосовується алгоритм Non maximum suppression, коли відкидається області, які перетинається з областю з найвищою оцінкою більш, ніж на T, де T оцінка перетину областей (див. Рисунок 1.3).

$$S_i = \begin{cases} S_i & \text{для } IoU < T \\ 0 & \text{для } IoU \geq T \end{cases} \quad (1.2)$$

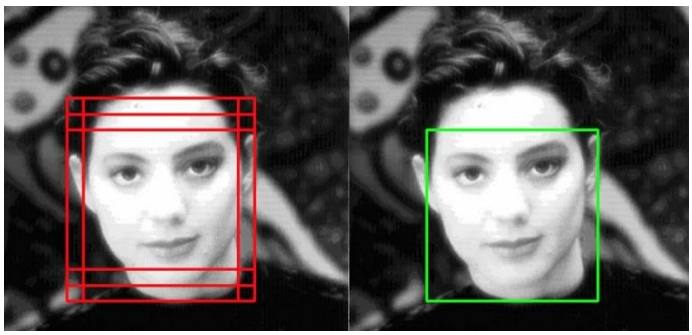


Рисунок 1.3 – приклад застосування NMS

Наступним кроком в розвитку систем детектування була публікація методу Histogram Oriented Gradients(HOG)[7] в 2007 році. Даний алгоритм також має 2 фази, тренування та використання отриманої моделі. Алгоритм приніс з собою нові ідеї по формування вектору ознак інтересу. В якості ковзаючого вікна даний алгоритм використовує область в 128 на 64 пікселі. Зрозуміло, що проходження вікна з таким розміром буде пропускати більші чи менші об’єкти. Тому алгоритм виконується декілька разів, на кожному кроці зображення сжимається зі збереженням пропорцій вихідного зображення і здійснюється прохід вікном того ж розміру.

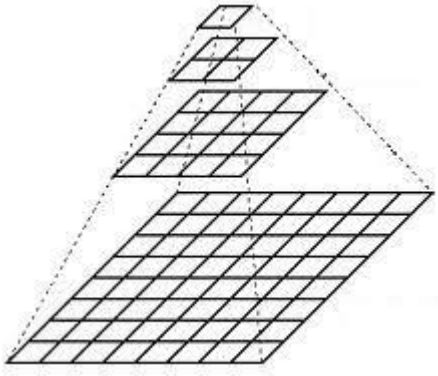


Рисунок 1.4 – Демонстрація ковзаючого вікна методом HMS

В якості вектора ознак використовується градієнт зображення, що являє собою значення в змінах яскравості пікселів та направлення цих змін. Для зменшення впливу оствітленості на якість розпізнавання застосовується техніка нормалізації. В результаті формування вектора ознак ми маємо масив градієнтів, які й розпізнаються класифікатором.

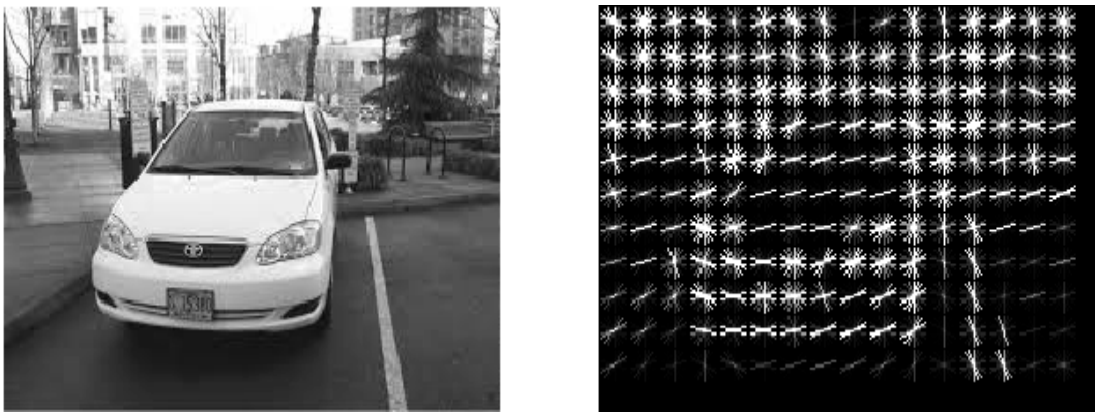


Рисунок 1.5 – візуалізація вектора ознак

Наступник кроком в технологіях детектування об’єктів став вихід моделі Deformable Part Model(DPM)[8]. Ця модель вирішувала проблему того що доволі складно натренувати модель на об’єкти складної форми. Для прикладу модель детектування пішохода повинна детектувати об’єкт з величезною кількістю можливих позицій. Автори пропонують інший підхід, тренується модель для детектування голови, рук, ніг, торсу. Потім за місцем положення детектованих об’єктів, можна зробити передбачення про клас об’єкту, який шукається(див. Рисунок 1.6). Такий підхід дозволив побудувати модель, яка може працювати зі значною кількістю варіацій положень об’єктів, які детектуються.

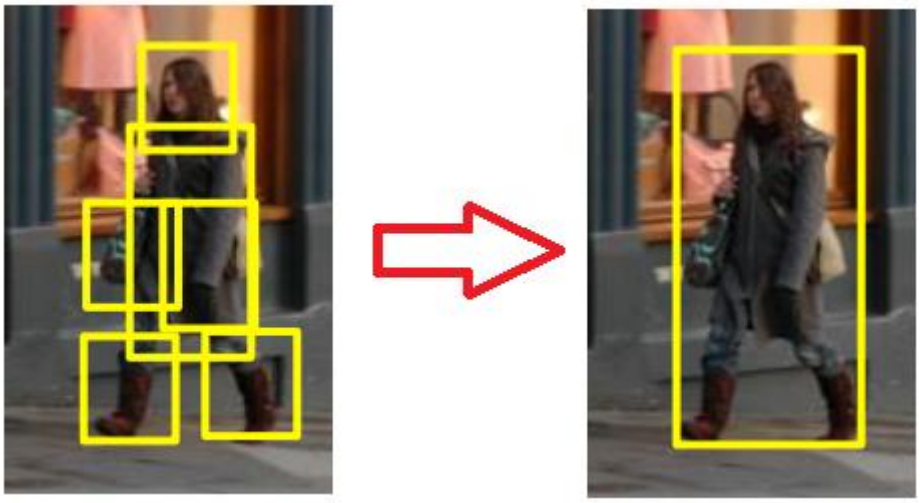


Рисунок 1.6 – демонстрація підходу моделі DPM

Наступним кроком в розвитку стала оптимізація алгоритмів пошуку областей для розпізнавання. Метод ковзаючого вікна потребував занадто багато ресурсів. Тому виникла ідея алгоритмів, які будуть генерувати області інтересу, настала ера так званих алгоритмів для пропозицій регіонів. Одним з таких методів став метод Selective search[9]. В ході роботи алгоритму пікселі, які мають подібний колір та яскравість заливаються одним кольором. Виконання закінчується коли всі області поєднуються в одну(див. Рисунок 1.7). Віднайдені області передається на кожному кроці до блоку детектування. В роботі автора вказується що для знаходження всіх об’єктів інтересу знаходиться 1000-1200 областей рекомендацій.

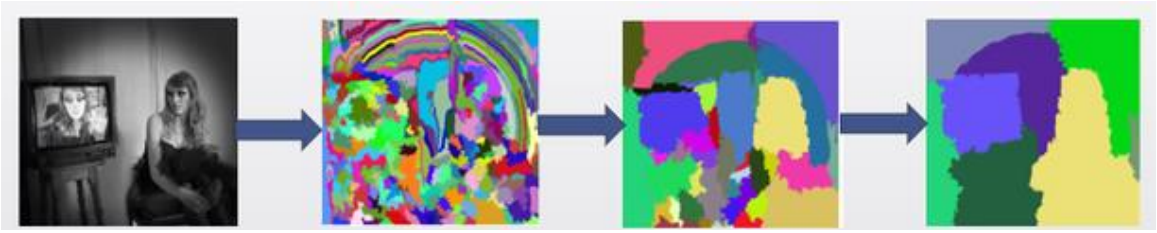


Рисунок 1.7 – візуалізація алгоритму Selective search

Далі в технологіях детектування об’єктів настала ера глибокого навчання. Завдяки розвитку розрахункових можливостей, нейронні мережі стало можливо застосовувати в задачах детектування. В 2012 році була представлена згорткова нейромережа AlexNet, яка побудована на технологіях CUDA. Сама мережа побудована ще на ідеях викладених в 1989 році Яном Лекуном[10], проте практична реалізація довго була неможливою через обмежені розрахункові ресурси.

В еру використання згорткових мереж можна розділити всі моделі, що розробляються на дві категорії:

- Моделі, що працюють в два етапа
- Моделі, що працюють в один етап

До моделей, що працюють в два етапи відносять сімейство RCNN(див. Рисунок 1.8), яке має дві окремі фази розпізнавання та класифікації. Такі моделі мають вищу точність, проте швидкість їх роботи нижча.

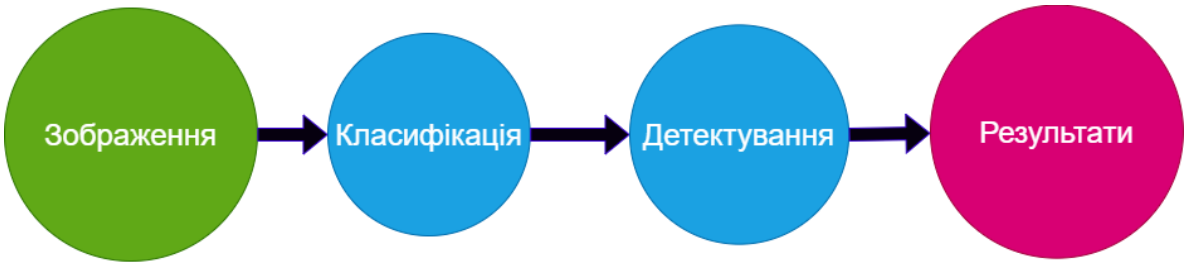


Рисунок 1.8 – Схема роботи RCNN

Моделі, що працюють в один прохід це сімейство SSD та YOLO(див. Рисунок 1.9). Швидкість цих моделей вища, на відміну від їх якості роботи.

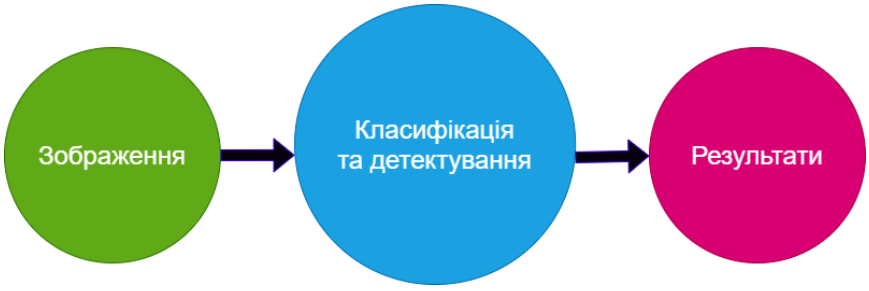


Рисунок 1.9 – Схема роботи моделей SSD та YOLO

1.3. Аналіз проблеми та постановка задачі

В ході даної роботи необхідно розглянути застосування технологій комп’ютерного зору для розробки систем захисту від безпілотних літаючих апаратів. Для перевірки гіпотези про можливість застосування даної технології необхідно побудувати програмний засіб, який може обробляти вхідний відео потік на якому проводиться детектування БПЛА.

Дослідження буде проводитися в декілька фаз. Перш за все необхідно сформувати набір даних для навчання. Для навчання детектору об’єктів інтересу необхідно дібрати широкий набір зображень, це можна зробити взявши з відкритих джерел необхідні фото та попередньо розмітивши ці фото. Після чого необхідно навчити обрану модель детектування на відібраному датасеті. Навчений детектор об’єктів інтересу має детектувати об’єкти, які нас цікавлять та видавати в якості вихідних даних координати боксу, який описує наш об’єкт. В разі побудови якісного класифікатора ми можемо контролювати об’єкти, які з’являються в області зору камери, яка знімає необхідну ділянку неба, яка нас цікавить.

Отримані результати від детектору передаються від детектору до трекеру. Відфільтрувавши детектовані об’єкти за класом ми передаємо координати об’єкту інтересу до нашого трекеру. Модель трекінгу подібна до детектору, проте має декілька ключових відмінностей:

- Працює з декількома кадрами;
- алгоритм трекінгу потребує значно менше обчислювальних ресурсів;
- оптимізований для задач відслідковування переміщення об’єкту.

2. ВИБІР МЕТОДУ РІШЕННЯ

2.1. Огляд моделей

Сімейство YOLO (You only look once)

Вперше дана модель була представлена Дж. Редмоном у 2015 році в його роботі: “You Only Look Once: Unified, Real-Time Object Detection.”. Даний підхід являє собою одну єдину навчену нейромережу, яка отримує зображення на вхід та передбачає координати та класи для кожного об’єкту інтересу. Точність застосування даного методу низька (особлива характерна помилка передбачення положення об’єкту), проте метод може обробляти від 45 кадрів відео в секунду та до 155 кадрів на секунду з оптимізованою моделлю. Модель працює за принципом розділення зображення на квадратні комірки та обробку кожної комірки окремо. Якщо модель вважає, що комірка містить об’єкт то повертається позитивний сигнал. Комірки, які розташовуються поряд об’єднуються за координатами і як результат ми маємо координати об’єкта, який може займати декілька комірок одночасно (див. Рисунок 2.1). Для розпізнавання класів будується так звана карта імовірностей для кожної комірки. Результатом роботи моделі буде поєднання результатів знаходження об’єктів та класифікації.

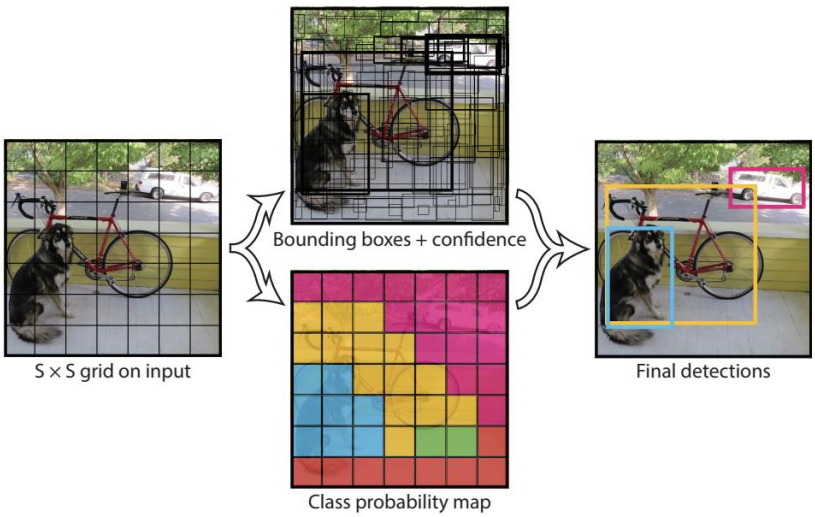


Рисунок 2.1 – схема роботи детектора сімейства YOLO

Сімейство детекторів SSD

Дане сімейство детекторів було представлено у праці Крістіана Чегеді [11]. За результатами проведених досліджень розроблена модель має значення середньої точності в 74% показника mAP(показник середнього значення точності) на швидкість детектування в 59 кадрів на секунду на стандартному на відібраному датасеті COCO.

Можна визначити 3 особливості даної моделі:

- Техніка одиничного проходу – тобто виокремлення областей інтересу та класифікація проходить в один етап, на відміну від моделей F-CNN. Це сприяє швидкості роботи моделі
- Техніка мультибоксів – розроблена К. Чегеді техніка, яка дозволяє в фазі навчання сформувати області інтересу з фіксованими розмірами, за допомогою яких знаходяться об’єкти в режимі реального виконання.
- Техніка детектування – техніка, яка поєднує в собі одночасно і функціонал детектування та розпізнавання.

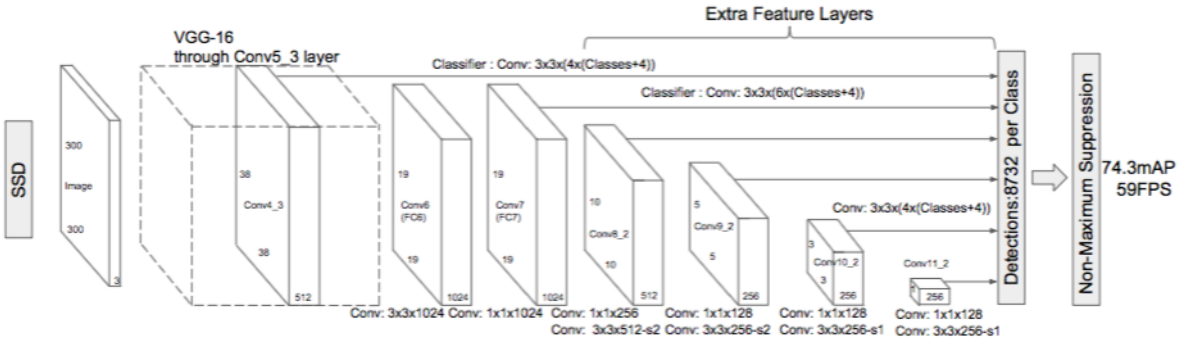


Рисунок 2.2 – архітектура моделі SSD

Як видно з Рисунок 2.2, в основі даної моделі лежить загально відома мережа VGG-16, в якій прибрані нижні повнозв’язні шари, замість яких додано 6 власних шарів нейромережі, 5 з яких використовуються для детектування об'єктів. Перші шари високої розмірності необхідні для знаходження великих за масштабом об’єктів. Нижні шари, з більш низькою розмірністю покликані для виявлення невеликих за масштабом об’єктів. На

даних отриманих від різних шарів, модель накладає сітку з областями інтересу. Дана сітка будується за схожим принципом з R-CNN сімейством. Обирається певна точка на зображенні, від неї відкладається прямокутник певного розміру, в результаті ми маємо вектор координат (x, y, h, w), для кожної точки інтересу будується декілька прямокутних областей в залежності від налаштувань моделі. Розміри та співвідношення сторін областей налаштовуються в процессі навчання, для прикладу для знаходження пішохода краще відходить вертикальна область зі співвідношенням сторін 0.42, для детектування автомобілів краще підходить горизонтальна область. Так як областей інтересу значна кількість, то використовується техніка не максимального подавлення, в результаті роботи якої області з показниками імовірності розпізнавання нижчі за 0.1 та з показниками IoU нижчі за 0.45 відкидаються. Для кожної прямокутної області, що потенційно містить шуканий об’єкт проводиться розпізнавання. Результатом якого є вектор з 21 елементу(20 існуючих класів та 1 елемент що в даній області не знаходиться об’єктів, які нас цікавлять).

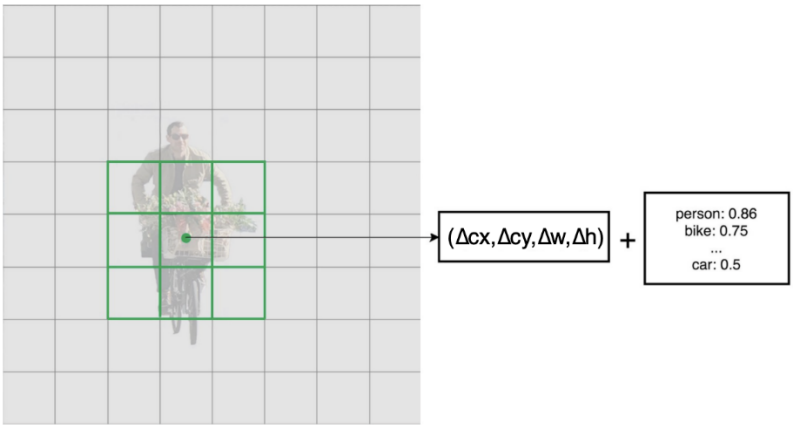


Рисунок 2.3 – приклад класифікації моделі SSD

Для оптимізації точності модель використовує підхід х аугментацією вхідного зображення. Для обробки ситуацій, які не представлені в тестоваому наборі даних використовується 3 техніки:

- Перевертання
- Обрізання

- Трансформація кольорів

Кожне зображення в навчальному датасеті може випадково піддатися 3 модифікаціям:

- Відсутність будь-якої модифікації
- Випадково змінити область, де значення IoU рівна 0.1, 0.3, 0.5, 0.7, 0.9
- Модифікація випадкової зони.

За результатами опублікованих досліджень точність від застосування техніки аугментації зростає наступним чином

Таблиця 2.1

Тип аугментації	**Модель SSD**		
Поворот	x	x	x
Випадкове обрізання		x	x
Розширення області			x
Точність моделі	**65**	**74**	**77**

Сімейство детекторів R-CNN

Сімейство детекторів ідея використання CNN нейромережі була висвітлена в роботах Роса Гиршака. Ним були розроблені такі моделі як R-CNN, Fast R-CNN та Faster R-CNN. Головна ідея даного сімейства алгоритмів полягає в застосуванні трьох модулів:

1. Модуль знаходження регіонів розпізнавання.
2. Модуль визначення ознак
3. Класифікатор

Перший модуль займається знаходженням так званих областей інтересу, які подаються на вхід наступним модулям. На цьому кроці виконується генерація областей інтересу(region proposals), для моделі R-CNN кількість таких областей може сягати 2000 областей. Моделі Fast та Faster R-CNN оптимізують даний крок, тому кількість областей для розпізнавання в цих моделей значно нижча. Для розпізнавання областей можна застосовувати декілька алгоритмів, найбільш популярними можна назвати алгоритми Edge Boxes[12] та Selective search[9].

Застосовуючи алгоритм Edge box ми формуємо з нашого зображення карту контурів, які являють собою рамки які обмежують об’єкт. Побудувавши навколо контурів прямокутник мінімального розміру в який входять всі контури одного об’єкта ми отримуємо прямокутник(див. Рисунок 2.4), який потенційно обмежує шуканий об’єкт.

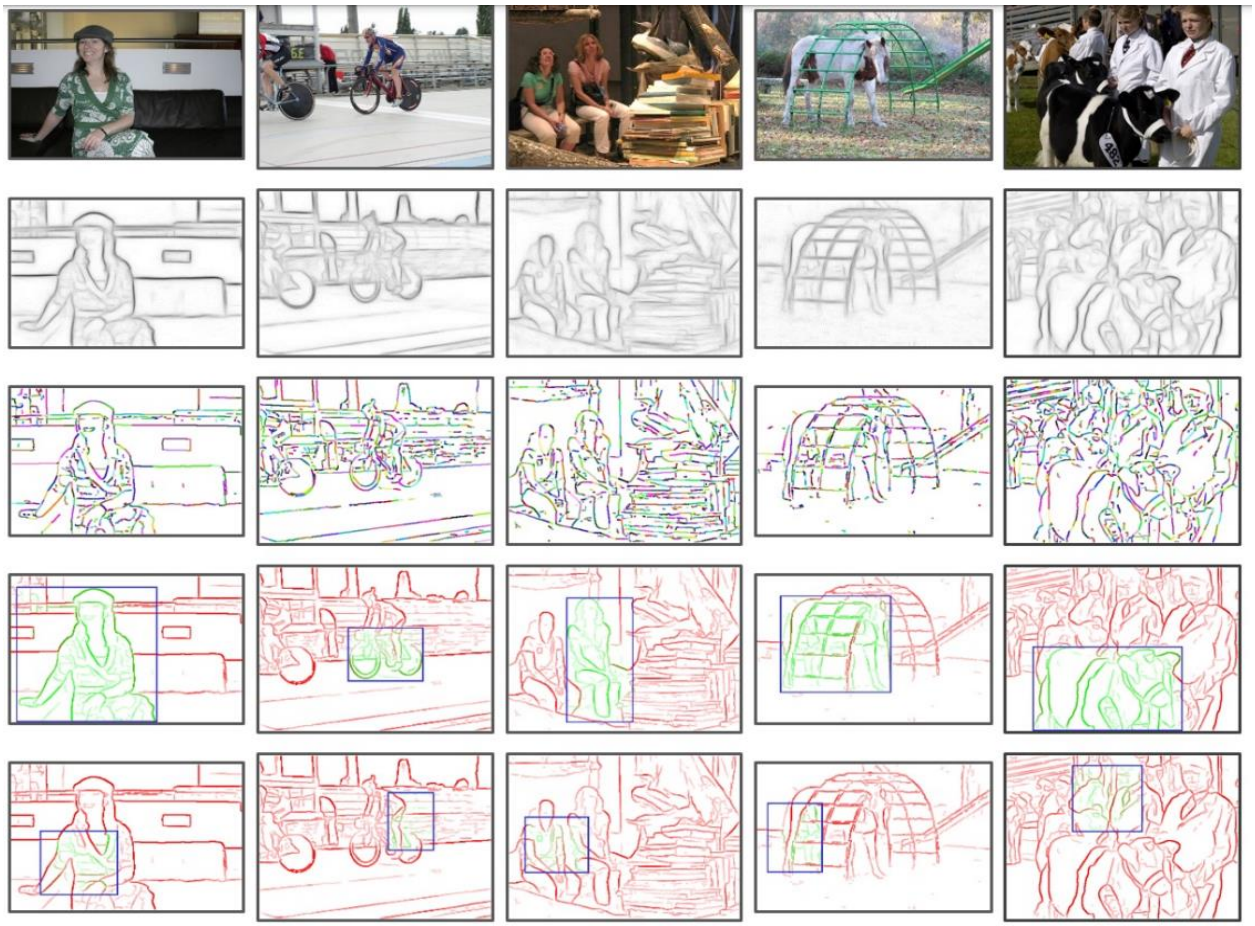


Рисунок 2.4 – Візуалізація роботи алгоритму Edge box

Застосовуючи алгоритм селективного пошуку ми користуємося ієрархічним групуванням за ознаками кольору, текстури, розміру та форми об’єкта. Для кожного пікселя зображення визначаємо параметри його ознак, потім будуємо граф, де вага кожного ребра дорівнює потужності пікселя за ознаками. Потім в результаті обходу графу ми групуємо пікселі за признаком того, що пікселі однієї групи повинні мати меншу вагу ребер між собою, тобто менше відрізнятися за ознаками. Після того як найбільш схожі регіони згруповані ми знову повторюємо цей процес, про те групуються вже не окремі пікселі, а області. Далі повторюємо процес доти поки все зображення не перетворюється в один сегмент.

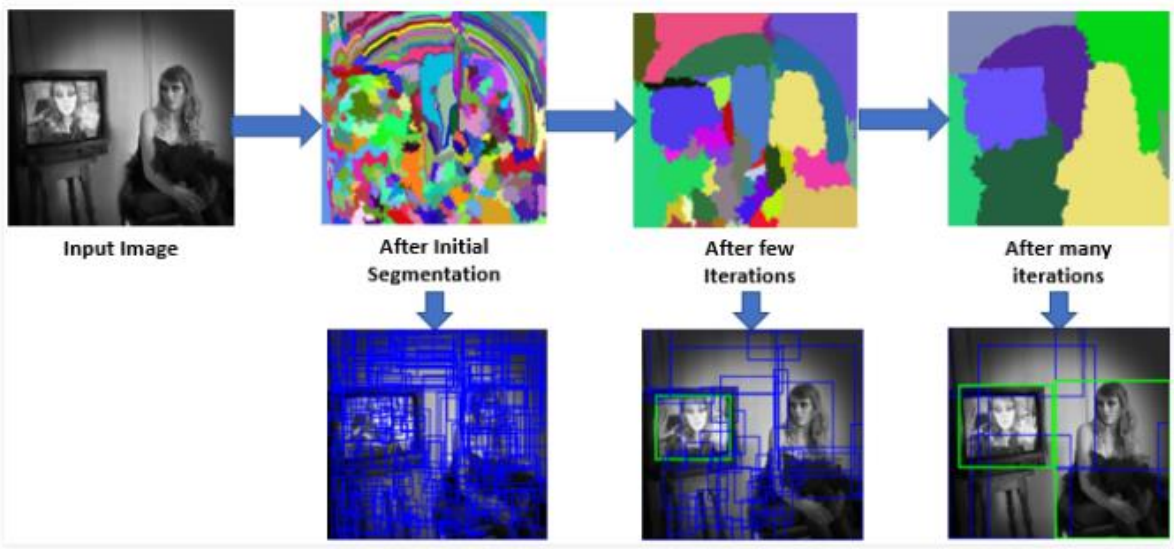


Рисунок 2.5 – візуалізація роботи алгоритму Selective search

Тобто селективний пошук це алгоритм знаходження сегментів на зображеннях з застосуванням техніки побудови та обходу графів залежності ознак пікселів на зображенні.

На кроці 2 знайдені області передаються до наступного шару мережі. Отримані області змінюють розмір та приводяться до розміру 227*227 пікселів, бо саме на такі зображення налаштована CNN мережа (див. Рисунок 2.6). Для того, щоб отримати квадратну область прямокутну область

доповнювали, а потім отримана квадратне зображення масштабували до необхідного розміру та подавали на вхід мережі.

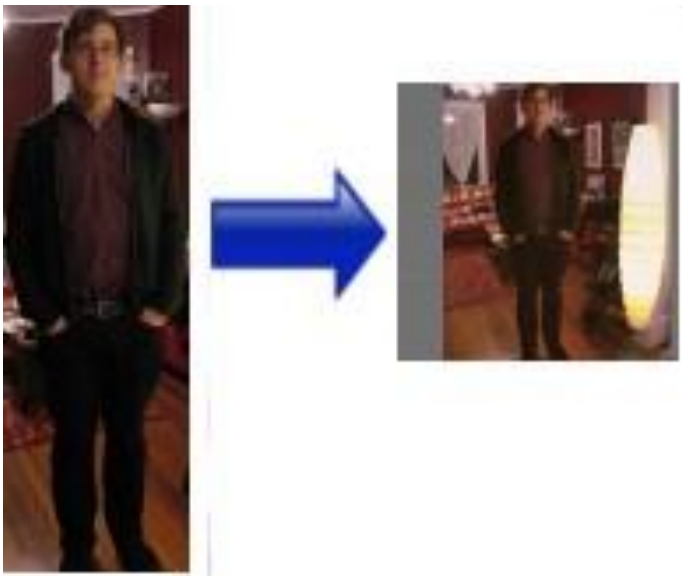


Рисунок 2.6 – трансформація зображень для обробки CNN мережею

Результатом обробки CNN мережі являє собою вектор з розмірністю в 4096 ознак для кожного об’єкту інтересу. Архітектура згорткової мережі виглядає наступним чином:

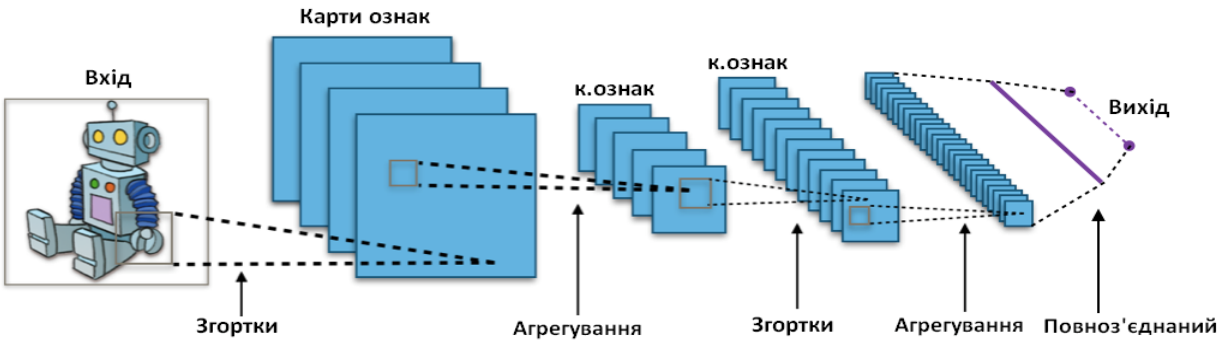


Рисунок 2.7 – Архітектура згорткової мережі

На кроці 3 проводиться класифікація отриманих векторів з використанням методу опорних векторів(SVM)[13].

Загальна архітектура роботи мережі R-CNN виглядає наступним чином:

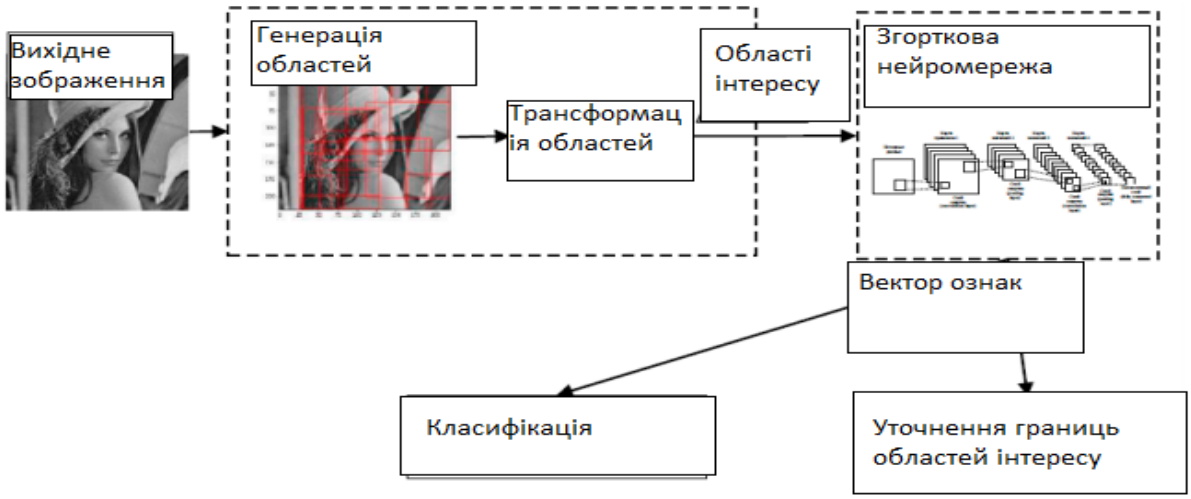


Рисунок 2.8 – Архітектура моделі сімейства R-CNN

Тепер розглянемо технологію трекінгу об’єктів. На відміну від алгоритмів детектування алгоритми трекінгу отримують координати вже детектованого об’єкту і займаються відстеженням переміщення об’єкту в кадрах відео. Алгоритми детектування та трекінгу працюють в парі, спочатку об’єкт інтересу детектується на відео за допомогою алгоритму детектування, потім отримані координати від детектора передаються на вхід до трекеру. Трекер може визначати положення об’єкту на основі поточних координат та інформації з поточного кадру. Такий підхід дозволяє проводити обробку і відслідковування об’єктів інтересу з високою частотою обробки кадрів, що є обов’язковою вимогою для обробки кадрів у real-time режимі.

Найпростішим підходом до відслідковування об’єкту є пошук його за певним шаблоном у кадрі. Пошук наступного положення об’єкту в новому кадрі виконується методом ковзаючого вікна з використанням деякої міри схожості об’єктів, для прикладу в якості міри схожості об’єктів може застосовуватися Евклідова відстань.

Faster-RCNN належить до четвертого покоління трекерів даного сімейства. Основна відмінність даного трекеру полягає в заміні алгоритму

Selective search, який використовується для сегментації зображення для розпізнавання на нейромережу Region Proposal Networks.

Детектор Faster-RCNN складається з декількох шарів та має наступну архітектуру:

На першому рівні використовується згорткова нейромережа(CNN)[14] для побудови карти ознак (див. Рисунок 2.9). Як слідує з назви дана мережа згортає розмірність вхідних даних при цьому кодуючи і залишаючи достатню інформаційну цінність.

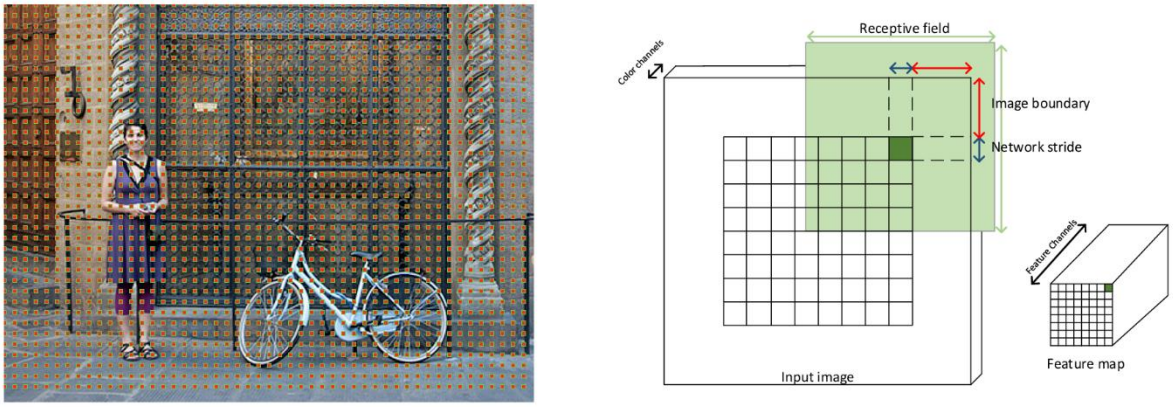


Рисунок 2.9 – Виділення областей інтересу

Результатом цієї операції є так звана матриця ознак. Кожному елементу матриці відповідає певне область зображення з довільними розмірами(див. Рисунок 2.14 –Рисунок 2.14). Мережа CNN має 3 шари згортання[15]:

- шар згортання
- шар активації
- шар пулінгу або субдескритизації
- шар повнозв’язної нейромережі

В процесі згортання над вхідним зображенням представленим у вигляді матриці значень вхідних пікселів проходять іншою матрицею з меншою розмірністю, цю матрицю називають фільтром або ядром згортки. При

накладенні матриці знаходиться скалярний добуток обраної частини вхідних і фільтру, результат записуємо в матрицю результату згортання.

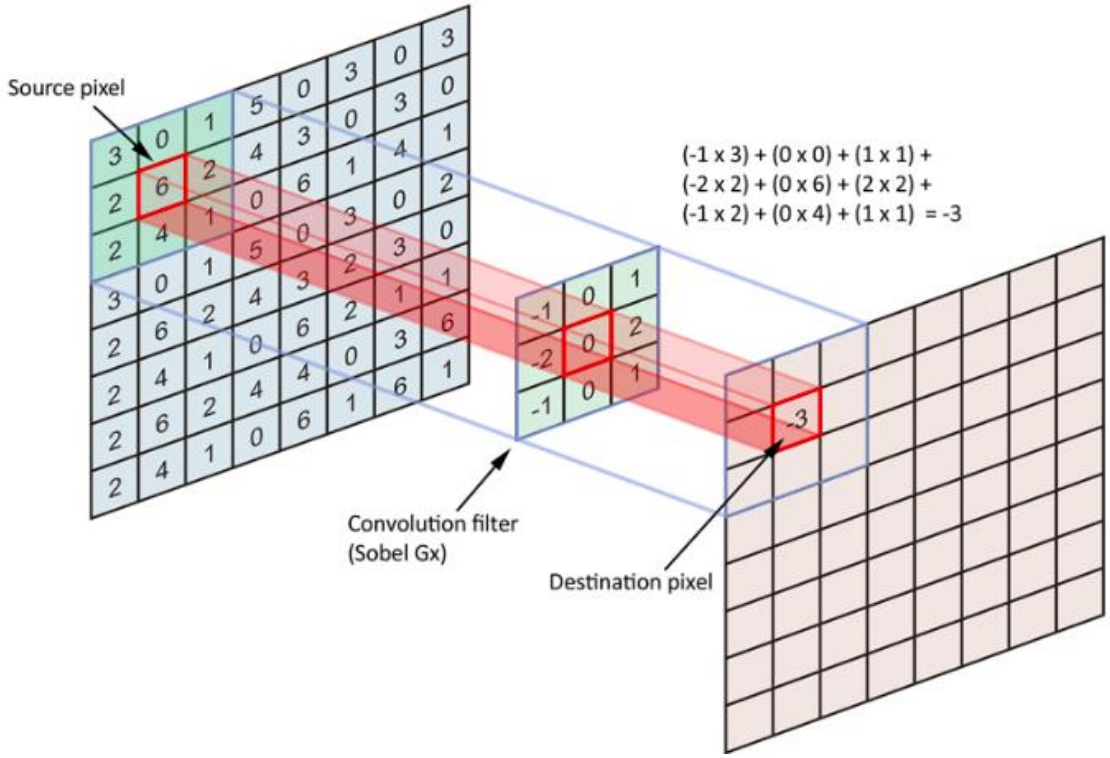


Рисунок 2.10 – Шар згортання

Для оптимізації кодування розмірність вихідної матриці може збільшуватися шляхом додавання до вихідної матриці рядків та стовпчиків, які заповнені нулями (див. Рисунок 2.11). Доповнені нулі не додають нову інформаційну цінність до вихідних даних, проте дозволяють закодувати вихідні комірки, які знаходяться на гранях матриці декілька разів.

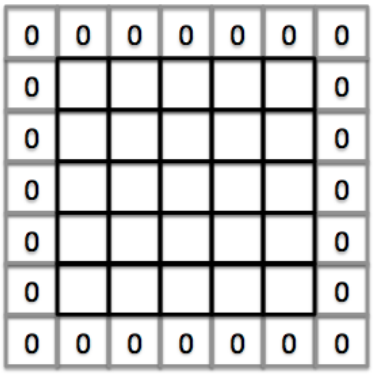


Рисунок 2.11 - Оптимізована вихідна матриця

Розмірність закодованої матриці від вихідної матриці розмірності N, можна обрахувати за формулою

$$M = \frac{(N + 2P - F)}{S + 1} \quad (2.1)$$

де M розмірність матриці результатів; N = розмірність вихідної матриці; F – розмірність матриці фільтра; P – кількість комірок заповнених нулями для оптимізації; S – крок для переходу фільтра.

В ході виконання шару активації над матрицею отриманою в результаті виконання попереднього кроку ми трансформуємо отриману матрицю за допомогою деякою функції активації. Існує доволі багато відповідних функцій активації (див. Рисунок 2.12), хоча найбільш часто використовуваною є ReLu(Rectified linear unit)[16]. Перевагою даної функції, що вона не може активувати всі нейрони одночасно, як видно з графіку даною функції вона перетворюю всі негативні значення в 0 та нейрон не активуються, що позитивно впливає на обчислювальну складність так як виключає непотрібні спрацювання.

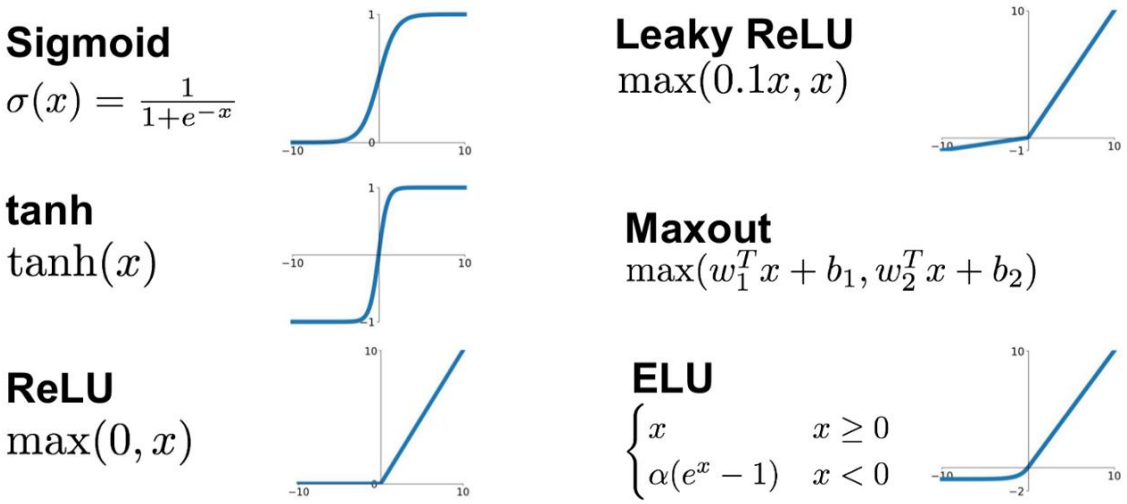
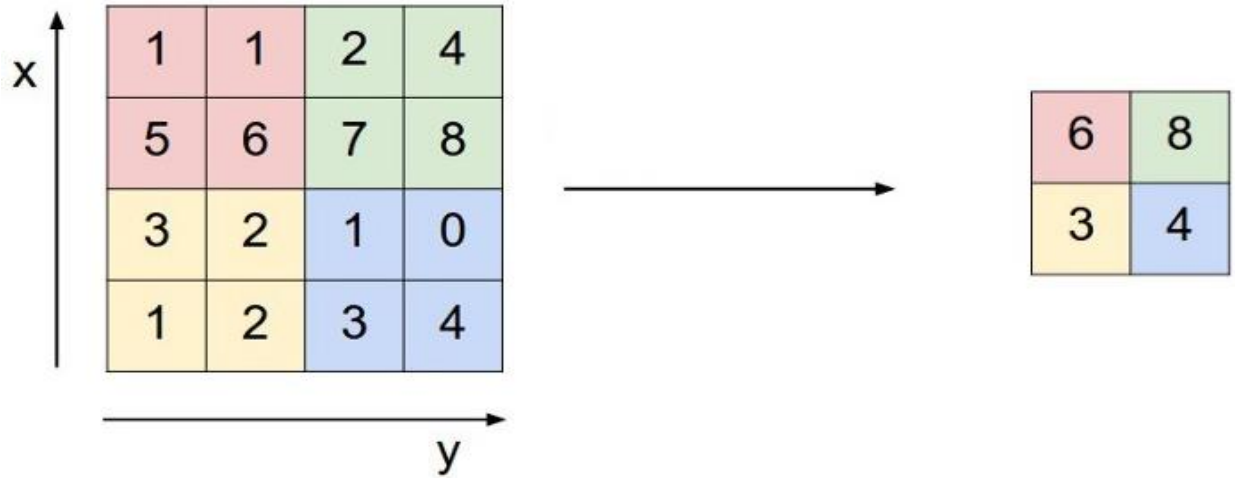


Рисунок 2.12 - функції активації

- крок пулінгу необхідний для зменшення розмірності матриці ознак, що допомагає боротися з процесом перенавчання нейромережі. Зазвичай на

цьому кроці використовується два підходи, пулінг по середньому та максимальному значенні. Коли використовується підхід вибору максимального, то матриця ознак ділиться на субматриці, для кожного з яких знаходиться максимальний елемент, який і потрапляє до матриці результатів. Метод пулінгу по середньому значенню відрізняється лише тим, що знаходиться середнє значення замість максимального.



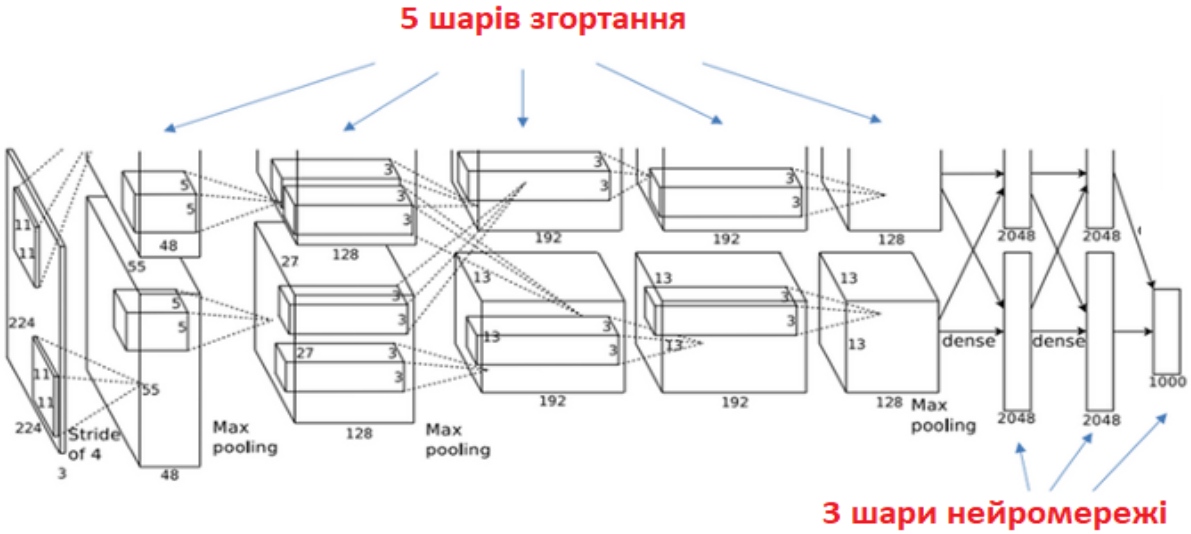
2.2 - Візуалізація пулінгу методом знаходження максимального

Розмірність вихідної матриці при застосуванню методу пулінгу можна знайти за формулою:

$$M = \frac{(N - F)}{S + 1} \tag{2.3}$$

де N розмірність вхідної матриці, F – розмірність субматриці для виконання пулінгу, S – крок з який застосовується операція пулінгу

- шар повноз’язної нейромережі працюю з матрицею ознак, яка отримане від використання попередніх 3 шарів, на даному кроці користуються нейромережею, яка має декілька шарів.



2.4 – Візуалізація архітектури згорткової нейромережі(CNN)

Існує доволі багато реалізації згорткових мереж. До найбільш точних та актуальних можна віднести:

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

Faster R-CNN для побудови карти ознак використовує реалізацію VGG-16, точність якої досягає за тестами 93%. Таку назва дана мережа має через те, що складається з 16 шарів(див. Рисунок 2.13). Для виконання задачі отримання карти ознак детектор Faster R-CNN забирає дані після виконання шару conv-5.3, що означає що карта ознак не потрапляє до вбудованих в VGG-16 нейромереж.

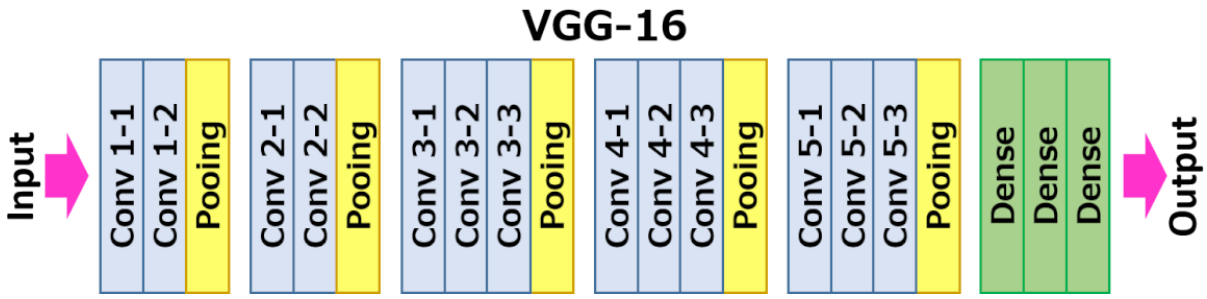


Рисунок 2.13 – візуалізація згорткової нейромережі VGG-16

Після до знаходження карти ознак виконується пошук областей інтересу з використанням нейромережі Region Proposal Network. Це нова концепція до знаходження областей інтересу, що значно підвищує швидкість роботи детектору.

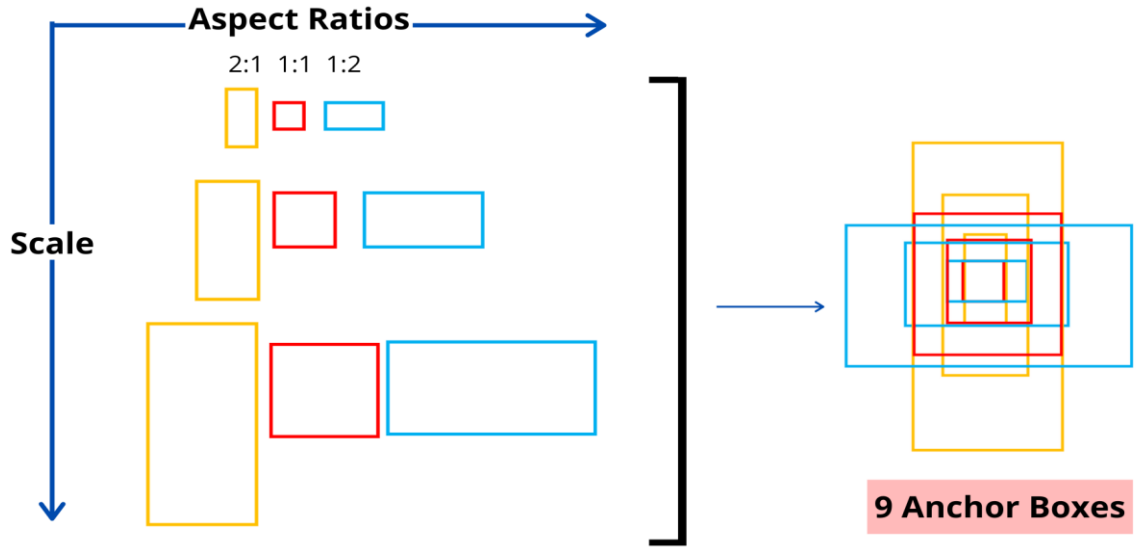


Рисунок 2.14 – Побудова областей якорів за допомогою RPN

На вхід дана мережа приймає зображення і повертає координати прямокутників, де прогнозовано знаходяться об’єкти та імовірність знаходження об’єктів там. RPN не працює з вихідним зображенням, на вхід отримується карта ознак від згорткової мережі на основі якої будуються так звані якорні зони (див. Рисунок 2.14). Навколо таких зон визначаються прямокутники з різними розмірами та співвідношенням сторін. По замовчуванню генерується по 3 варіанти на ознаку ширини, довжини та

співвідношення сторін, тобто 9 областей на кожну область інтересу[17]. Для кожної з областей інтересу знаходяться показники метрики IoU(метрика степеню перетину між двома областями (див. Рисунок 2.15).

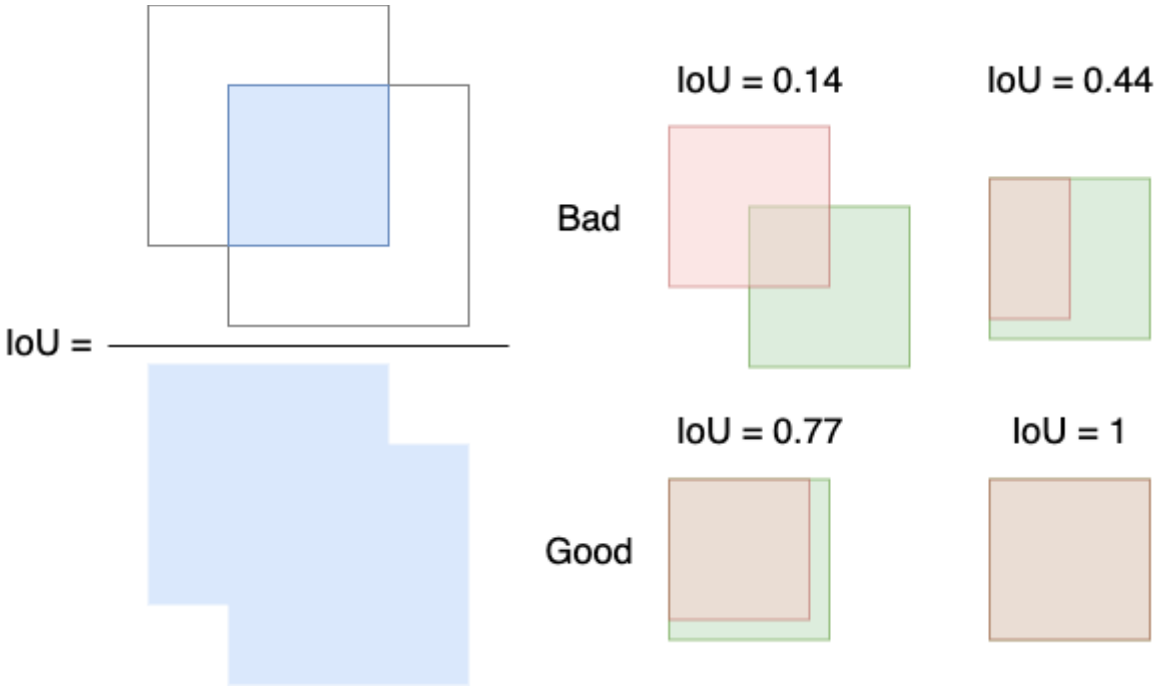


Рисунок 2.15 – візуалізація критерію IoU

У випадку якщо значення IoU більше або рівне 0.7, область якоря класифікується як позитивний випадок, при значенні 0.3 і менше область класифікується як негативний випадок. Дане правило можна виразити наступним чином:

$$p_i^* = \begin{cases} if\ IoU > 0.7, than\ 1 \\ if\ IoU < 0.3\ than\ 0 \\ nothing \qquad otherwise \end{cases} \tag{2.5}$$

де p_i^* - правильний номер класу

Так як RPN і знаходить імовірні координати і класифікує об’єкти в області інтересу, то для вирішення цих задач використовуються дві різні мережі. Для мережі RPN функція втрат набуває наступного вигляду:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{loc}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \qquad (2.6)$$

де i – номер якорю, p_i – імовірність знаходження об’єкта в якорі *i*; p_i^* - правильний номер класу; t_i – прогнозовані координати, t_i^* - справжні координати; { p_i }, { t_i } – висновки класифікаційної та регресійної моделі відповідно; λ – коефіцієнт налаштування балансу між класифікацією та регресією.

На наступному кроці працює мережа Region Of Interest Pooling, яка необхідна для об’єднання карти ознак та областей інтересу отриманих на попередньому кроці(див. Рисунок 2.16). Наклавши на області інтересу знайдені мережею RPN карту ознак отриману від згорткової мережі ми отримаємо області інтересу з коефіцієнтами отриманими від згорткової мережі, після чого на виконується операція пулінгу за максимальним значенням. Перевагою даного підходу є те що карта ознак використовується повторно, а не кожного разу вираховується для області інтересу.

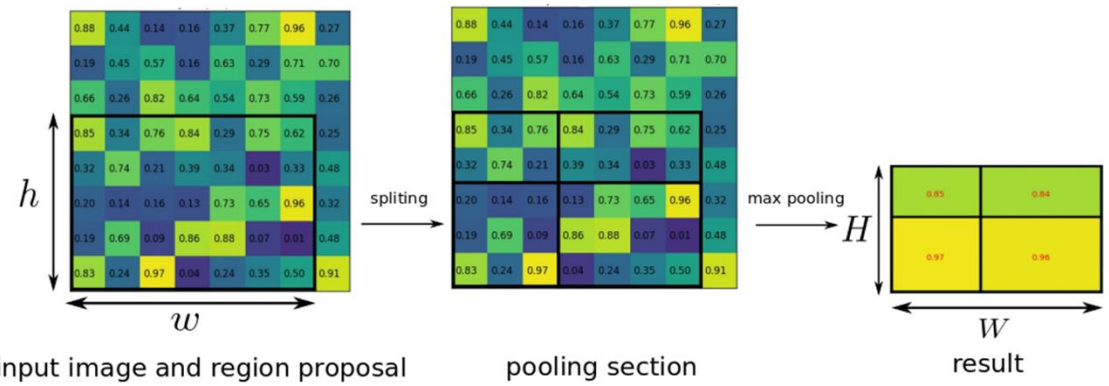


Рисунок 2.16 – Демонстрація роботи шару RoI

Результат роботи RoI передається до двох повнозв’язних нейромереж, одна з яких займається уточненням координат положення об’єкта, а друга класифікацією розпізнаних об’єктів. Для оптимізації швидкості роботи обробка вхідних даних за допомогою класифікатора та регрессора відбувається паралельно.

2.2. Вибір мови програмування для вирішення задачі

За статистикою найбільш популярними мовами для вирішення задач машинного навчання є[18]:

- Python
- C++
- JavaScript
- Java
- C#

Python вже давно зайняв нішу мови по замовчуванню в задачах машинного навчання та обробки масивів даних. Це мова високого рівня, що орієнтується на швидкість розробки ПО та зручність підтримки та написання коду. Дана мова була розроблена Гвідо ван Россумом ще в 1991 році, в якості допоміжної скриптової мови для операційної системи Amoeba. Python є інтерпретованою мовою, що виконується на однойменній машині PythonVM. Для підвищення швидкості виконання коду розроблені альтернативні реалізації на базі JVM та .Net/CLR. Також існує реалізація компілятора мови Python в C/C++ код, що має назву Cython. Використовуючи Cython можна не лише підвищити швидкість виконання за рахунок компіляції, а також легко викликати код написаний на мовах C та C++ без використання додаткових абстракцій. Окрім використання додаткових платформ чи компіляторів є популярною практикою використання надбудов над бібліотеками на мовах C, C++. Поєднання зрозумілого коду на мові Python з якісними бібліотеками на C, зробило Python найбільш популярною мовою для обробки даних. Для прикладу одна з найбільш популярних бібліотек NumPy якраз і є реалізацією подібної архітектури.

Дана мова має такі популярні інструменти для задач машинного навчання як TensorFlow, Keras, PyTorch, scikit-learn, OpenCV.

JavaScript – мова програмування, яка спочатку з’явилася в якості вбудовуваної мови для браузера NetScape. Згодом дана мова з’явилася у всіх реалізація веб-браузерів, отримала свій стандарт Ecma-262 та вийшла за рамки браузерів. Сучасний JavaScript може є популярною мовою для вбудовуваних систем, для прикладу веб-сервер Nginx пропонує власну реалізацію JS для побудови додатків. Навіть Microsoft Office дозволяє будувати макроси написані на мові JavaScript. Завдяки можливості виконання JS в різних середовищах легко можливо переносити один і той же код. Цим часто користуються і в машинному навчанні. Навчивши модель для прикладу в середовищі Node можна перенести її в користувацьке середовище клієнта.

Серед популярних інструментів екосистеми JS для задач машинного навчання слід відзначити Keras.JS, Tensorflow.JS, Neuro.JS, ML5.js, ConvNetJS.

C++ — третє за популярністю мова серед проектів машинного навчання. Проте дана мова є найбільш популярною для реалізації низькорівневих бібліотек та фреймворків в машинному навчанні. Для прикладу Tensorflow, Keras та OpenCV екосистеми Python та JS побудовані саме на базі C++ коду. Сама є мова розроблена Бєрном Страуструпом ще в 1980-х роках як розширення для мови C. Порівняно з Python чи JS мова C++ доволі низькорівнева. Ця мова не має автоматичного виділення та очищення пам’яті. До того ж ця мова компілюється в нативний бінарний файл для платформи на відміну від JS та Python і виконується без віртуальних машин. Швидкість виконання даної мови вища в десятки разів від мови Python чи JS. Проте написання коду на мові C++ вимагає більших трудовитрат та контролю від розробника, тому прототипи та перевірки гіпотез будують на інших мовах.

Для вирішення поставленого завдання було вирішено використати мову Python, як найбільш популярну і просту мову для побудови і перевірки гіпотез.

2.3. Вибір фреймворку машинного навчання

PyTorch – фреймворк для машинного навчання, що розробляється та підтримується компанією Facebook. Розробка розпочата ще в 2016 році та на даний момент актуальної версією 1.8.0. В основному фреймворк спеціалізується на тензорних обчисленнях та оптимізації цих обчислень. Для роботи з тензорними величинами фреймворк має спеціальну бібліотеку подібну до NumPy. Виконання обчислень можливе як на CPU, та і на GPU, але лише на відеокартах Nvidia з підтримкою архітектури CUDA. Фреймворк має спеціальні версії на для мобільних платформ Android та iOS.

Keras – фреймворк побудований на основі Theano, Deeplearning4j, TensorFlow і націлений на розробку мереж глибокого навчання. На відміну від інших фреймворків інтерфейс для розробників є доволі високорівневим і насправді є лише надбудовою над іншими фрейморками. Сам фреймворк повністю написаний на мові Python, так як використовує низькорівненіві реалізації інших засобів. Є підтримка обчислень на GPU з підтримкою одночасно декількох пристроїв. Розробникам подобається даний фреймворк за широкі можливості в модульності, зручності для користувача та розширюваності. Розробкою в основному займаються такі компанії як Microsoft, Amazon, Google та Nvidia.

Tensorflow – є найбільш використовуваною бібліотекою для машинного навчання. Розробкою займається компанія Google. На даний момент даний фреймворк можна використовувати з мовами R, C#, Haskell, Java, Go, Swift, JavaScript. Хоча основний API фрейморку написаний на мові Python. Для оптимізації швидкості виконання низько рівневі компоненти написані на мові C++. Для виконання обчислень наявна підтримка GPU з технологією CUDA. За назвою можна зробити висновок, що фреймворк добре спеціалізується на виконанні тензорних обчислень так же як і PyTorch. Для оптимізації компанія Google розробила спеціальні інтегральні схеми, які оптимізовані для завдань TensorFlow.

Scikit-learn – це бібліотека на мові Python для вирішення задач машинного навчання, яка розробляється відкрито, без участі великих компаній. Сама бібліотека написана на мові Python та використовує бібліотеку NumPy для високоефективних обчислень лінійної алгебри та операцій над векторами. Для оптимізацій швидкості виконання деякі компоненти написані з використанням Cython. Для візуалізації результатів бібліотека має інтеграції з такими бібліотеками як Plotly, MatPlotLib, Pandas, Scipy.

Для вибору правильного інструменту розглянемо деяку статистику

Таблиця 2.2 – Порівняння інструментів машинного навчання

Фреймворк	**Кількість зірочок**	**Кількість Issue**	**Кількість Pull-request**	**Якість документації**	**Кількість комітів**
Tensorflow	155k	3.8k	198	4.7	108k
Keras	51k	3.2k	22	4.2	5.5k
PyTorch	47k	5k+	2.6k	4.5	35k
Scikit-learn	45k	1.6k	741	4	45k

Логіка побудови даної таблиці наступна. Кількість зірочок свідчить про задоволеність користувачів від використання та популярності даного інструменту. Це один з найважливіших показників для вибору. Кількість Issue свідчить про кількість проблем, помилок чи незрозумілих ситуацій при використання інструменту. Тут чим вище число тим гірше. Кількість Pull-request свідчить про активність розробки інструменту. Загалом чим вища число тим краще, проте треба окремо аналізувати типи цих пул реквестів. Якщо в основному там виплавлення помилок, то це не свідчить про якість інструменту. Якість документації це особиста оцінка наданої документації від розробників інструменту. Залежить від зрозумілості опису, повноти інформації, зручності пошуку, кількості прикладів реалазацій. Кількість комітів це насамперед показник історії розвитку проекту. Якщо проект розвивається достатньо довго, то у нас багато комітів. Для нових проектів кількість комітів незначна.

Серед проаналізованих інструментів вигідно відрізняється Tensorflow, він перший майже за всіма показниками. Даний проект розробляється компанією Google вже понад 10 років. За цей час було вирішено безліч задач та проблем з цим інструментом. Тож для реалізації нашого завдання було обрано Tensorflow.

2.4.Огляд додаткових інструментів реалізації

Google Collab – це безкоштовний інструмент від компанії Google (див. Рисунок 2.17), який дозволяє інтерактивно працювати з інструментами з документами формату IPynb**.** Загалом даний сервіс надає в безкоштовне використання віртуальну машину, що надає 60Гб постійної пам'яті та 12 Гб Ram для виконання користувацького коду. По замовчування можна виконувати лише Python код та bash команди, хоча можливо налаштувати середовище виконання Javascript. Данні для роботи можна завантажувати з Google Drive чи Github. Також можна увімкнути середовище виконання типу GPU чи TPU(див. Рисунок 2.18) для виконання коду, що вимагає значних обчислювальних ресурсів.

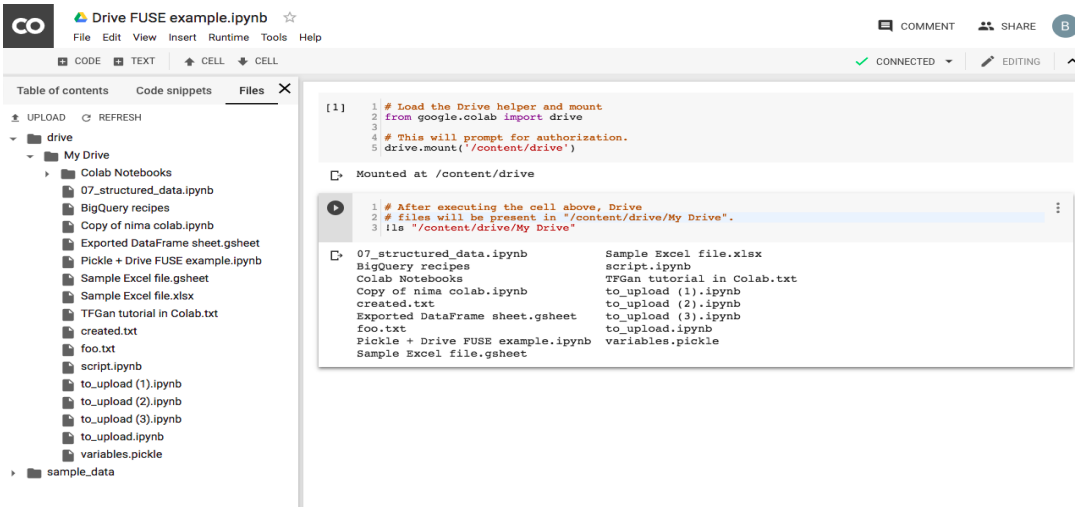


Рисунок 2.17 – середовище виконання Google Collab

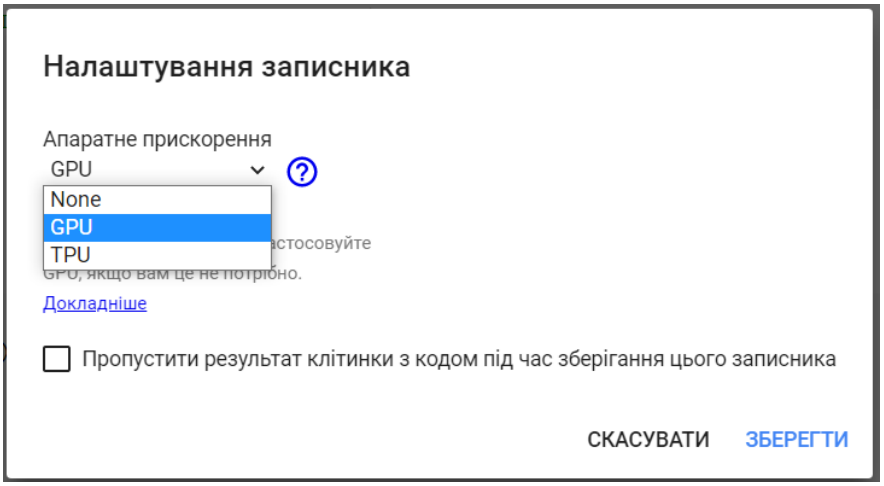


Рисунок 2.18 – налаштування середовища виконання

В режимі виконання GPU надається безкоштовна віртуальна машина з налаштованою відеокартою Nvidia Tesla K80, потужність якої складає 2.91 терафлопс(одиниця вимірювання кількості операцій за секунду) та має 4992 CUDA ядра. CUDA (Compute Unified Device Architecture) це архітектура для відеокарт компанії Nvidia, що дозволяє виконувати швидкі паралельні обчислення. За рахунок того що звичайний центральний процесор має лише від 1 до 16 ядер, а відеокарта має декілька сотень чи тисяч ядер можливо виконувати набагато більше операцій за одиницю часу. Проте за рахунок особливостей архітектури CUDA можливо виконувати лише арифметичні операції з такою відеокартою. Архітектура CUDA знайшла широке використання в задачах машинного навчання через можливості швидкого виконання паралельних операцій.

В режимі виконання TPU до віртуальної машини підключений спеціалізований процесор, який побудований на спеціальній архітектурі для виконання задач з низькою точністю, зазвичай лише до 8 знаків після коми. Проте швидкість виконання подібної схеми значно вища при меншому використанні електроресурсів. Розробкою даної архітектури займається компанія Google, яка і надає їх для використання в Google Collab. Одним з найбільш відомих проектів, що побудований на платформі TPU можна назвати штучний інтелект для гри в Go AlphaGo.

Для проведення експериментів був використаний Google Collab з середовищем GPU, так я технологія TPU, ще не достатньо поширена в реалізаціях бібліотек машинного навчання.

LabelImg — інструмент для графічного розмічення зображень для навчання. Даний інструмент побудований на мові Python з використанням фреймворку Qt. LabelImg має простий та інтуїтивний інтерфейс (див. Рисунок 2.19). За допомогою відповідних кнопок на панелі інструментів ми можемо відкрити одне чи відразу директорію з зображеннями для розмічення даних. За допомогою кнопки додавання області можна виділити

об’єкт інтересу(див. Рисунок 2.20). Таких областей може бути довільна кількість. Потім можна зберегти отримане зображення в форматі .xml. В збереженому файлі можна знайти координати виділених прямокутних областей в фарматі (xmin, xmax, ymin, ymax), для кожної області вказаний клас, а також мета інформацію про зображення, яке розмічалося даним програмним засобом.

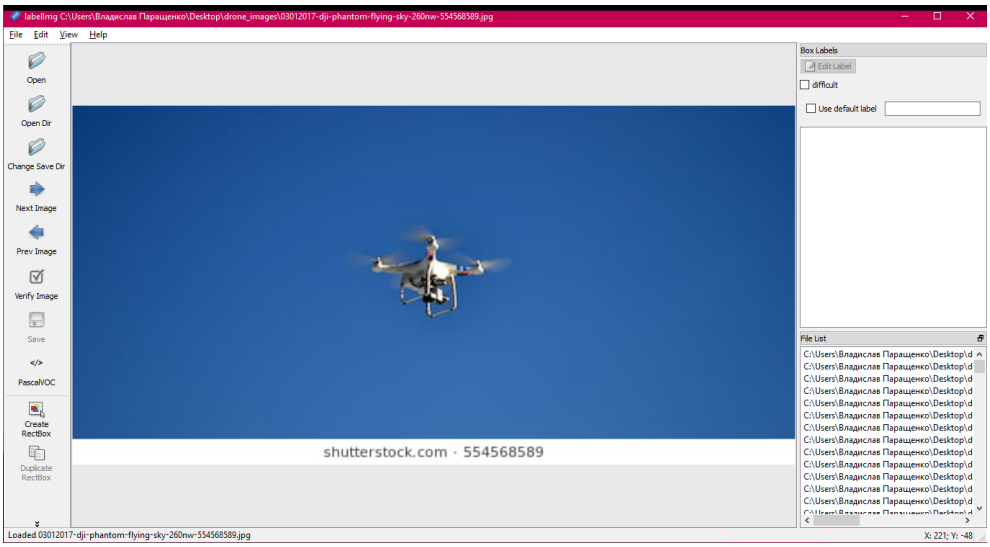


Рисунок 2.19 – інтерфейс додатку LabelImg

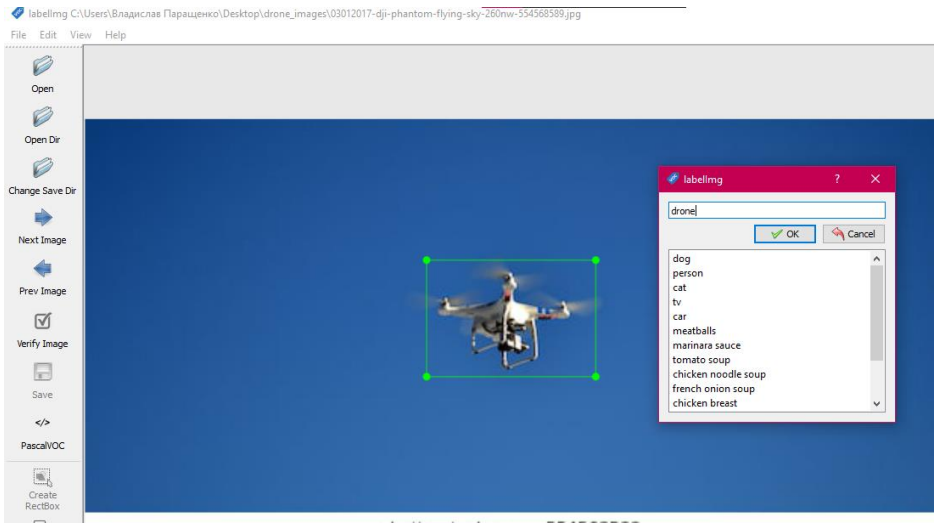


Рисунок 2.20 – Додавання області за допомогою LabelImg

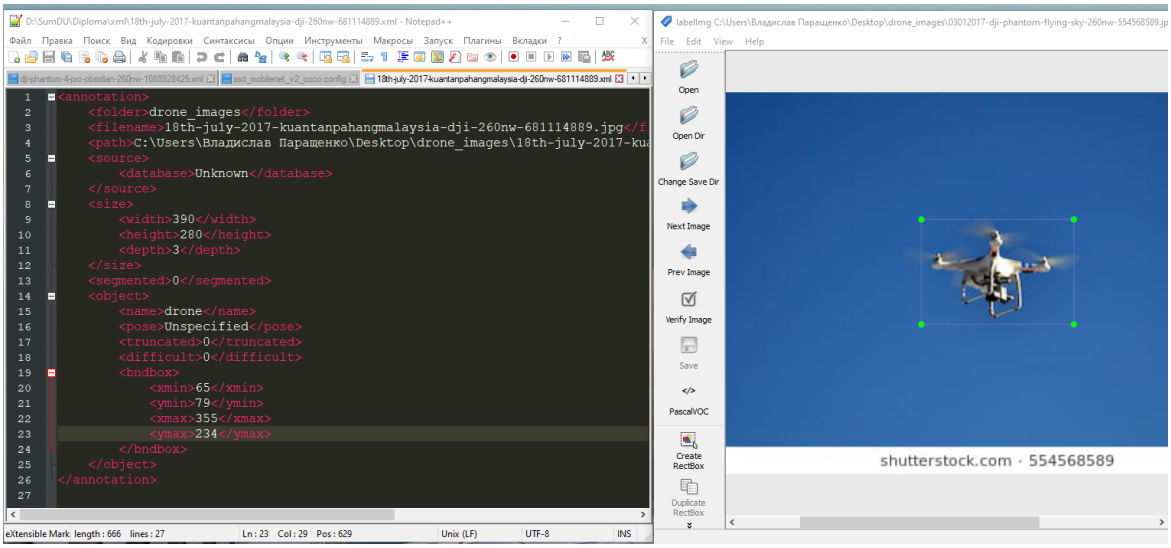


Рисунок 2.21 – Результат роботи LabelImg

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

Для розробки системи детектування та моніторингу було вирішено випробувати 2 моделі з різними налаштуваннями:

- Faster RCNN
- SSD

Для розробки було використано Python версії 3.9, Google Collab в якості середовища виконання та Visual Studio Code в якості інструменту для написання скриптів підготовки даних. Заздалегідь навчені моделі були взяті з відкритого репозиторію Github[19]. Перевірка якості отриманих результатів проводилася за показника.

- Імовірність детектування
- Точність IoU
- Швидкість виконання

Для візуалізація отриманих результатів була використана бібліотека pandas.

3.1. Підготовка вхідних даних

Для навчання було відібрано 2000 зображень, які були розмічені для навчання за допомогою інструменту LabelImg. Було підготовлено 2 .zip архіви з фото та .xml файлами розмітки(див. Рисунок 3.1)., які було завантажено на Google Drive.

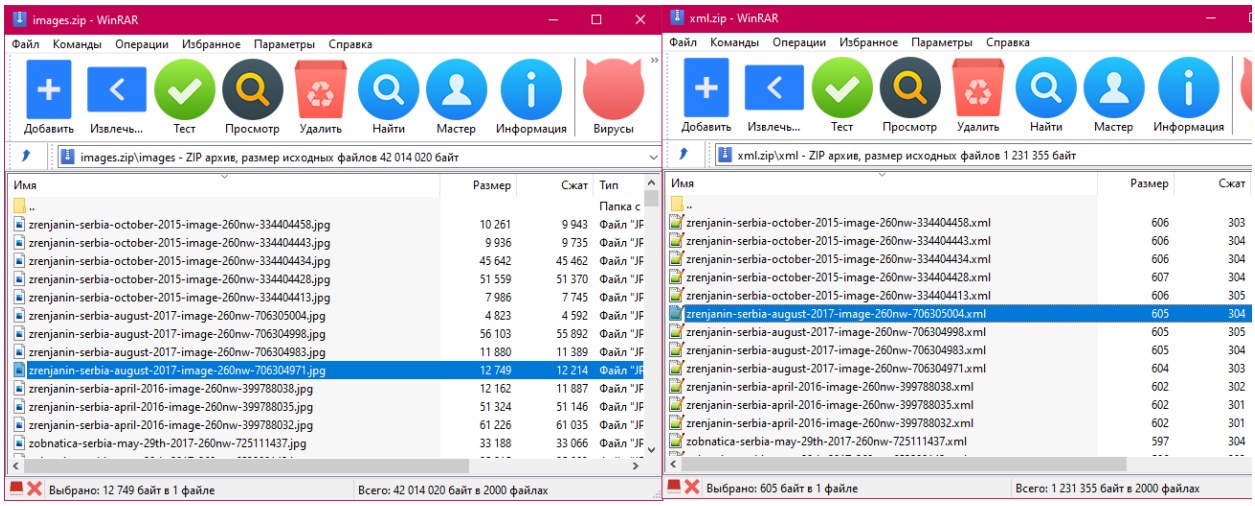


Рисунок 3.1- дані для навчання моделі

Підготовлений датасет був розділений у співвідношенні 80 до 20, на дані для тренування моделі та валідації.

3.2. Робота з підготовки вхідних даний

Для навчання спочатку необхідно підготувати правильну файлову структуру директорій. Для цього скористаємося можливостями виконання Bash команд на Google Collab (див. Рисунок 3.2.). Тут розділяється вхідні зображення та файли нотацій на дві різні директорії.

```
!rm -rf data
!mkdir data

!mkdir data/images data/train_labels data/test_labels


!cp images/* data/images

!cp xml/* data/train_labels

!ls data/train_labels/* | sort -R | head -400 | xargs -I{} mv {} data/test_labels

!ls -1 data/train_labels/ | wc -l

!ls -1 data/test_labels/ | wc -l
```

Рисунок 3.2 – виконання підготовки файлової структури

До процесу навчання необхідно підготувати файл .pbtxt, який містить очікувані для детектування номери класів та їх назви. Для поточного експерименту файл містить наступні класи (див. Рисунок 3.3).

```
item {
    id: 1
    name: 'drone'
    display_name: 'Drone'
 }
```

Рисунок 3.3 - файл конфігурації вхідних класів

Також необхідно перетворити файли вхідних нотацій в два .csv файли train_labels.csv та test_labels.csv, які містять всю необхідну інформацію. Для цього використаємо спеціально підготовлену функцію xml_to_csv, яка читає всі файли директорії за шаблоном *.xml, будує з вмісту файлу дерево xml представлення документу, а потім записує значення необхідних атрибутів в рядки відповідного csv файлу (див. Рисунок 3.4).

```
def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    file_name = xml_file.split('/')[1]
    name = file_name.split('.')[0]
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (name+'.jpg',
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
               int(member[4][1].text),
               int(member[4][2].text),
               int(member[4][3].text))
      xml_list.append(value)
  column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
  xml_df = pd.DataFrame(xml_list, columns=column_name)
  classes_names = list(set(classes_names))
  classes_names.sort()
  return xml_df, classes_names

for label_path in ['train_labels', 'test_labels']:
  image_path = os.path.join(os.getcwd(), label_path)
  xml_df, classes = xml_to_csv(label_path)
  xml_df.to_csv(os.path.join(ROOT_PATH, f'{label_path}.csv'), index=None)
  print(f'Successfully converted {label_path} xml to csv.')

label_map_path = os.path.join("label_map.pbtxt")

pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n    display_name: 'Drone'\n }}\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
```

Рисунок 3.4 – Функція підготовки вхідних даних

Далі необхідно безпосередньо підготувати файли, які будуть передані на вхід до наших моделей. Так як для навчання використовується фреймворк TensorFlow, то ми скористаємося форматом TfRecord[20]. Даний формат файлу був представлений у 2015 році, і являє собою формат, який складається з послідовності рядків закодованих у бінарному вигляді. Перевагою даного формату є зменшення розміру даних на диску, завдяки зберіганню їх в бінарному вигляді, легке поєднання декількох файлів між собою, можливість послідовного читання, що дозволяє не зберігати величезні структури даних в оперативній пам’яті, а послідовно читати необхідні дані з файлу. Для створення файлів tfrecord було розроблено спеціальний скріпт на мові Python (див. Рисунок 3.5).

```
def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example

for csv in ['train_labels', 'test_labels']:
  writer = tf.io.TFRecordWriter(DATA_BASE_PATH + csv + '.record')
  path = os.path.join(image_dir)
  examples = pd.read_csv(DATA_BASE_PATH + csv + '.csv')
  grouped = split(examples, 'filename')
  for group in grouped:
      tf_example = create_tf_example(group, path)
      writer.write(tf_example.SerializeToString())

  writer.close()
  output_path = os.path.join(os.getcwd(), DATA_BASE_PATH + csv + '.record')
  print('Successfully created the TFRecords: {}'.format(DATA_BASE_PATH +csv + '.record'))
```

Рисунок 3.5 – створення файлів tfrecord для навчання моделі

В результаті ми маємо два файли train.record та test.record, які подаються на вхід для навчання.

3.3. Підготовка файлів конфігурації моделей

Тепер потрібно підготувати файли конфігурацій. Це спеціальний файл, що містить налаштування у текстовому вигляді, який придатний для читання людиною(див. Рисунок 3.6).

```
model {
  ssd {
    num_classes: 90
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
train_config: {
  batch_size: 24
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/model.ckpt"
  fine_tune_checkpoint_type:  "detection"
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_train.record-?????-of-00100"
  }
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
}
```

Рисунок 3.6 – файл конфігурації моделі

Файл містить доволі багато параметрів(див. Таблиця 3.1), які можна налаштовувати для отримання кращих результатів.

Таблиця 3.1 – основні параметри конфігурації процесу навчення

Параметр	**Опис**
num_classes	Кількість класів розпізнавання, повинно збігатися з кількістю класів файлу .pbtxt
num_steps	Кількість кроків навчання
train_input_reader.tf_record_input_reader. input_path	Шлях до tfrecord файлу для тренування
train_input_reader. label_map_path	Шлях до файлу .pbtxt для навчання
test_input_reader.tf_record_input_reader. input_path	Шлях до tfrecord файлу для тестування
test_input_reader. label_map_path	Шлях до файлу .pbtxt для тестування
fine_tune_checkpoint	Шлях до попередньо файлу, який зберігає стан моделі останнього вдалого навчання
max_total_detections	Максимальна кількість знайдених об’єктів
max_detections_per_class	Максимальна кількість знайдених об’єктів одного класу
score_threshold	Мінімальний показник імовірності знаходження об’єкту(від 0 до 1)
iou_threshold	Показник перетину об’єктів, потрібен для відкидання зон, що дублюються

В якості експерименту до файлів конфігурації наших моделей додамо налаштування для використання техніки аугментації. Для цього просто додамо відповідні параметри до файлів конфігурацій. Підібраний конфіг випадково трансформує вхідне зображення. Для прикладу з імовірністю в 50% може виконатися поворот зображення по осі у чи зміниться яскравість вхідного зображення з імовірністю в 20%.

```
data_augmentation_options {
  random_horizontal_flip {
    probability: 0.5
  }

  random_vertical_flip {
    probability: 0.5
  }

   random_rotation90 {
    keypoint_rot_permutation: 3
    keypoint_rot_permutation: 0
    keypoint_rot_permutation: 1
    keypoint_rot_permutation: 2
    probability: 0.5
  }

  random_rgb_to_gray {
    probability: 0.8
  }

  random_adjust_contrast {
    min_delta: 0.7
    max_delta: 1.1
  }

  random_adjust_saturation {
    min_delta: 0.75
    max_delta: 1.15
  }

  random_image_scale {
    min_scale_ratio: 0.8
    max_scale_ratio: 2.2
  }

  random_adjust_brightness {
    max_delta: 0.2
  }
}
```

Рисунок 3.7 – налаштування для використання техніки аугментації

3.4. Результат

Для тестування отриманих моделей проведемо експерименти з розпізнавання 10 відео по 500 кадрів в кожному. Тестування також проводимо в середовищі Google Collab. Тестування проводимо за наступники показниками:

- **Precision** – співвідношення між кількістю позитивно розпізнаних об’єктів та дійсною кількістю позитивних прикладів
- **Recall – х**арактеризує, яку долю позитивних прикладів вдалося розпізнати
- **IoU** – міра перетину між очікуваним та отриманими областями детектування
- **Швидкість виконання**

Дані для тестування також завантажимо на Google Drive, як і в кроці навчання.

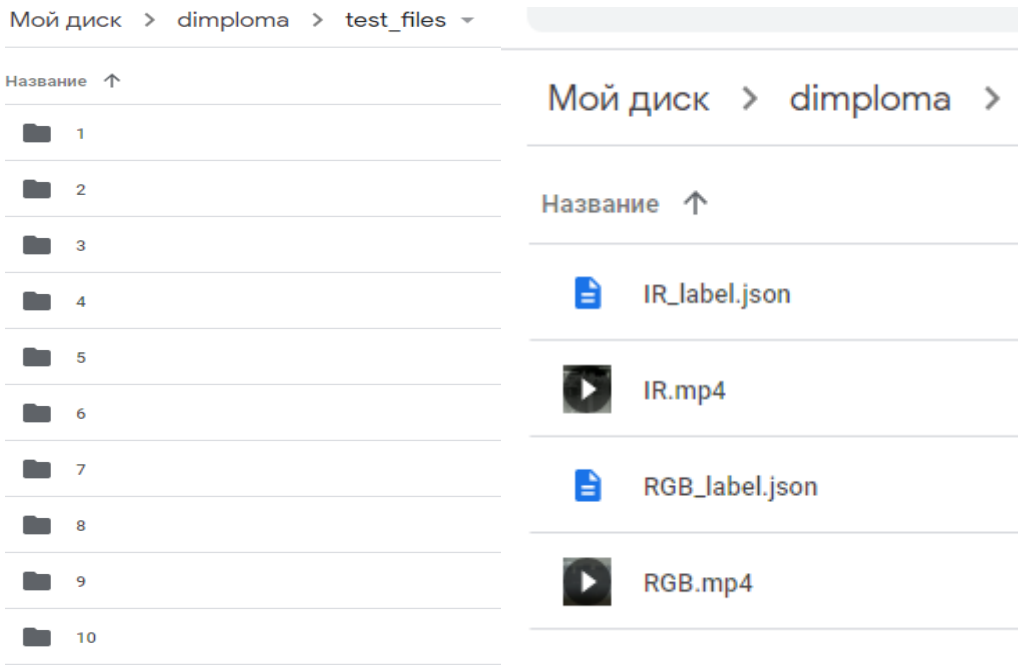


Рисунок 3.8 – Дані для проведення тестування на Google Drive

Для аналізу якості роботи моделей введемо дві допоміжні функції. Перша буде відповідати за розрахунок показника IoU (див. Рисунок 3.9), на вхід дана функція приймає два кортежі координат та знаходить міру перетину двох об’єктів.

```
def calculate_intersection_over_union(boxA, boxB):
  xA = max(boxA[0], boxB[0])
  yA = max(boxA[1], boxB[1])
  xB = min(boxA[2], boxB[2])
  yB = min(boxA[3], boxB[3])

  interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
  boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
  boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

  iou = interArea / float(boxAArea + boxBArea - interArea)

  return iou
```

Рисунок 3.9 – Розрахунок показника IoU

Друга функція слугує для того щоб рахувати правильні та неправильні розпізнавання моделей (див. Рисунок 3.10).

```
def calculate(option_values, received_value, expected_value):
  (tp, tn, fp, fn) = option_values
  if received_value == 1 and expected_value == 1:
    tp += 1
  if received_value == 0 and expected_value == 0:
    tn += 1
  if received_value == 1 and expected_value == 0:
    fp += 1
  if received_value == 0 and expected_value == 1:
    fn += 1

  return (tp, tn, fp, fn)
```

Рисунок 3.10 – підрахунок кількості правильних та неправильних детектувань

Для детектування додамо відповідну функцію, яка приймає об’єкти Tensorflow: граф та сессію. Граф це представлення даних моделі, сессія — визначає, які операції необхідно виконати над графом. Зображення представлено у вигляді трьох мірного масиву numpy, який зберігає дані про

кожен піксель вхідного зображення, яке розпізнається детектором.

```
def detect(sess, detection_graph, image):
  image_np_expanded = np.expand_dims(image, axis=0)
  image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
  boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
  scores = detection_graph.get_tensor_by_name('detection_scores:0')
  classes = detection_graph.get_tensor_by_name('detection_classes:0')
  num_detections = detection_graph.get_tensor_by_name('num_detections:0')

  start_time = time.time()
  (boxes, scores, classes, num_detections) = sess.run(
      [boxes, scores, classes, num_detections],
      feed_dict={image_tensor: image_np_expanded})

  return (boxes, scores, classes, num_detections)
```

Рисунок 3.11 – функція детектування

Для тестування якості розпізнавання відкриваємо потік з відеофайлу та очікувані результати розпізнавання з JSON файлу. Для роботи з відео використовуємо бібліотеку OpenCV, яка дозволяє працювати з зображеннями та відео. Також завантажуємо файл конфігурації класів розпізнавання. Для підрахунку якості розпізнавання ініціалізуємо змінні, які будуть зберігати результати (Рисунок 3.12).

```
cap = cv2.VideoCapture(f'/content/test_files/{iteration}/RGB.mp4')
labels = read_json(f'/content/test_files/{iteration}/RGB_label.json')

detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABEL_MAP)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

results = (0, 0, 0, 0) # (tp, tn, fp, fn)

index = 0;

width  = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

iou_values = []
times = []
```

Рисунок 3.12 – підготовка моделі до тестування

Розпізнавання проводимо в циклі за умовою, що файл ще не закінчився(див. Рисунок 3.13). Кожен отриманий кадр передаємо в функцію розпізнавання. Результати додаємо до раніше створених змінних. Швидкість розпізнавання фіксуємо за допомогою вбудованого в Python модулю time.

```
with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
      while(cap.isOpened()):
        ret, image_np = cap.read()

        start_time = time.time()
        (boxes, scores, classes, num_detections) = detect(sess, detection_graph, image_np)
        execution_time = time.time() - start_time

        times.append(execution_time)

        recognized = 1 if np.any(scores > 0.5) else 0

        results = calculate(results, recognized, labels['exist'][index])

        received_ymin, received_xmin, received_ymax, received_xmax = boxes[0][np.argmax(scores)]

        expected_xmin, expected_ymin, w, h = labels['gt_rect'][index]

        expected_ymax = expected_ymin + h
        expected_xmax = expected_xmin + w

        boxA = (received_xmin * width, received_ymin * height, received_xmax * width, received_ymax * height)
        boxB = (expected_xmin, expected_ymin, expected_xmax, expected_ymax)

        iou_values.append(bb_intersection_over_union(boxA, boxB))
```

Рисунок 3.13 - функція для тестування отриманих результатів

Отримані результати занесемо в таблицю.

Таблиця 3.2 – результати проведених експериментів

Модель	**Precision**	**Recall**	**IoU**	**Швидкість виконання**	**FPS**
SSD	1	0.5824	71%	0.012660	79
Faster R-CNN	1	0.7924	78%	0.076040	13
SSD з аугментацією	1	0.6382	74%	0.012499	80

Faster R-CNN з аугментацією	1	0.8382	79%	0.077476	13

Швидкість розпізнавання та показники в ході есперименті виглядали наступним чином.

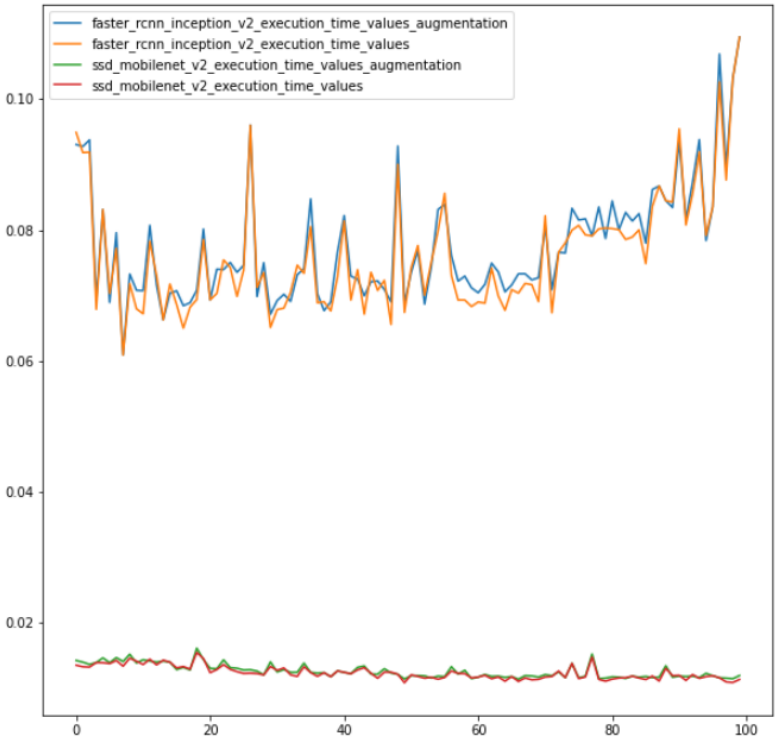


Рисунок 3.14 – швидкість розпізнавання в ході експериментів

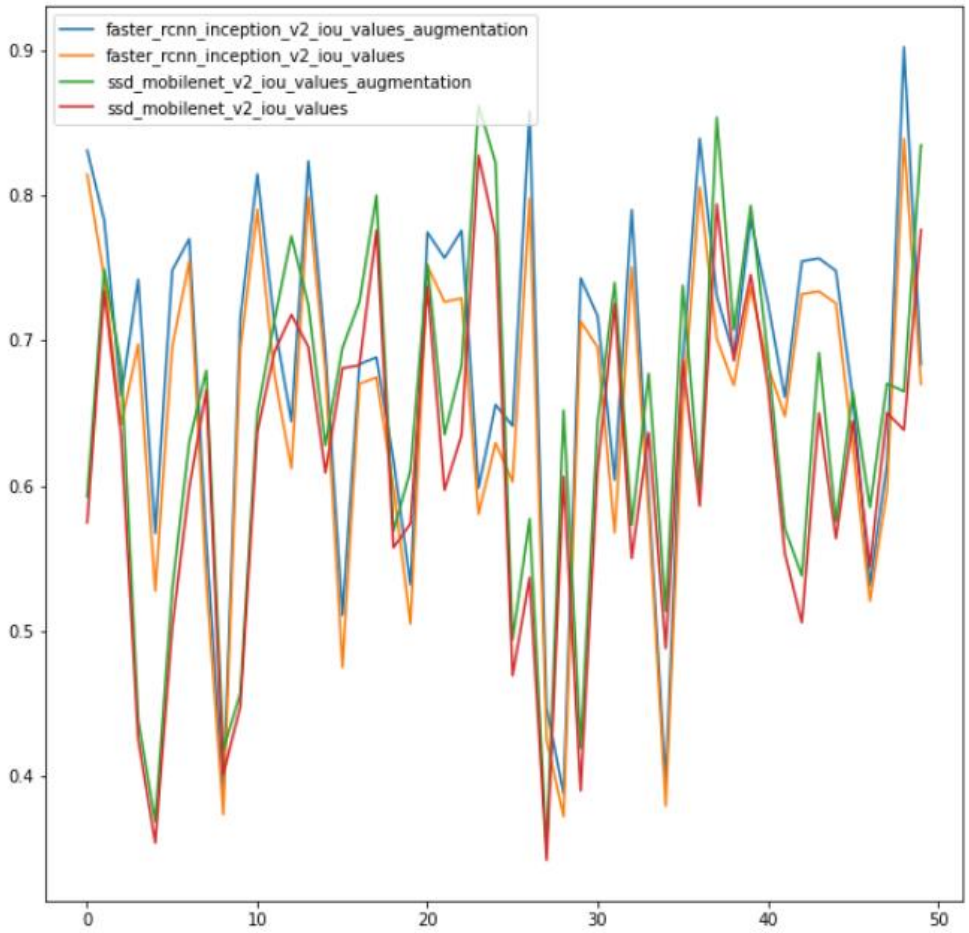


Рисунок 3.15 – показники IoU в ході експериментів

Як видно з Рисунок 3.14 та Рисунок 3.15 швидкість розпізнавання від застосування техніки аументації майже ніяк не змінилася. Відмінність результатів можна списати на особливості виконання конкретного експерименту. Як видно з графіку швидкість роботи моделі SSD відрізняється на 5 разів. За оцінкам IoU моделей показники виконання моделі Faster R-CNN вищі як з використанням техніки аугментації, так і без.

ВИСНОВКИ

Розвиток галузі безпілотних літаючих об’єктів спровокував застосування цих засобів в заборонених цілях, що в свою чергу спровокувало розвиток технологій захисту від подібних об’єктів. Існує доволі багато підходів до захисту, і майже для кожної системи захисту необхідна система детектування, яка могла б знаходити і реагувати на заборонені об’єкти, які з’являються в полі зору даної системи.

В ході виконання науково-дослідницької роботи було розглянуто можливості застосування технологій комп’ютерного зору в цілях детектування та трекінгу безпілотних літаючих об’єктів. Застосування даної технології дозволяє побудувати комерційно вигідні технічні засоби, які будуть мати достатню точність розпізнавання та швидкість роботи.

В ході літературного огляду було розглянуто декілька найбільш сучасних і якісних реалізацій детекторів. В якості практичної перевірки гіпотез було вирішено побудувати модель системи з застосуванням моделей Faster RCNN та SSD. Дані моделі відрізняються між собою в швидкості роботи та точності. В ході роботи було розглянуто чи можливо застосувати кожну з моделей в задачах захисту від безпілотних літаючих об’єктів. Також робота розглядає швидкість та якість обробки відеопотоку для перевірки можливості застосування системи в режимі живого часу.

Для оптимізації моделей була розглянута техніка аугментації, яка дозволяє покращити якість детектування як для екстремальних випадків детектування, так і для детектування в умовах звичайного відеопотоку.

З проведених експериментів ми визначили, що найкращим способом детектування буде застосувати модель SSD з використанням техніки аугментації, так як результати якості роботи задовільні, а швидкість виконання значно вища.

СПИСОК ЛІТЕРАТУРИ

1. H. Tian, T. Wang, Y. Liu, X. Qiao, and Y. Li, “Computer vision technology in agricultural automation —A review,” *Information Processing in Agriculture*. 2020, doi: 10.1016/j.inpa.2019.09.006.
2. “Gatwick Airport drone attack: Police have ‘no lines of inquiry’ - BBC News.”. URL: https://www.bbc.com/news/uk-england-sussex-49846450. (Дата звернення 15.12.2020)..
3. “Gang who flew drones carrying drugs into prisons jailed - BBC News.” URL: https://www.bbc.com/news/uk-england-45980560. (Дата звернення 17.12.2020).
4. “Drone law in the UK | Nottinghamshire Police.” URL: https://www.nottinghamshire.police.uk/advice/drone-law-uk. (Дата звернення 17.12.2020).
5. “Flying your drone safely and legally.” URL: https://tc.canada.ca/en/aviation/drone-safety/flying-your-drone-safely-legally. (Дата звернення 17.12.2020).
6. P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, vol. 1, doi: 10.1109/cvpr.2001.990517.
7. N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. I, pp. 886–893, doi: 10.1109/CVPR.2005.177.
8. P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008, doi: 10.1109/CVPR.2008.4587597.
9. J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M.

Smeulders, “Selective search for object recognition,” *Int. J. Comput. Vis.*, 2013, doi: 10.1007/s11263-013-0620-5.

10. Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.
11. W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.
12. C. L. Zitnick and P. Dollár, “Edge boxes: Locating object proposals from edges,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, doi: 10.1007/978-3-319-10602-1_26.
13. C.-J. L. Chih-Wei Hsu, Chih-Chung Chang, “A Practical Guide to Support Vector Classification,” *BJU Int.*, 2008.
14. S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv*. 2018.
15. T. W. Hui, X. Tang, and C. C. Loy, “A lightweight optical flow CNN – Revisiting data fidelity and regularization,” *arXiv*. 2019, doi: 10.1109/tpami.2020.2976928.
16. X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Journal of Machine Learning Research*, 2011.
17. R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, doi: 10.1109/CVPR.2014.81.
18. “GitHub: The top 10 programming languages for machine learning - TechRepublic.”. URL: https://www.techrepublic.com/article/github-the-top-10-programming-languages-for-machine-learning/.(Дата звернення 09.04.2021).
19. “models/tf1_detection_zoo.md at master · tensorflow/models.” . URL:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md (Дата звернення 15.04.2021).

20. “TFRecord and tf.train.Example | TensorFlow Core.” . URL: https://www.tensorflow.org/tutorials/load_data/tfrecord. (Дата звернення 17.04.2021).

ДОДАТОК А

Код навчання моделей:

```
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -qq Cython contextlib2 pillow lxml matplotlib

!pip install -qq pycocotools

%tensorflow_version 1.x

import tensorflow as tf
print(tf.__version__)

from google.colab import drive
drive.mount('/content/drive', force_remount=True)


%cd /content/

!rm -rf xml.zip
!rm -rf images.zip

!cp -a drive/MyDrive/dimploma/xml.zip .
!cp -a drive/MyDrive/dimploma/images.zip .
!cp -a drive/MyDrive/dimploma/configs .
!cp -a drive/MyDrive/dimploma/config_templater.py .

!unzip -oq images.zip
!unzip -oq xml.zip

from __future__ import division, print_function, absolute_import

import pandas as pd
import numpy as np
import csv
import re
import cv2
import os
import glob
import xml.etree.ElementTree as ET

import io
import tensorflow.compat.v1 as tf

from PIL import Image
from collections import namedtuple, OrderedDict

import shutil
```

```
import urllib.request
import tarfile

from google.colab import files

from config_templater import prepare_template

ROOT_PATH = '/content'

!rm -rf data
!mkdir data

!mkdir data/images data/train_labels data/test_labels


!cp images/* data/images

!cp xml/* data/train_labels

!ls data/train_labels/* | sort -R | head -400 | xargs -
I{} mv {} data/test_labels

!ls -1 data/train_labels/ | wc -l

!ls -1 data/test_labels/ | wc -l



%cd /content/data
images_extension = 'jpg'

def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    file_name = xml_file.split('/')[1]
    name = file_name.split('.')[0]
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (name+'.jpg',
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
               int(member[4][1].text),
               int(member[4][2].text),
```

```
                int(member[4][3].text))
        xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', '
xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    classes_names = list(set(classes_names))
    classes_names.sort()
    return xml_df, classes_names

for label_path in ['train_labels', 'test_labels']:
    image_path = os.path.join(os.getcwd(), label_path)
    xml_df, classes = xml_to_csv(label_path)
    xml_df.to_csv(os.path.join(ROOT_PATH, f'{label_path}.csv'), index=None)
    print(f'Successfully converted {label_path} xml to csv.')

label_map_path = os.path.join("label_map.pbtxt")

pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n    display_name: 'Drone
'\n }}\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)

%cd /content/
!rm -rf models
!git clone --q https://github.com/tensorflow/models.git --branch r1.13.0

os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/re
search/slim/'

!python3 object_detection/builders/model_builder_test.py



from object_detection.utils import dataset_util
%cd /content/models

DATA_BASE_PATH = '/content/'
image_dir = '/content/data/images/'

def class_text_to_int(row_label):
    if row_label == 'drone':
        return 1
    else:
```

```
        None


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.grou
ps.keys(), gb.groups)]

def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename))
, 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes
_text),
        'image/object/class/label': dataset_util.int64_list_feature(classe
s),
    }))
```

```
    return tf_example

for csv in ['train_labels', 'test_labels']:
  writer = tf.io.TFRecordWriter(DATA_BASE_PATH + csv + '.record')
  path = os.path.join(image_dir)
  examples = pd.read_csv(DATA_BASE_PATH + csv + '.csv')
  grouped = split(examples, 'filename')
  for group in grouped:
      tf_example = create_tf_example(group, path)
      writer.write(tf_example.SerializeToString())

  writer.close()
  output_path = os.path.join(os.getcwd(), DATA_BASE_PATH + csv + '.record'
)
  print('Successfully created the TFRecords: {}'.format(DATA_BASE_PATH +cs
v + '.record'))

prepare_template(f'/content/drive/MyDrive/dimploma/configs/{pipeline_file}
', '/content/training_config.config', {
    "STEP_NUMBER": "10000",
    "TRAIN_INPUT_PATH": "/content/train_labels.record",
    "TRAIN_LABEL_PATH": '/content/data/label_map.pbtxt',
    "TEST_INPUT_PATH": '/content/data/test_labels.record',
    "TEST_LABEL_PATH": '/content/data/label_map.pbtxt',
    "MODEL_PATH": '/content/models/research/pretrained_model/model.ckpt'
})
model_pipeline = '/content/training_config.config'

!python3 /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={model_pipeline}\
    --model_dir={model_dir} \
    --alsologtostderr \

lst = os.listdir(model_dir)
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')
last_model_path = os.path.join(model_dir, last_model)
print(last_model_path)

#exports the model specifed and inference graph
!python /content/models/research/object_detection/export_inference_graph.p
y \
    --input_type=image_tensor \
    --pipeline_config_path={model_pipeline} \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path}
```

ДОДАТОК Б

```
from google.colab import drive
drive.mount('/content/drive')


model_name = 'ssd_mobilenet_v2'


!cp -a drive/'MyDrive'/dimploma/models/{model_name}/label_map.pbtxt .
!cp -a drive/'MyDrive'/dimploma/models/{model_name}.pb .

pip install tensorflow-object-detection-api
%tensorflow_version 1.x
!cp -a drive/MyDrive/dimploma/test_files/ .

import json

def read_json(path):
  with open(path) as json_file:
    return json.load(json_file)

def calculate_intersection_over_union(boxA, boxB):
  xA = max(boxA[0], boxB[0])
  yA = max(boxA[1], boxB[1])
  xB = min(boxA[2], boxB[2])
  yB = min(boxA[3], boxB[3])
  interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

  boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
  boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

  iou = interArea / float(boxAArea + boxBArea - interArea)

  return iou

def calculate(option_values, received_value, expected_value):
  (tp, tn, fp, fn) = option_values
  if received_value == 1 and expected_value == 1:
    tp += 1
  if received_value == 0 and expected_value == 0:
    tn += 1
  if received_value == 1 and expected_value == 0:
    fp += 1
  if received_value == 0 and expected_value == 1:
    fn += 1

  return (tp, tn, fp, fn)

def detect(sess, detection_graph, image):
```

```
  print(image.shape)
  image_np_expanded = np.expand_dims(image, axis=0)
  print(image_np_expanded.shape)
  image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
  boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
  scores = detection_graph.get_tensor_by_name('detection_scores:0')
  classes = detection_graph.get_tensor_by_name('detection_classes:0')
  num_detections = detection_graph.get_tensor_by_name('num_detections:0')

  start_time = time.time()
  (boxes, scores, classes, num_detections) = sess.run(
      [boxes, scores, classes, num_detections],
      feed_dict={image_tensor: image_np_expanded})

  return (boxes, scores, classes, num_detections)

import numpy as np
import os
import tensorflow as tf
import cv2
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
from google.colab.patches import cv2_imshow
import time

PATH_TO_FROZEN_GRAPH =  '/content/model.pb'

PATH_TO_LABEL_MAP = '/content/label_map.pbtxt'

PATH_TO_TEST_IMAGES = '/content/test_images/'
NUM_CLASSES = 1



def recognize(iteration, max_iter = None):
  cap = cv2.VideoCapture(f'/content/test_files/{iteration}/RGB.mp4')
  labels = read_json(f'/content/test_files/{iteration}/RGB_label.json')

  detection_graph = tf.Graph()
  with detection_graph.as_default():
      od_graph_def = tf.GraphDef()
      with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
          serialized_graph = fid.read()
          od_graph_def.ParseFromString(serialized_graph)
          tf.import_graph_def(od_graph_def, name='')

  label_map = label_map_util.load_labelmap(PATH_TO_LABEL_MAP)
  categories = label_map_util.convert_label_map_to_categories(label_map, m
ax_num_classes=NUM_CLASSES, use_display_name=True)
  category_index = label_map_util.create_category_index(categories)
```

```
  results = (0, 0, 0, 0) # (tp, tn, fp, fn)

  index = 0;

  width  = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
  height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

  iou_values = []
  times = []


  with detection_graph.as_default():
      with tf.Session(graph=detection_graph) as sess:
        while(cap.isOpened()):
          ret, image_np = cap.read()

          start_time = time.time()
          (boxes, scores, classes, num_detections) = detect(sess, detectio
n_graph, image_np)
          execution_time = time.time() - start_time

          times.append(execution_time)

          recognized = 1 if np.any(scores > 0.5) else 0

          results = calculate(results, recognized, labels['exist'][index])

          received_ymin, received_xmin, received_ymax, received_xmax = box
es[0][np.argmax(scores)]

          expected_xmin, expected_ymin, w, h = labels['gt_rect'][index]

          expected_ymax = expected_ymin + h
          expected_xmax = expected_xmin + w

          boxA = (received_xmin * width, received_ymin * height, received_
xmax * width, received_ymax * height)
          boxB = (expected_xmin, expected_ymin, expected_xmax, expected_ym
ax)

          iou_values.append(bb_intersection_over_union(boxA, boxB))

          index += 1;

          vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
```

```
            category_index,
            use_normalized_coordinates=True,
            line_thickness=8,
            )

          cv2_imshow(cv2.resize(image_np, (500, 600)))

          if(max_iter and max_iter == index):
            break

  return {
      'execution_time': times,
      'count': index,
      'iou_values': iou_values,
      'results': results
    }
```