НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

«До захисту допущено»
Завідувач кафедри
_____ Олег Чертов
«___» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Наука про дані та математичне моделювання»
спеціальності 113 «Прикладна математика»
на тему: «Математичне та програмне забезпечення системи виявлення військової техніки на фото»

Виконав:
студент IV курсу, групи КМ-93
Данилевич Олександр Олександрович _____

Керівник:
Старший викладач, канд. техн. наук, доцент,
Ліскін Вячеслав Олегович _____

Консультант з розділу Моделі системи виявлення техніки на фото:
Доцент, канд. техн. наук, ст. наук. співроб.,
Маслянко Павло Павлович _____

Консультант з нормоконтролю:
Старший викладач,
Мальчиков Володимир Вікторович _____

Рецензент:
Доцент, канд. техн. наук, доцент,
Тарасенко-Клятченко Оксана Володимирівна _____

Засвідчую, що в цій дипломній роботі
немає запозичень із праць інших авторів
без відповідних посилань.
Студент Данилевич О. О.

Київ — 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра прикладної математики

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 113 «Прикладна математика»

Освітньо-професійна програма «Наука про дані та математичне моделювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олег Чертов
«___» _____ 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Данилевичу Олександру Олександровичу

1. Тема роботи: «Математичне та програмне забезпечення системи виявлення військової техніки на фото», керівник роботи Ліскін Вячеслав Олегович, старший викладач, канд. техн. наук, доцент, затверджені наказом по університету від «31» травня 2023 р. № 2108-С.
2. Термін подання студентом роботи: «12» червня 2021 р.
3. Вихідні дані до роботи: початкові зображення, точність розробленої системи повинна бути вище 60% за представленими критеріями.
4. Зміст роботи: дослідити існуючі рішення для виявлення об’єктів на фото, проаналізувати та адаптувати обрані методи для вирішення проблеми, розробити конкурентоспроможну систему виявлення техніки на фото, протестувати розроблену систему.
5. Перелік ілюстративного матеріалу: візуалізація роботи методів розпізнавання зображень, архітектури нейронних мереж, модель системи у вигляді діаграми компонентів, результати процесу навчання моделі, візуалізації роботи програми.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 3. Модель системи виявлення техніки на фото	Маслянко П. П., доцент		

7. Дата видачі завдання: «06» лютого 2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за тематикою та збір даних	12.11.2022	
2	Проведення порівняльного аналізу математичних методів розпізнавання зображень	14.12.2022	
3	Проведення порівняльного аналізу математичних методів виявлення об’єктів на фото	24.12.2022	
4	Підготовка матеріалів першого розділу роботи	01.02.2023	
5	Розроблення математичного забезпечення для системи виявлення техніки на фото	01.03.2023	
6	Підготовка матеріалів другого розділу роботи	15.03.2023	

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
7	Підготовка матеріалів третього розділу роботи	05.04.2023	
8	Розроблення програмного забезпечення для системи виявлення техніки на фото	15.04.2023	
9	Підготовка матеріалів четвертого розділу роботи	03.05.2023	
10	Оформлення пояснювальної записки	01.06.2023	

Студент _____ Данилевич О. О.

Керівник роботи _____ Ліскін В. О.

АНОТАЦІЯ

Дипломну роботу виконано на 54 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 22 найменувань. У роботі наведено 11 рисунків та 3 таблиці.

Метою даної дипломної роботи є реалізація конкурентоспроможної системи для вирішення проблеми виявлення військової техніки на фото.

У роботі зроблено аналіз існуючих рішень даної задачі — розглянуто ненейронні підходи та підходи з використанням штучних нейронних мережі, розроблено систему виявлення військової техніки на фото, виявлено переваги і недоліки.

Розроблено систему виявлення об’єктів (техніки) на фото, проведено тестування системи на вибірці.

Ключові слова: фотографія, згорткова нейронна мережа, YOLO v1, система виявлення об’єктів, функції втрат.

ABSTRACT

The thesis is completed on 54 sheets, it contains 2 appendices and a list of references to used sources from 22 names. The work contains 11 figures and 3 tables.

The goal of this thesis is to implement a competitive system for solving the problem of detecting objects (vechicles) in a photo.

The paper analyzes the existing solutions to this problem non-neural approaches and approaches using artificial neural networks are considered, a system for detecting military equipment in a photo has been developed, advantages and disadvantages are identified.

A system for detecting objects (techniques) in the photo was developed, and the system was tested on a sample of images.

Keywords: photography, convolutional neural network, YOLO algorithm, object detection system, loss functions.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN – Convolutional Neural Network, Згорткова нейронна мережа,

HOG - Histogram of Oriented Gradients, Гістограма орієнтованих градієнтів,

IoU – Intersection over Union, Перетин по об’єднанню

R-CNN – Region CNN, Згорткова нейронна мережа з вибором регіонів,

SIFT - Scale-invariant Feature Transform, Масштабно-інваріантна трансформація ознак,

YOLO – You Only Look Once, Ти дивишся тільки один раз,

ШН – Штучний нейрон,

ШНМ – Штучні нейронні мережі.

ВСТУП

Людині достатньо на долю секунди побачити зображення і в голові вже будуть знання про об’єкти, що зображені на ньому. Ці механізми настільки швидкі та повсякденні, що ми навіть не задумуємось про їх існування. З розвитком нашої цивілізації ми прийшли до технологій збереження та передавання інформації за допомогою комп’ютерів і вже зараз об’єм інформації настільки великий що людям довелося б витратити життя для того щоб самостійно обробити всю інформацію. Саме тому зараз досить великого розвитку набувають технології які покликані автоматизувати процес обробки інформації, в тому числі і з графічною.

Однією з проблем обробки графічної інформації є задача виявлення об’єктів на зображенні. Наприклад на виробництві раніше люди особисто перевіряли присутність виробу на конвеєрі, а система розпізнавання сьогодні дає можливість робити це автоматично. Також чудовим прикладом є системи автопілоту в автомобілях які за допомогою візуальних зображень можуть розпізнавати об’єкти навколо автомобіля і підказувати людині за кермом або взагалі самостійно керувати автомобілем.

З початком повномасштабного вторгнення росії в Україну ми побачили велике проникнення технологій спостереження і розвідки на полі бою і всі вони працюють з графічними даними. Дані надходили з різних джерел – вуличних відеокамер, дронів, телефонів громадян, і така велика кількість інформації потребує обробки. Цією обробкою зазвичай займаються люди проте це є не дуже ефективним методом вирішення проблеми та має свої недоліки, як от потреба у постійному спостереженні.

Тому для вирішення даної проблеми можна використовувати сучасні технології, що будуть прискорено та автоматизовано виконувати цей процес. Система виявлення об’єкту на зображенні за допомогою нейронних мереж оптимізує витрату часу та сил.

1 ПОСТАНОВКА ЗАДАЧІ

Метою написання роботи є розробка системи для виявлення військової техніки на фото на основі машинного навчання.

При розробленні відповідного забезпечення потрібно розв’язати наступні завдання:

а) проведення порівняльного аналізу алгоритмів системи виявлення об'єктів військової техніки на фото;

б) обрати та адаптувати існуючий метод для вирішення задачі виявлення об’єктів;

в) створення математичного забезпечення;

г) розробка програмного забезпечення на базі вибраного математичного методу;

д) тестування розробленої автоматизованої системи.

Реалізована система має задовольняти такі вимоги:

а) мати високі показники ефективності розпізнавання;

а) виявляти об'єкт військової техніки на фото швидше ніж за 5 секунд;

б) надавати аналітичну інформацію про результати роботи програми.

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВВИЯВЛЕННЯ ОБ’ЄКТІВ

2.1. Огляд існуючих програмних рішень

У відкритому доступі немає повноцінних систем виявлення військової техніки на зображеннях оскільки зазвичай вони мають невеликий бізнес потенціал, а приватні розробки зазвичай знаходяться у руках військових що ними користуються. Більшість таких систем зав’язані на вирішенні більш побутових проблем з іншими видами техніки, наприклад системи виявлення кораблів на морі для уникнення зіткнень, або системи навчені на розпізнавання інших об’єктів – літаків, автомобілів, людей для інших бізнес-задач.

2.1.1. Онлайн застосунок Google Lens

Суть застосунку Google Lens [1] в тому що він на основі зображення шукає схожі зображення в інтернеті. Обробка зображень і виведення схожих результатів відбувається досить швидко, але результат сильно залежить від того яку частину зображення ви виділите та й основне завдання вирішується частково, оскільки, хоч застосунок і видає схожі зображення і певною мірою підказує чи є об'єкт на фото – він ніяким чином не локалізує місцезнаходження об'єкту на самому фото.

2.1.2. Онлайн-засіб Aspose

Сервіс Aspose [2]пропонує виявити авто на зображенні. Загалом цей сервіс працює непогано але коли ми переходимо до більш специфічних видів техніки окрім автомобілів то цей сервіс майже не розпізнає об’єктів на фотографіях.

2.2. Огляд існуючих методів виявлення об'єктів

Загалом усі методи розподіляють на дві основні категорії: ненейронні алгоритми та алгоритми що використовують нейронні мережі. Ненейронні методи спочатку виділяють ознаки за допомогою одного, з алгоритмів а потім використовують метод опорних векторів для проведення подальшої класифікації. Алгоритми що використовують нейронні мережі зазвичай базуються на згорткових нейронних мережах.

2.2.1. Метод Віоли-Джонса

Метод виявлення об’єктів Віоли–Джонса [3] — це метод машинного навчання для виявлення об’єктів на зображеннях. Він був запропонований в 2001 році Полом Віолою та Майклом Джонсом. Загалом метод спеціалізується на виявленні облич людей на зображеннях проте може бути пристосований і для вирішення інших задач.

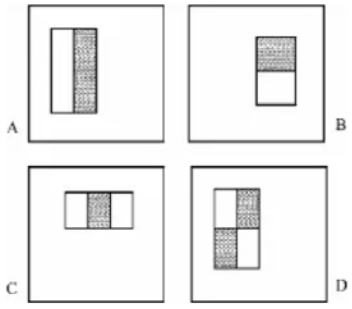


Рис. 2.1 – Приклади ознак Хаара [3]

Суть методу полягає у сумуванні інтенсивності пікселів у певних областях, ці області називають признаками Хаара. Вони певним чином позиціонуються на зображенні, а потім беруть різницю суми інтенсивності пікселів – ця різниця називається ознакою Хаара. Потім будується класифікатор на основі ознак Хаара який надалі і відповідає на питання чи є на зображенні об’єкт який ми шукаємо.

2.2.2. Метод масштабно-інваріантної трансформації ознак

Спочатку в Scale-invariant Feature Transform (SIFT) [4] визначаються ключові точки з набору навчальних зображень і записуються в базу даних. Об'єкт розпізнається на новому зображенні шляхом порівняння ознак (визначних точок) з нового зображення з ознаками що знаходяться в базі даних, самі ж варіанти класифікації визначаються на основі евклідової довжини між векторами. З усіх ознак у новому зображенні вибирається підмножина ключових точок, які найбільше погоджуються з об’єктом за його розташуванням, масштабом і орієнтацією. Визначення відповідних ознак виконується за допомогою хеш-таблиці узагальненого перетворення Хафа.

Рис 2.2 – Візуалізація ознак у методі SIFT [4]

Кожен блок із 3 або більше ознак, що погоджуються з об’єктом і його положенням, підлягає подальшій детальній перевірці відповідності моделі, а блоки що різко відрізняють відкидаються. В кінці, обчислюється ймовірність, що визначений набір ознак говорить про присутність об'єкта.

2.2.3. Гістограма орієнтованих градієнтів

Гістограма орієнтованих градієнтів [5] — це структура особливих точок, що використовуються в обробці зображень для виявлення об’єкті на них. Алгоритм методу базується на підрахунку кількості направлень градієнту в локалізованих частинах зображення. Цей метод подібний до SIFT, але відрізняється тим, що він

обчислюється на структурі що являє собою сітку з рівномірно розташованих комірок і використовує нормалізацію локального контрасту, що перекривається, для покращення точності.

2.2.4. Алгоритми що використовують нейронні мережі

Група математичних моделей, відомих як штучні нейронні мережі, імітує функціонування біологічних нейронів у живих істотах. Штучні нейронні моделі використовуються в широкому спектрі задач машинного навчання, включаючи класифікацію, прогнозування, обробку даних тощо.

Мережа штучних нейронів, з’єднаних певною взаємодією, утворює ШНМ. Реальні числові значення зазвичай є сигналом, який взаємодіє між ШН. Вагові коефіцієнти на кожній лінії зв’язку контролюють, наскільки потужним є сигнал між ШН. Кожен нейрон у структурі взаємодії отримує певний сигнал від попереднього нейрона або зовнішньої взаємодії та обчислює результат, який передається наступному нейрону в структурі. Оптимізація вихідного сигналу нейрона відбувається шляхом корекції вагових коефіцієнтів ваг вихідних сигналів та функції активації.

Структурно нейронна мережа зазвичай представлена у вигляді шарів які зв’язані між собою.

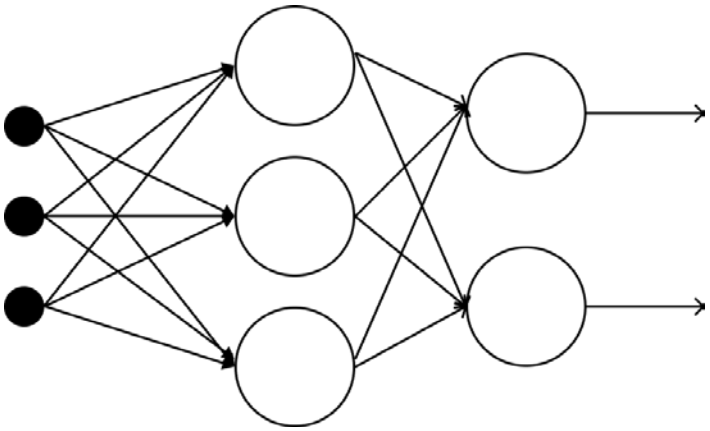


Рис. 2.3 – Структура нейронної мережі з вхідними шарами, 1-м прихованим шаром та вихідним шаром

Навчання нейронної мережі – це процес, під час якого у зв’язках між ШН встановлюються оптимальні значення вагових коефіцієнтів для розв’язання штучною нейронною мережею конкретних задач. Існує дві основних підходи для навчання ШНМ: навчання з учителем та навчання без учителя.

Навчання з учителем – методика, де використовується певний набір даних, в якому відмічені правильні відповіді або цільові значення. В процесі навчання, ШНМ на базі початкових даних генерує результат, який потім порівнюється з правильними даними. Для покращення результатів роботи ШНМ, обчислюється похибка між вихідним результатом та правильними даними та коригуються ваги в ШНМ. Цей процес відбувається поки похибка не буде задовільною.

Навчання без учителя – це методика, де набір даних не має маркувань правильних відповідей. ШНМ намагається самостійно виконати потрібну задачу, без використання знань ззовні (вчителя). Такий підхід використовується для виявлення мережею певних взаємозалежностей, структур або закономірності в даних. Для навчання ШНМ без вчителя треба визначити параметри мережі, функції втрат та оцінку мережі за допомогою різних метрик.

2.2.4.1. Згорткові нейронні мережі та R-CNN

Згорткові нейронні мережі або CNN – це клас штучних нейронних мереж, які загалом використовуються для роботи з зображеннями або відео. Вони відомі як одні з біологічних моделей, що використовувалися для задач розпізнавання образів, таких як розпізнавання облич і розпізнавання рукописних цифр. [6]

Ідея згорткових нейронних мереж полягає у використанні набору фільтрів, які аналізують початкове зображення та виявляють з нього корисну інформацію, формуючи карту ознак. Після згортки йде операція об’єднання (pooling), яка зменшує розмір карти та зберігає найбільш корисну інформацію. Після цього процесу застосовуються повнозв’язні шари, що, в свою чергу, здійснюють різні завдання залежно від постановки.

Загалом CNN використовуються у задачах виявлення об’єктів але дуже часто виникає проблема з обчисленнями. Основна причина, чому ми не можемо вирішити цю проблему шляхом побудови стандартної згорткової мережі з подальшим повнозв’язним шаром, полягає в тому, що довжина вихідного шару є змінною, а не постійною, тому що кількість входжень об’єктів, що ми шукаємо непостійна. Найпростішим підходом до вирішення цієї проблеми було б взяти різні підмножини з зображення та використовувати CNN для класифікації присутності об’єкта в цій області. Проблема цього підходу полягає в тому, що цікаві об’єкти можуть мати різні просторові розташування на зображенні та різні співвідношення сторін тому такий підхід є дуже затратним і з точки зору обчислень і з точки зору часу.

Щоб обійти проблему вибору величезної кількості регіонів, Росс Гіршик запропонував метод, за допомогою якого ми використовуємо вибірковий пошук, щоб виділити лише близько 2000 регіонів із зображення – Region CNN.

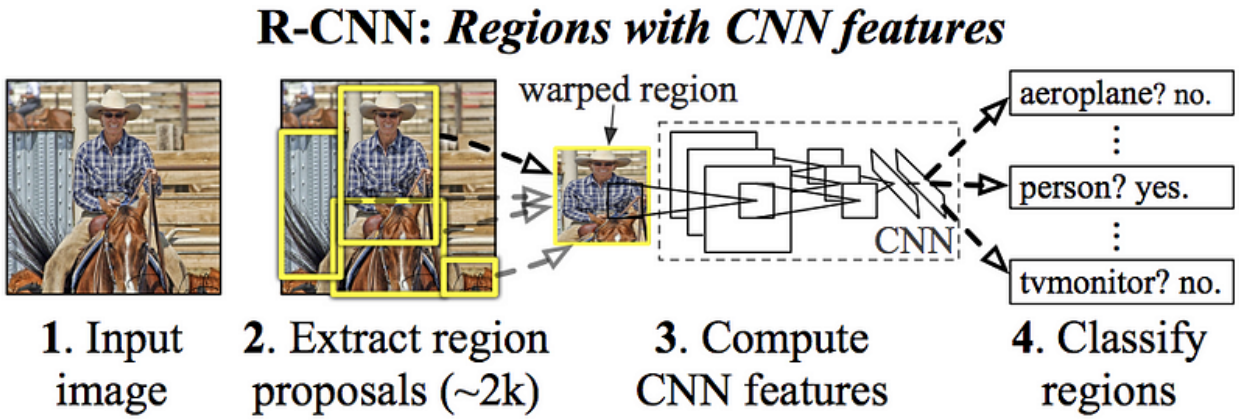


Рис. 2.4 – Архітектура R-CNN згорткової нейронної мережі з повнозв'язними шарами [8]

Детектор R-CNN [8] спочатку генерує пропозиції регіонів за допомогою такого алгоритму, як Edge Boxes [7]. Області пропозиції обрізаються із зображення та змінюються в розмірі. Потім CNN класифікує обрізані та змінені регіони. Нарешті, обмежувальні рамки пропозиції регіону уточнюються машиною опорних векторів (SVM), яка навчена за допомогою функцій CNN.

2.2.4.2. Модель You Look Only Once (YOLO)

Порівняно з іншими мережами класифікації пропозицій регіону (R-CNN та її модифікації), які виконують виявлення на різних пропозиціях регіону і, таким чином, виконують багаторазове прогнозування для різних регіонів зображення, архітектура YOLO більше схожа на FCNN (повністю згорткова нейронна мережа) і передає зображення $(n \times n)$ один раз через усю мережу, а виводом є передбачення $(m \times m)$. Ця архітектура розділяє вхідне зображення в сітці $m \times m$ і для кожного покоління сітки 2 обмежувальні рамки та ймовірності класу для цих обмежувальних рамок.

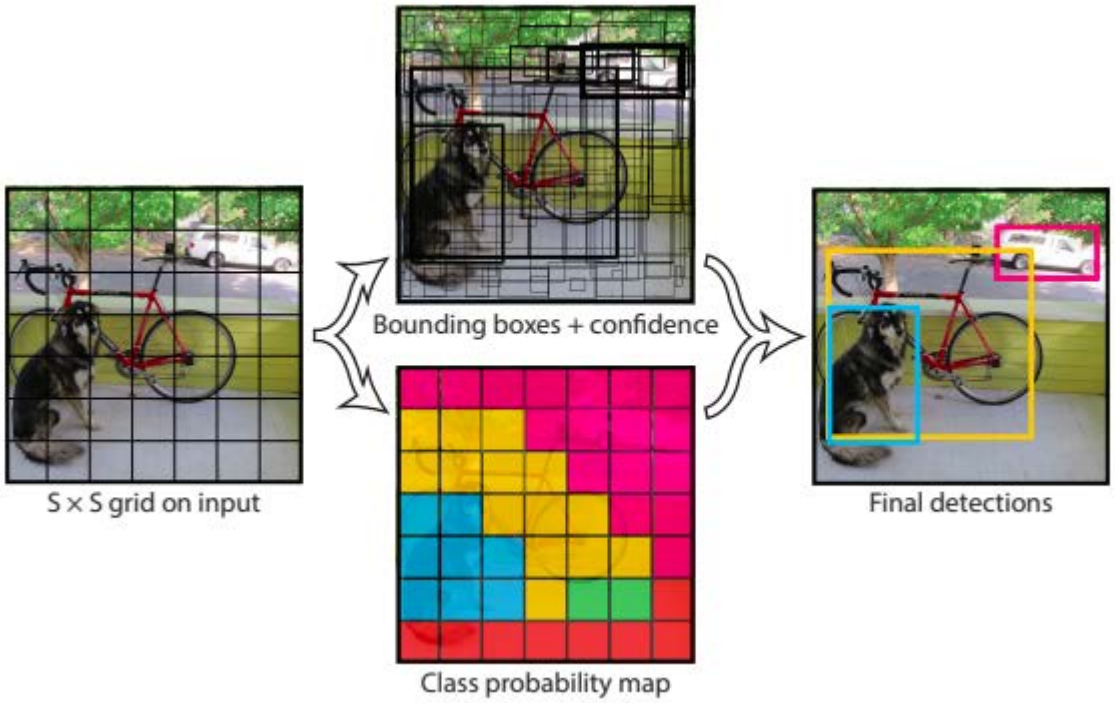


Рис. 2.5 – Візуалізація алгоритму роботи YOLO [9]

Система YOLO розбиває вхідне зображення на сітку SxS. Якщо центр об'єкта потрапляє в комірку сітки, ця комірка використовується для його виявлення. Крім того, YOLO прогнозує обмежувальні прямокутники B і оцінки достовірності в клітинці сітки. Показник достовірності показує, наскільки ймовірно, що рамка містить об’єкт. Таким чином, кожна обмежувальна рамка має 5 прогнозів: x, y, w, h і показник достовірності, де (x, y) є відносно клітинки сітки, а (w, h) – відносно всього зображення.

2.3. Висновки до розділу

Після аналізу наведених матеріалів, зробимо порівняння існуючих можливостей систем з виявлення об’єктів на фото:

Таблиця 2.1 Порівняння існуючих рішень виявлення техніки на фото

	Google Lens	Aspose
Ціна	Безкоштовно	Безкоштовно
Час	до 2 секунд	до 30 секунд
Якість результатів	Середня	Низька
Кількість об’єктів	Велика	Мала

Тут наведені два різних рішення, що відрізняються один від одного.

Після визначення основних можливих методів для створення оброблених зображень, зробимо їх порівняння :

Таблиця 2.2 Порівняння методів виявлення об’єктів на фото

	Метод HOG	Метод SIFT	Метод Віоли-Джонса	R-CNN	YOLO
Використання у виявленні об’єктів	Широке	Можливе	Можливе	Широке	Широке
Здатність до генералізації	Низька	Низька	Низька	Середня	Висока
Потреба у ресурсах	Середня	Середня	Низька	Висока	Середня
Можливість модифікації	Так	Так	Так	Так	Так
Швидкість роботи	Низька	Середня	Низька	Середня	Висока

Після аналізу та порівняння було обрано YOLO як метод виявлення об’єктів на фото. YOLO має перевагу у швидкості обробки зображень, кращих можливостях до генералізації та не потребує великої кількості обчислювальних потужностей.

3 МОДЕЛЬ СИСТЕМИ ВИЯВЛЕННЯ ТЕХНІКИ НА ФОТО

3.1. Компонентна модель системи виявлення військової техніки на фото

Структуру системи можна представимо у вигляді діаграми компонентів (рисунок 3.1). Компоненти діаграми формуються за функціональною ознакою. Відношення між компонентами відображається через реалізацію відповідних інтерфейсів за допомогою яких відбувається взаємодія.

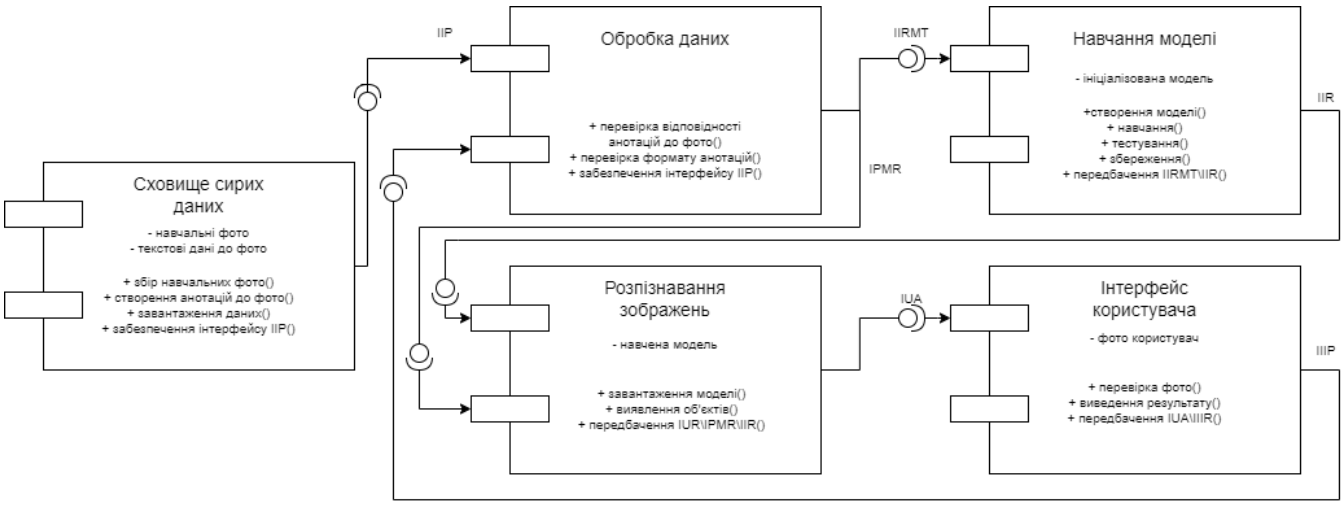


Рис. 3.1 – Модель системи виявлення військової техніки на фото. Діаграма компонентів у нотації UML

Компонент «Сховище сирих даних» – це сховище, у якому зберігаються набір даних, потрібних для навчання та тестування моделі детекції військової техніка на зображенні.

Набір даних складається з файлів двох основних видів: зображень техніки, які представлені файлами з розширенням .jpg та файлів анотацій з інформацією про місцезнаходження об’єкту на фото, які представлені файлами з розширенням .xml. Файли що містять інформацію про знаходження об’єкту на фото повинні називатись так само як і файл зображення якому він відповідає. У базі даних усі фото мають різний розмір та різне забарвлення. Формат даних у текстових файлах виглядає

наступним чином – ми маємо інформацію про зображення, його назву, шлях, розмір, і кількість об’єктів, а також їх місцезнаходження. Його місце визначається значеннями розташування пікселів: X мінімального, X максимального, Y мінімального, Y максимального, що задають межі прямокутника який обмежує об’єкт на зображенні. Вся ця інформація міститься у спеціальних тегах які мають чітко визначений порядок.

База даних була зібрана шляхом об’єднання декількох баз даних з веб сайту roboflow [11] та додаванням власних зображень що були зібрані з відкритих джерел та анотовані на веб-сервісі Img Lab [12]. Всього було використано 9 баз даних [13-22], які були відфільтровані на повторення та помилки і об’єднані. Усього отримано 8966 фото та таку ж кількість файлів анотацій з інформацією про об’єкти.

Компонент «Обробка даних» призначений для приведення формату даних про об’єкти до формату що використовує алгоритм YOLO, перетворення зображень у відповідний розмір моделі який становить 448*448 пікселів та розподілення бази даних на тренувальну, валідаційну та тестову вибірки.

Дані що подаються на вхід до моделі повинні мати вигляд: X , Y, Width, Height – де X та Y це координати нижнього правого кута прямокутника, а інші 2 параметри його ширина та висота відповідно. Для цього застосовуються прості геометричні обчислення після яких ми отримаємо файл у форматі .txt в якому через пробіл записані 4 числа.

Розподіл даних був виконаний наступним чином: навчальна вибірка: - 4481 зображення, валідаційна вибірка – 3806, тестова вибірка – 679.

Усі перетворення були виконані за допомогою бібліотек мови програмування Python.

Компонент «Навчання моделі» призначений для проведення ініціалізації моделі, завантаження усіх даних, проведення навчання, валідації та тестування отриманої моделі.

Побудова моделі розпочинається з додавання шарів до моделі за допомогою методів бібліотеки Keras. Також створюється функція підготовки малих партій

даних (токенізації) з усієї бази даних для навчання моделі. Після цього, ініціалізується функція обчислення помилки моделі за результатом якої, методом зворотнього розповсюдження помилки, оновлюються значення ваг у кожному шарі моделі, а також функція обчислення Intersection over Union (IoU) що використовується для оцінки ефективності виявлення об’єктів шляхом порівняння прогнозованої обмежувальної рамки з реальною обмежувальною рамкою. В кінці ініціалізується модель з задаванням оптимізатора та гіперпараметрів для навчання.

Для визначення результатів обчислюються матриця невідповідностей з певним заданим рівнем впевненості. Після чого визначається accuracy, precision та IoU для заданого рівня впевненості.

Компонент «Розпізнавання зображень» відображає результати роботи моделі на тестових даних або на даних що надав користувач. Компонент завантажує останню версію моделі, отримує шлях до зображень які потрібно обробити, перетворює зображення у числове представлення та передає його у модель після чого отримує відповідь від моделі з прогнозом чи є на зображенні об’єкт військової техніки.

Інтерфейси:

– інтерфейс IIP (Interface Image Processing) – інтерфейс передання завантажених в оперативну пам’ять навчальних даних на вхід процесу оброблення;

– інтерфейс IIRMT (Interface Image Recognition Model) – інтерфейс передавання вихідних (із модуля попередньої обробки) опрацьованих (очищених, токенізованих, нумеризованих) навчальних даних на вхід процесу навчання;

– інтерфейс IIR (Interface Image Recognition) – інтерфейс завантаження останньої найефективнішої версії збереженої навченої моделі для її подальшого використання в процесі розпізнавання попередньо оброблених даних завантажених користувачем;

– інтерфейс IUA (Interface User-Application) – інтерфейс передання результату обробки зображення користувача до модуля Інтерфейс користувача для його остаточного виведення;

– інтерфейс IIIP (Interface Input Image Processing) – інтерфейс передавання зображення, введеного користувачем, до модуля первинного оброблення;

– інтерфейс IPMR (Interface Processing-Machine Recognition) – інтерфейс передавання вихідних (із модуля попередньої обробки) опрацьованих фото для подальшого розпізнавання.

3.2. Висновки до розділу

У даному розділі було побудовано і описано модель системи виявлення військової техніки на зображенні у вигляді діаграми компонентів. Це діаграма структурного представлення системи, компоненти якої формуються за функціональною ознакою. Відношення між компонентами відображається через реалізацію відповідних інтерфейсів. Описано призначення та зміст компонентів.

4 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Обробка та підготовка даних

Першим пунктом у роботі з зображеннями є їх розмічування та анотація, вона потрібна для того щоб вказати що саме і де знаходиться на зображенні. Це потрібно для контрольованого навчання моделі, або іншими словами навчання з вчителем.

Одним з найпоширеніших форматів збереження анотацій до зображень є формат PASCAL VOC. Він зберігається у форматі .xml та вся інформація в ньому знаходиться у парних тегах які відповідають за певні атрибути, як от розмір зображення, назва, координати (позиція пікселів) 2-х протилежних кутів прямокутника що обмежують об’єкт.

```
<?xml version="1.0"?>
<annotation>
    <folder>images</folder>
    <filename>1.jpg</filename>
    <path>images/1.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>1280</width>
        <height>853</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>unlabelled</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>233</xmin>
            <ymin>309</ymin>
            <xmax>512</xmax>
            <ymax>414</ymax>
        </bndbox>
    </object>
</annotation>
```

Рис. 4.1 – Приклад наповнення файлу анотацій у форматі PASCAL VOC

Як бачимо в нас є координати 2-х протилежних кутів прямокутника який обмежує об'єкт проте модель YOLO отримує дані у вигляді: номер категорії, координати позиції центру обмежуючого прямокутника відносно усього

зображення, його ширину та висоту. Тому для перетворення ми виконуємо прості перетворення:

$$x_{centr} = \frac{x_{min}+x_{max}}{2*w} \quad (4.1)$$

$$y_{centr} = \frac{y_{min}+y_{max}}{2*h} \quad (4.2)$$

$$width = \frac{x_{max}-x_{min}}{w} \quad (4.3)$$

$$height = \frac{y_{max}-y_{min}}{h} \quad (4.4)$$

x_{centr}, y_{centr} – відносні координата центру обмежувального прямокутника у форматі YOLO;

$x_{min}, y_{min}, x_{max}, y_{max}$ – координати обмежувального прямокутника у форматі PASCAL VOC;

w, h – ширина та висота зображення відповідно;

$width, heigt$ – відносна ширина та висота обмежувального прямокутника у форматі YOLO відповідно.

Після переведення даних в потрібний формат анотації зберігаються у файлах з розширенням .txt у порядку $category, x_{centr}, y_{centr}, width, height$ через пробіл проте оскільки в нашому випадку категоризація не передбачена завданням дипломної роботи перший елемент опускається.

Для того щоб працювати з зображеннями спочатку слід перетворити їх з візуального формату у числовий, для цього ми переводимо кожен піксель зображення спочатку до RGB формату з яскравістю наповнення кожного кольору в діапазоні від 0 до 255, після чого проводимо нормалізацію розділивши це значення на 255. Отримуємо список для кожного пікселя який складається з 3-х значень що

відповідають кольорам в діапазоні від 0 до 1 – таке матричне представлення зображення називається тензор.

4.2. Модель YOLO

4.2.1. Структура нейронної мережі YOLOv1 та її компененти

Архітектура моделі YOLOv1 (You Only Look Once version 1) включає наступні компоненти:

1. Шар згортки (convolutional layer): YOLO використовує згорткову нейронну мережу для витягування ознак з вхідного зображення. Ці згорткові шари виконують операцію згортки за допомогою набору навчальних фільтрів, також відомих як ядра або детектори ознак на вхідному зображенні, що допомагають виявляти локальні ознаки.

На вхід згортковий шар приймає тривимірний тензор, який є активаціями або відповідями попереднього шару. У випадку зображення цей тензор зазвичай має розміри (висота, ширина, канали), де канали відносяться до кількості колірних каналів (наприклад, зображення RGB мають три канали).

Далі відбувається операція згортки: згортковий рівень застосовує набір фільтрів що навчаються, до вхідного тензора. Кожен фільтр є маленькою матрицею з шириною та висотою, меншими за вхідний тензор. Фільтр переміщується по вхідному тензору у вигляді ковзаючого вікна, виконуючи поелементне множення матриць та підсумовування в кожній позиції.

У кожній позиції фільтр обчислює скалярний добуток між його вагами (параметрами) і значеннями у відповідному просторовому розташуванні вхідного тензора. Ця операція застосовується незалежно до кожного каналу вхідного тензора. Результатом є одне значення, яке представляє відфільтровану відповідь у цій позиції.

Згортковий шар зазвичай має декілька фільтрів, від кількох десятків до сотень або більше. Кожен фільтр вчиться виявляти різні особливості або закономірності у вхідних даних. Операція згортки виконується для кожного фільтра, в результаті чого створюється набір карт функцій, також відомих як карти активації або канали функцій. Кожна карта функцій представляє відповідь певного фільтра на весь вхідний тензор. Таким чином зображення розбивається на декілька його відфільтрованих представлень кожне з яких виділяє або приглушує різні пікселі що дозволяє мережі виділяти певні ознаки об'єкту. Для прикладу проведемо обрахунок результату застосування одного фільтру згорткового шару на матриці 4 на 4 та фільтрі розміром 2 на 2:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \times \begin{bmatrix} f_1 & f_2 \\ f_3 & f_4 \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix} \tag{4.5}$$

де:

$$y_{11} = f_1 * x_{11} + f_2 * x_{12} + f_3 * x_{21} + f_4 * x_{22} \tag{4.6}$$

$$y_{12} = f_1 * x_{12} + f_2 * x_{13} + f_3 * x_{22} + f_4 * x_{22} \tag{4.7}$$

$$y_{13} = f_1 * x_{13} + f_2 * x_{42} + f_3 * x_{23} + f_4 * x_{24} \tag{4.8}$$

де: x – вхідні значення;

f- довільні значення фільтрів, проте зазвичай застосовують значення 1, 0, -1;

y- вихідні значення які подаються на функцію активації.

У згорткового шару є додаткові параметри для налаштування моделі. Заповнення (padding) це додаткові пікселі що містять 0 у всіх позиціях. Їх можна застосувати до вхідного тензора перед операцією згортки для того щоб додати додаткові межі пікселів навколо вхідних даних, що дозволить фільтрам обробляти

пікселі на краях вхідного тензора. Заповнення допомагає зберегти просторові розміри та зменшити втрату інформації біля меж зображення.

Параметр кроку (stride) визначає розмір кроку, з яким фільтр переміщується через вхідний тензор. Більше значення кроку зменшує просторові розміри отриманих карт функцій, що призводить до зменшення вибірки, тоді як крок, рівний 1 повністю зберігає просторові розміри.

На виході моделі застосовується функція активації (наприклад, в даному випадку ReLU - Rectified Linear Unit) до кожного виходу кожного фільтра, щоб ввести нелінійність і зафіксувати складні моделі в даних.

2. Шар об’єднання (pooling layer): цей шар, використовується в CNN для зменшення просторових розмірів карт функцій отриманих з шарів згортки. вибірки та вилучення найважливіших ознак із вхідних даних. Шар об’єднання допомагає досягти інваріантності та зменшити складність обчислень.

На вхід шар об’єднання приймає 3D-тензор або карту об’єктів, яка представляє собою активації з попереднього згорткового шару.

Подібно до згорткового шару операція об’єднання застосовується за допомогою підходу ковзного вікна. Невелике вікно фіксованого розміру, як правило, 2x2 або 3x3, ковзає над вхідною картою об’єктів. Це вікно рухається із заздалегідь визначеним кроком, зазвичай такого самого розміру, як і саме вікно, забезпечуючи області, що не перекриваються. У кожній такій області виконується попередньо визначена операція для агрегування інформації. Найпоширенішим типом операції об’єднання є максимальне об’єднання (max-pooling), де вибирається максимальне значення в області. Крім цього є ще, середнє об’єднання, яке обчислює середнє значення або об’єднання L2-нормалізації, яке обчислює квадратний корінь із суми квадратів значень у регіоні.

Результат агрегації передається на наступний рівень, відкидаючи інші значення. Цей процес зменшує просторові розміри карти функцій, ефективно зменшуючи її вибірку та зберігаючи найбільш важливу інформацію.

3. Повнозв’язний шар: цей шар відповідає за подальшу обробку та перетворення витягнутих ознак із згорткових шарів для виконання завдань класифікації та регресії [10]. Вони слідують за згортковими шарами в мережевій архітектурі моделі YOLOv1 і дають остаточну відповідь стосовно присутності об’єкту на зображенні і до якого класу він відноситься.

Вхідними даними для повнозв’язних шарів є тензор сплощених ознак, отриманий із вихідних даних попередніх згорткових шарів. Ця операція зведення перетворює тривимірний тензор (просторові розміри × канали) в одновимірний тензор (вектор), об’єднуючи активації з усіх згорткових фільтрів. Кожен нейрон у повністю зв’язаних шарах пов’язаний з кожним елементом активації, зв’язки представлені ваговими коефіцієнтами, які налаштовуються під час навчання і які визначають силу зв’язку між кожним нейроном і вхідними активаціями. Після отримання значення застосовується функція активації, зазвичай така ж як і в згоркових нейронах - ReLU (Rectified Linear Unit) або сигмоїда.

Останній повнозв’язний шар проектується відповідно до конкретного завдання, яке виконується. Для виявлення об’єктів у YOLOv1 цей рівень зазвичай складається з нейронів, що представляють різні класи та параметри регресії для передбачень обмежувальної рамки. Функція активації, застосована до вихідного рівня, залежить від завдання, наприклад softmax для багатокласової класифікації або лінійна активація для регресії.

4.2.2. Архітектура YOLOv1

Архітектура нейронної мережі YOLOv1 складається з 24 згорткових шарів, 4 шарів об'єднання і 2 повнозв'язних шарів. На малюнку наведена візуалізація архітектури з наведеними розмірами комірок та додатковими параметрами.

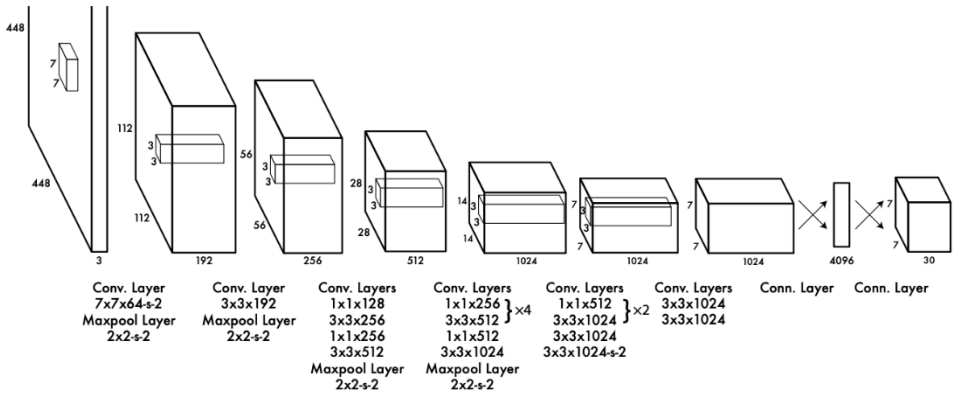


Рис. 4.2 – Зображення архітектури нейронної моделі YOLOv1 [9]

На вхід моделі подається тензор який відповідає математичному представленню зображення розмірами 448 на 448 пікселів та 3-ма кольоровими каналами.

Перший згортковий шар має розмір комірки 7 на 7 з 64 вихідними фільтрами для кожного кольору та кроком 2, з якого ми отримуємо тензор розмірами 224 на 224 на 192. Отримані результати передаються у функцію активації Rectified Linear Unit (ReLU) яка визначається формулою:

$$f(x) = \max(0, x) \tag{4.9}$$

Далі отриманий результат передається до шару об’єднання де використовується метод max-pooling з коміркою розміром 2 на 2 та кроком 2, тому на виході ми отримуємо тензор розміром 112 на 112 на 192. Далі відбуваються перетворення у відповідності до зображення.

Перед подачею результатів на повнозв'язні шари початкове зображення розбивається на сітку з фіксованими розмірами $S \times S$. Кожна комірка сітки відповідає певному регіону зображення. Відповідно для них обраховується значення загальної обмежувальної рамки.

В кінці для кожної комірки сітки алгоритм YOLOv1 прогнозує один або кілька огороджувальних рамок (bounding boxes) саме для цього і використовуються шари

згорткового детектора, що виконуються на останній згортковій карті. На виході моделі YOLOv1 ми отримуємо вектор який визначає клас об’єкту, обмежувальну рамку та імовірність його знаходження в ній.

Алгоритм YOLOv1 розрахований на виявлення 20 класів та прогнозування 2-х основних обмежуючих рамок, тому вектор-результат для цієї моделі має вигляд:

$$[c_1, c_2, \dots, c_{20}, p_{c1}, x_1, y_1, w_1, h_1, p_{c2}, x_2, y_2, w_2, h_2] \tag{4.10}$$

де: c – імовірність належності об’єкта до класу;

p – імовірність знаходження певного об’єкту у комірці;

x, y, w, h - координати центру обмежуючого прямокутника та його ширина та висота відповідно.

4.1.3. Обчислення помилки моделі

Помилка моделі потрібна щоб за допомогою методу зворотнього розповсюдження помилки оптимізатор міг навчати нейрони, тобто коригувати значення їх ваг так щоб вони видавали правильний результат у відповідності до навчальних даних.

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j-0}^{B} 1_{ij}^{obj} [\,(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j-0}^{B} 1_{ij}^{obj} [\,\left(\sqrt{w_i} - \sqrt{\widehat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (\mathrm{C}_i - \hat{\mathrm{C}}_i)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (\mathrm{C}_i - \hat{\mathrm{C}}_i)^2 \\
&+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{4.11}$$

де: x_i – реальне значення координати центра обмежувальної рамки об'єкта по горизонтальній осі;

$\hat{x}_i$ – прогнозоване значення координати центра обмежувальної рамки об'єкта по горизонтальній осі;

y_i – реальне значення координати центра обмежувальної рамки об'єкта по вертикальній осі;

$\hat{y}_i$ – прогнозоване значення координати центра обмежувальної рамки об'єкта по вертикальній осі;

Перший елемент функції відноситься до обчислення помилки передбачення координат x та y обмежувальної рамки.

Другий елемент відноситься до обчислення помилки передбачення ширини w та висоти h обмежувальної рамки.

Третій елемент відноситься до обчислення помилки впевненості моделі у відношенні об’єкту до певного класу, четвертий відноситься до впевненості відсутності відношення об’єкту до певного класу.

Останній елемент відноситься до обчислення помилки пов’язаною з впевненістю що об’єкт є у обмежувальній рамці.

Таким чином ми отримуємо функцію що враховує усі потрібні нам аспекти інформації що ми отримуємо на виході з моделі. Далі ми обраховуємо помилку та відбувається навчання моделі. Цей процес продовжується поки не буде досягнутий потрібний рівень точності або поки не закінчаться дані для навчання.

4.3. Метрики оцінки якості моделей

Обчислюються наступні метрики якості класифікації для моделей машинного навчання.

Accuracy (точність) – доля правильних відповідей алгоритму. Обчислюється за формулою:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (4.12)$$

де TP $(True\ Positive)$ – правильне спрацьовування алгоритму;

TN $(True\ Negative)$ – алгоритм правильно спрацював на хибне значення;

FP $(False\ Positive)$ – хибне спрацьовування алгоритму;

FN $(False\ Negative)$ – алгоритм помилився на правильне значення.

Precision (влучність) – здатність алгоритму відрізняти потрібний клас від решти:

$$precision = \frac{TP}{TP+FP} \quad (4.13)$$

IoU (Intersection Over Union) - це число, яке кількісно визначає ступінь перекриття між двома блоками. У разі виявлення та сегментації об’єктів IoU оцінює перекриття справжньої рамки (Ground Truth) і передбаченим регіоном (Prediction region). В даному випадку True Positive: область перетину між Ground Truth (GT) і сегментаційною маскою (S). Тобто:

$$TP = GT \cap S \quad (4.14)$$

False Positive: передбачена область за межами справжньої рамки. Тобто:

$$FP = (GT + S) - S \quad (4.15)$$

False Negative результат: кількість пікселів у зоні базової істинності, яку модель не змогла передбачити.

$$FN = (GT + S) - S \quad (4.16)$$

Ми знаємо, що IoU – це відношення площі перетину до об’єднаної площі передбачення та базової істинності. Оскільки значення TP, FP і FN є нічим іншим, як площами або кількістю пікселів; ми можемо записати IoU наступним чином.

$$IoU = \frac{TP}{(TP+FP+FN)} \tag{4.17}$$

4.4. Модифікація обраного методу

Для вирішення задачі поставленій у цій роботі, а саме виявлення військової техніки на зображенні, нам не потрібно виконувати задачу класифікації, яка передбачена у запропонованій моделі алгоритму. Щоб прибрати цей функціонал нам доведеться зробити зміни у останніх 2-х повнозв’язних шарах, а саме: додати згортковий шар з розмірами ковзаючого вікна 4 на 4 та 512 фільтрами, застосувати перетворення наступного шару з 3-вимірного у одновимірний вектор, після цього застосувати повнозв’язний шар з 245 нейронами та на виході знову застосувати перетворення шару, але цього разу з одно-вимірного до 3-вимірного шару який відповідає розмірам сітки на яку розбивається зображення на вході, тобто S на S та 5 вихідними значеннями.

В даній задачі вихідними значеннями є:

$$[\,p, x, y, w, h] \tag{4.18}$$

де: p – імовірність знаходження певного об’єкту у комірці;

x, y, w, h - координати центру обмежуючого прямокутника та його ширина та висота відповідно.

Таким чином для кожної клітинки, на які розбивається зображення, ми отримуємо координати центру обмежувальної рамки об’єкта, його висоту та ширину, а також вірогідність його знаходження там, таким чином модель виконує поставлену задачу тільки передбачаючи місцезнаходження об’єкта в обмеженій області з певною впевненістю.

Також була змінена функція обрахунку помилки, адже у моделі YOLOv1 враховується помилка яку робить модель при класифікації об’єктів, а оскільки наша задача не передбачає завдання класифікації ми можемо скоротити функцію помилки до 3-х частин: обрахування помилки центральних координат обмежувальної рамки, висоти та ширини обмежувальної рамки та впевненості моделі у місцезнаходженні об’єкта. Кінцева функція помилки моделі виглядає так:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j-0}^{B} 1_{ij}^{obj} [\,(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j-0}^{B} 1_{ij}^{obj} [\,\left(\sqrt{w_i} - \sqrt{\widehat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2]$$
$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \qquad (4.19)$$

Під час перших навчань моделі виникли проблеми з її швидкістю та самим процесом навчанням. Першою проблемою був час навчання моделі – за попередніми підрахунками він мав складати 12 годин безперервного навчання. Другою проблемою стали попередні результати навчання моделі, хоча спершу помилка моделі зменшувалась, після декількох епох навчання помилка почала збільшуватись. Подальші тести показали що використання функції активації ReLU може призводити до того що нейрон ніколи не буде активований і відповідно ніколи не відбудеться покращення значення його ваг. Це може призводити до того що активація цих нейронів сильно впливатиме на результат і відповідно збільшуватиме помилку.

Щоб вирішити першу проблему було вирішено додати нормалізацію ваг на кожному згортковому та повнозв’язному шарі. Нормалізація ваг виконується для того щоб зменшити абсолютне відхилення значень ваг від 0 та призводить до більш-стабільних результатів нейронів на кожній ітерації, а також зменшує кількість обчислень що призводить до зменшення часу виконання ітерації навчання на кожному зображенні. Також було зменшено кількість шарів в моделі шлях вилучення шарів що дублюються, таким чином з 24 згорткових шарів залишилось 13.

Другу проблему вирішено було за допомогою заміни функції активації ReLU на функцію активації LeakyReLU. Функція має вигляд:

$$f(x) = \max(0.1 * x, x) \tag{4.20}$$

Також було вирішено збільшити початкове ковзаюче вікно з розмірів 7 на 7 до розмірів 32 на 32. Це дозволяє моделі захоплювати більші зони і, відповідно, оперувати більшими представленнями про наповнення зображення.

4.5. Висновки до розділу

У цьому розділі було розглянуто математичні методи обробки зображень, підготовки їх до числового вигляду та токенізації. Для попередньої обробки зображень та анотацій використовувались прості математичні перетворення. Після проведеної попередньої обробки зображення готові до навчання, валідації та тестування моделі.

Також описано математичне забезпечення методів машинного навчання розв’язання задачі виявлення об’єктів на фото, а саме описані складові частини згорткової нейронної мережі архітектури YOLOv1: згорткові шари, шари

об’єднання, повнозв’язні шари. Розглянуто алгоритм навчання моделі та обчислення помилки моделі для зміни вагових коефіцієнтів методом зворотнього поширення помилки. Встановлено, що для оцінки якості виявлення об’єктів на фото можна використати наступні метрики: accuracy, precision, IoU. Описано математичне забезпечення наведених метрик. Запропоновані модифікації методу для вирішення поставленої задачі.

Визначені математичні алгоритми які слід покращити для кожного кроку програмної реалізації з яких: збільшення нейронної сітки, зміна функції активації з ReLu на LeakedReLu для уникнення неактивованих нейронів, модифікації функції обчислення помилки відповідно до поставленої задачі, використання нормалізації ваг для усунення великих стрибків на кожній ітерації та балансуванні шарів. Незначні зміни у архітектурі вихідних шарів для вирішення поставленої задачі.

5 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

5.1. Структура програми

Структура програми складається з 3-х основних частин: підготовка даних, ініціалізація та навчання моделі, тестування моделі та режим розпізнавання на фотографіях користувача.

Усе програмне забезпечення реалізовано за допомогою мови програмування Python та її бібліотек. Python - це високорівнева мова програмування, яка є найпопулярнішою в сфері машинного навчання. Вона є зручною в користуванні оскільки має досить велику мережу бібліотек які значно розширюють функціонал самої мови програмування і мають великий спектр інструментів зручних у використанні в сфері аналітики та машинного навчання.

Ось опис бібліотек, які використовуються в програмному забезпеченні:

sys: ця бібліотека надає доступ до системних параметрів і функцій. Вона дозволяє взаємодіяти з інтерпретатором Python, включаючи аргументи командного рядка та системні функції.

os: ця бібліотека забезпечує спосіб взаємодії з операційною системою. Вона надає функції для роботи з файлами та каталогами, такі як створення, видалення, перейменування та перелік файлів і каталогів. За допомогою неї відбувається взаємодія з файлами у вибірках для навчання, валідації та тестування.

natsort: ця бібліотека надає алгоритми сортування для списків рядків. Це допомагає сортувати назви файлів або будь-які інші рядки зручним для людини способом, беручи до уваги числові значення в рядках.

imutils: ця бібліотека надає набір службових функцій для роботи із зображеннями. Функції включають зміну розміру, обертання, обрізання та інші операції обробки зображень.

cv2: Ця бібліотека є OpenCV для Python. Він забезпечує різні функції комп’ютерного зору та обробки зображень, такі як читання та запис зображень,

маніпулювання зображеннями, виявлення об’єктів тощо. В програмному забезпеченні використовується для візуалізації результатів роботи моделі.

matplotlib: ця бібліотека створена для малювання та візуалізації на Python. Вона надає функції для створення та налаштування графіків, діаграм і фігур. Вона також використовується для візуалізації результуючих зображень.

numpy: ця бібліотека є основним пакетом для наукових обчислень на Python. Вона забезпечує підтримку великих багатовимірних масивів і матриць разом із набором математичних функцій для роботи з цими масивами.

tensorflow і keras: ці бібліотеки використовуються для створення та навчання моделей глибокого навчання. TensorFlow — популярна платформа глибокого навчання з відкритим вихідним кодом, а Keras — це високорівневий API для нейронних мереж, який працює на основі TensorFlow. Вони надають функції та інструменти для побудови нейронних мереж, визначення рівнів, компіляції моделей і моделей навчання.

path: цей клас із модуля pathlib надає об’єктно-орієнтований інтерфейс для керування шляхами файлів і каталогів. Він спрощує роботу зі шляхами до файлів, надаючи зручні методи та властивості.

Функції цих бібліотек використовуються в усіх структурах програми.

5.2. Підготовка даних

Частина підготовки даних включає в себе завантаження даних зображень та анотацій до них зі сховища комп’ютера, перетворення файлів анотацій з формату PASCAL VOC до формату YOLO та збереженні відповідних файлів анотацій у файлах для подальшого використання. Після цього відбувається токенізація завантажених зображень та розбиття їх на тренувальні партії, які ітеративно будуть подаватись на вхід моделі під час навчання.

Завантаження даних відбувається за допомогою методів бібліотеки path, а саме Path. Передаючи в цю функцію шлях до каталогу вона повертає список усіх об’єктів що там знаходяться, тому з її допомогою ми можемо пройтись по всім файлам анотацій та застосувати наші перетворення. Для відкриття файлів і отримання потрібної інформації застосовується метод з вбудованої бібліотеки Python xml.etree.ElementTree який розбиває файл з розширенням .xml в формат списку з якого ми просто витягаємо потрібні нам значення координат обмежувальної рамки.

5.3. Створення моделі

Далі відбувається побудова структури моделі, функції отримання даних та функції обрахунку посилки моделі, налаштування гіперпараметрів таких як функція оптимізації, крок оптимізації, файл збереження, файл завантаження моделі, кількість епох, розміри навчальних партій.

Побудова структури моделі відбувається за допомогою методів бібліотек tensoflow та keras. Метод Sequential ініціалізовує нейронну мережу, а наступні методи, такі як: tf.keras.layers.Conv2D, tf.keras.layers.BatchNormalization, LeakyReLU, tf.keras.layers.MaxPooling2D додають шари до нейронної мережі з налаштуванням гіперпарамертів в залежності від потрібного функціоналу.

Наприклад гіперпараметрами для tf.keras.layers.Conv2D є: кількість фільтрів (також відомих як вихідні канали) у згортковому шарі, розмір згорткового ядра, схема доповнення, яка буде використана, параметр який встановлює вхідну форму моделі.

Для методу tf.keras.layers.MaxPooling2D гіперпараметрами також є розміри ковзаючого вікна та крок руху цього вікна.

Загалом модель складається з 13 згорткових шарів після яких слідують шари нормалізації ваг та функції активація LeykuReLU, також в моделі присутні 5 шарів об’єднання і на виході моделі є повнозв’язний шар.

Функція отримання даних використовує бібліотеки os та sys для звернення до файлів у папках та завантаження їх до списку. Далі цей список розбивається на тренувальні вибірки, в роботі їх кількість обрана як 32 зображення в кожній вибірці.

Потім ініціалізується функція обрахунку помилки моделі та оптимізатор. В роботі використаний оптимізатор Adam з наступними гіперпараметрами: learning_rate: цей параметр визначає розмір кроку або швидкість навчання на кожній ітерації процесу оптимізації. Він контролює величину оновлення параметрів моделі. Швидкість навчання встановлена 0,00005. beta_1: це імпульс, він контролює експоненціальну швидкість спаду для першої оцінки моменту (середнього значення) градієнтів. Він визначає, наскільки минулі градієнти впливають на поточне оновлення. Встановлене значення становить 0,9. beta_2: цей параметр контролює експоненціальну швидкість спаду для оцінки другого моменту (нецентрована дисперсія) градієнтів. Він визначає швидкість, з якою вплив минулих градієнтів зменшується. Зазвичай використовується значення 0,999 як і в нашому випадку. epsilon: цей параметр є малою константою, доданою до знаменника, щоб запобігти діленню на нуль. Це забезпечує чисельну стабільність під час розрахунку адаптивної швидкості навчання. У цьому випадку використовується значення 0,00000001.

В кінці відбувається ініціалізація моделі на основі побудованої структури за допомогою методу compile() куди передається функція обрахунку помилки моделі та оптимізатор. Далі запускається процес навчання за допомогою методу fit() в який передаються навчальні дані, інформація про файл збереження та кількість повідомлень що виводяться у консоль. Під час навчання модель зберігає свої дані про ваги коефіцієнтів та значення середньої помилки моделі на тренувальних даних у файл, що дає змогу відновити інформацію у випадку відключення світла або виникнення помилки під час виконання процесу навчання оскільки він займає довгий період часу.

5.4. Навчання моделі

Навчання моделі відбувалось 6 годин та 22 хвилини. За цей час модель навчилась на вибірці з 4481 зображень та файлів анотацій до них.

На рисунку 5.1 показаний графік того як змінювалось значення помилки моделі на кожній ітерації процесу навчання.

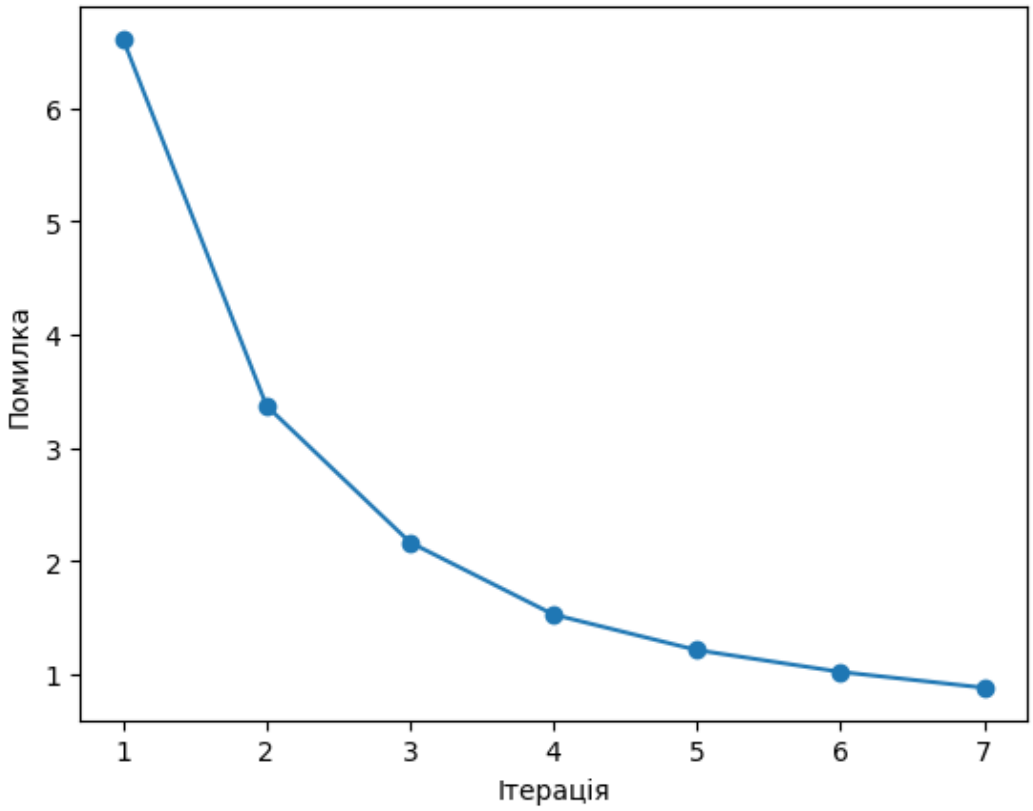


Рис 5.1 – Графік значення похибки на кожній ітерації

Початкове значення помилки становило 6.6071 в той час як після завершення процедури навчання помилка становила лише 0.8821.

5.5. Тестування моделі та аналіз отриманих результатів

Тестування моделі проводилось на 679 зображеннях. Для тестування у модель передаються зображення які модель ще не зустрічала та виконуються обчислення співпадіння її результату з відомою нам інформацією.

Рис 5.2 – Приклад візуалізації результату роботи моделі

Після передачі тестового зображення в моделі та отримання передбачення обмежуючої рамки об’єкта на зображенні та значення впевненості моделі у тому що об’єкт знаходиться всередині обмежувальної рамки відбувається візуалізація отриманої інформації разом з відомими нам даними які ми отримуємо з файлу анотацій.

Також ми отримуємо інформацією про присутність та відсутність об’єктів військової техніки на фото, що дозволяє нам обрахувати метрики оцінки якості моделі.

Таблиця 5.1 Метрики оцінки якості моделі в залежності від рівня впевненості

Рівень впевненості	Вірні прогнози	Невірні прогнози	Точність (Accuracy)	Влучність (Precision)	IoU
0.2	366	313	0.539	1.0	0.6457
0.3	390	289	0.574	0.984	0.6285
0.4	435	244	0.641	0.959	0.6086
0.5	459	220	0.676	0.879	0.5615
0.51	462	218	0.680	0.859	0.5508
0.52	464	215	0.683	0.846	0.5414
0.53	457	222	0.673	0.813	0.5229
0.54	452	227	0.666	0.791	0.5087
0.55	360	219	0.678	0.788	0.505
0.59	459	220	0.676	0.708	0.46
0.6	453	226	0.668	0.675	0.4403
0.7	399	280	0.588	0.369	0.2401
0.8	346	333	0.509	0.110	0.0714

Як бачимо найкращу точність модель показує при рівні впевненості 0.52. При менших значеннях модель просто всюди вказує про те що на йото є об’єкт, а при більших значеннях перестає бути впевненою у місцезнаходженні об’єктів і відкидає вірні відповіді.

Матриця невідповідностей для рівня впевненості 0.52 має такий вигляд:

TP (True Positive) = 307;

TN (True Negative) = 157;

FP (False Positive) = 159;

FN (False Negative) = 56.

Слід звернути увагу також на показник влучності який показує рівень виявлення зображень де справді є об’єкти військової техніки. Наприклад при рівні впевненості 0.4 модель все таки показує меншу точність але такий показник як влучність залишається на рівні 95.9%, це означає що такий рівень впевненості можна використовувати у задачі фільтрації, де ми хочемо точно відкинути об’єкти на яких немає об’єктів військової техніки. Таким чином ми могли б відкинути 20% зображень що точно не містять військової техніки при цьому зберігши основну вибірку зображень де ми з певною впевненістю можемо казати що там присутні об’єкти військової техніки.

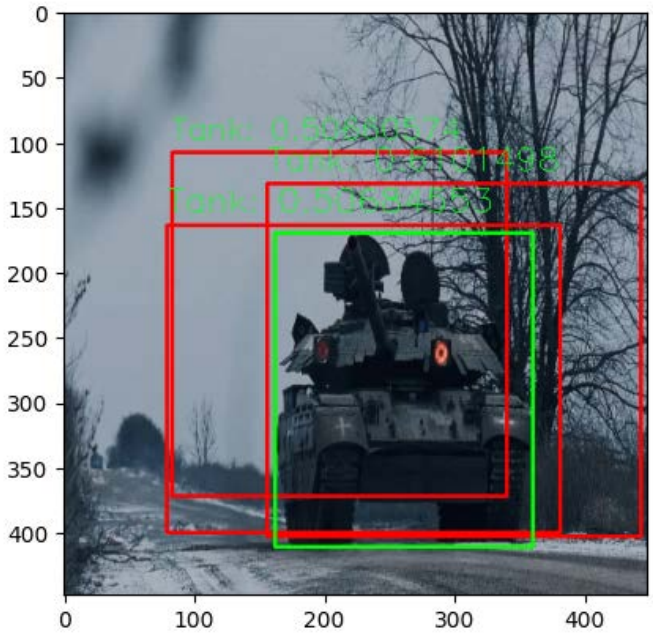


Рис 5.3 – Візуалізація результату роботи моделі з рівнем впевненості 0.2

Показник Intersection over Union зі збільшенням рівня впевненості постійно падає. Це пов’язано з тим що при меншому рівні впевненості модель генерує більшу кількість рамок які частіше пересікаються з вірною обмежувальною рамкою. І навпаки, при більшому рівні впевненості модель взагалі може не видавати обмежувальної рамки для зображення на якому є об’єкт, що означає що значення

IoU дорівнюватиме 0, що й призводить до падіння середнього значення цього показнику.

5.6. Висновки до розділу

У цьому розділі було описано програмне забезпечення системи обробки зображень та системи перетворення та збереження анотацій до зображень, які використовуються відповідними компонентами моделі. Наведено складові згорткової нейронної моделі та їх створення за допомогою програмного забезпечення.

Також описано програмне забезпечення методів машинного навчання для реалізації нейронної моделі розв’язання задачі виявлення об’єкту на фото, а саме: складові згорткової нейронної мережі, навчання та тестування моделі.

Встановлено, що найкращі результати були отримані при рівні впевненості моделі на рівні 0.52. Були проаналізовані результати роботи моделі на тестових даних за допомогою визначених метрик.

Окремо слід відзначити швидкість роботи моделі. Швидкість розпізнавання об’єкта на навченій моделі складає від 0.1 секунди до 0.2 секунд що дозволяє використовувати отриману модель на відео і отримувати результати в реальному часі.

6 ВЕРИФІКАЦІЯ ТА ВАЛІДАЦІЯ

Верифікація. У процесі написання роботи було створено систему виявлення військової техніки на зображенні. Проведено аналіз методів машинного навчання, за допомогою яких розв’язувалась задача виявлення: згорткова нейронна мережа, згорткові, об’єднуючі та повнозв’язні шари нейронної мережі. Найкращим результатом виявлення об’єктів військової техніки на валідаційній вибірці створеною моделлю була точність у 80,7% при рівні впевненості 0.55.

Валідація. Найкращі результати виявлення об’єктів на тестовій вибірці становлять 68.3% при рівні впевненості 0.52. Середнє значення IoU при цьому становить 0.5413. Також слід відзначити що при рівні впевненості 0.4 точність методу менша і становить 64.1%, натомість таку модель можна використовувати у випадках коли ми хочемо відсіяти частину зображень де ми точно впевнені що техніка відсутня. В такому випадку кількість True Negative становить від 10 до 20 % в залежності від вибірки.

Система може бути допоміжним інструментом для військових аналітиків та систем відеоспостереження як інструмент автоматизації частини роботи людини.

ВИСНОВКИ

У результаті виконаної роботи:

а) було розглянуто наявні методи розв’язання поставленої задачі виявлення об’єктів військової техніки (метод Віоли-Джонса, метод масштабно-інваріантної трансформації ознак, гістограма орієнтованих градієнтів, згорткова нейронна мережа, ієрархічний аналіз, штучні нейронні мережі, метод You Look Only Once) та програмні рішення, у яких реалізовані описані методи. У даній роботі проблема виявлення військової техніки на фото розглядається як задача комп’ютерного зору з виявлення присутності об’єкта на зображенні та обмеженні його місцезнаходження, тому обраний метод машинного навчання розв’язання задачі на основі моделі розпізнавання об’єктів YOLO;

б) було побудовано систему виявлення військової техніки на фото у вигляді діаграми компонентів – діаграми структурного представлення системи. Її компоненти формуються за функціональною ознакою, а відношення між компонентами відображається через реалізацію відповідних інтерфейсів. Описано призначення та зміст компонентів;

в) було розглянуто математичні методи, які використовуються відповідними компонентами. Також описано математичне забезпечення методів машинного навчання розв’язання задачі виявлення об’єктів, а саме: згорткової нейронної мережі та її компонентів, згорткового шару, об’єднуючого шару, повнозв’язного шару, функції активації та функції обчислення помилки моделі. Встановлено, що для оцінки якості обраного методі виявлення військової техніки на фото можна використати наступні метрики: accuracy, precision, IoU. Описано математичне забезпечення наведених метрик;

г) було розроблено програмне забезпечення системи виявлення військової техніки на фото. Описано програмне забезпечення використаних методів машинного навчання розв’язання задачі, а саме: згорткової нейронної мережі та її компонентів,

згорткового шару, об’єднуючого шару, повнозв’язного шару, функції активації та функції обчислення помилки моделі. Проведено аналіз отриманих результатів. Встановлено, що найкраща точність моделі досягається при рівні впевненості 0.52. Значення точності моделі становить 68.3% на тестовій вибірці, що є цілком пристойним результатом;

д) проведено верифікацію та валідацію системи, у результаті чого встановлено відповідність системи вимогам: модель забезпечує розпізнавання об’єктів військової техніки на фото. Система може бути допоміжним інструментом для військових аналітиків та систем відеоспостереження як інструмент автоматизації частини роботи людини.

ПЕРЕЛІК ПОСИЛАНЬ

1. Google Lens [Електронний ресурс] – https://lens.google/
2. Aspose [Електронний ресурс] – https://products.aspose.app/imaging/ru/object-detection/car
3. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.
4. Lowe, David G. (1999). "Object recognition from local scale-invariant features" (PDF). Proceedings of the International Conference on Computer Vision. Vol. 2. pp. 1150–1157.
5. Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
6. Bhadeshia H. K. D. H. (1999). Neural Networks in Materials Science. ISIJ International 39 (10): 966–979.
7. Zitnick, C. Lawrence, and P. Dollar. "Edge boxes: Locating object proposals from edges." Computer Vision-ECCV. Springer International Publishing. Pages 391-4050. 2014.
8. Girshick, R., J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." CVPR '14 Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition. Pages 580-587. 2014
9. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

10. M. B. Blaschko and C. H. Lampert, "Learning to localize objects with structured output regression," In Computer Vision ECCV 2008, pp. 2--15, Springer, 2008.
11. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [Електронний ресурс] / [J. Donahue, Y. Jia, V. Oriol та ін.] // arXiv. – 2013. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.1310.1531.
12. roboflow [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/.
13. Img Lab [Електронний ресурс] – Режим доступу до ресурсу: https://imglab.in/.
14. Dataset 1 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/dsfsfg/cfghj/dataset/1.
15. Dataset 2 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/vgty/tanks-gxufc/dataset/10.
16. Dataset 3 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/playground-ls3ru/tanks-65plh/dataset/1.
17. Dataset 4 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/kmutnb-pncxb/ai_wartank/dataset/4.
18. Dataset 5 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/uce03211-gmail-com/go-plg7l/dataset/5.
19. Dataset 6 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/beijing-institute-of-techology/tank-rr0k7/dataset/6.
20. Dataset 7 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/t-s-nanda-kishore-1alay/13k_dataset/dataset/2.
21. Dataset 8 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/university-of-georgia/military_tanks_planes_helicopters_ifv_apc_artilery_trucks/dataset/2.
22. Dataset 9 [Електронний ресурс] – Режим доступу до ресурсу: https://universe.roboflow.com/aerial-detection/pure-tank/dataset/1.

Додаток А

Лістинги програм

```
import pandas as pd
import numpy as np

import xml.etree.ElementTree as ET
from pathlib import Path

def pascal_voc_to_coco(x1, y1, x2, y2):
    return [x1,y1, x2 - x1, y2 - y1]


path = 'C:/Users/Asus/Univercity 4 lvl/Diploma/xml_new'
def parsethefile(listOfFiles):
    for myFile in listOfFiles.iterdir():

        filePath = myFile
        parser = ET.XMLParser(encoding="utf-8")
        targetTree = ET.parse(filePath, parser=parser)
        rootTag = targetTree.getroot()
        width = int(list(rootTag)[4][0].text)
        height = int(list(rootTag)[4][1].text)
        #print(width,height)
        try:
            xmin = int(round(float(list(rootTag)[6][5][0].text)))
            ymin = int(round(float(list(rootTag)[6][5][2].text)))
            xmax = int(round(float(list(rootTag)[6][5][1].text)))
            ymax = int(round(float(list(rootTag)[6][5][3].text)))
            category = 0

            #new_size = convertLabels(xmin, ymin, xmax, ymax, height, width, category)
            new_size = pascal_voc_to_coco(xmin, ymin, xmax, ymax)
            to_file = ' '.join([str(round(new_size,17)) for new_size in new_size])
```

```
            name_to_write = str(filePath).split("\\")
            #print(name_to_write)
            name_to_write = name_to_write[-1].replace('_pvoc_imglab.xml','')
            #print(name_to_write)
            with                                    open('C:/Users/Asus/Univercity                    4
lvl/Diploma/prepared_annotations/{}.txt'.format(name_to_write), 'w+') as f:
                f.write(to_file)
        except:
            print(filePath)
# Replace this with the class label of your object
parsethefile(Path(path))

import sys
import os
from natsort import natsorted
import imutils
import cv2
import matplotlib.pyplot as plt
import numpy as np
from math import trunc
import json

import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
from keras.layers import LeakyReLU
from keras.regularizers import l2
from keras.models import Model
from pathlib import Path
```

```
def listdir_fullpath(d):
    return [os.path.join(d, f) for f in os.listdir(d)]

bboxes_paths = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\labels_new'))
#img_paths = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\YOLOv1\data\train\images'))
img_paths = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\images_raw'))
assert len(bboxes_paths) == len(img_paths), f"Number of bboxes_paths doesn't correspond with number of images!"

#Read all the text files and create a list of list of bounding boxes for the training, one list per image
bboxes = []*len(bboxes_paths)
for path in bboxes_paths:
    bboxes_it = []
    file = open(path, 'r')
    Lines = file.readlines()
    for line in Lines:
        box = line.split(' ')
        if(not(int(box[1]) == 0 and int(box[2]) == 0 and int(box[3]) == 0 and int(box[4]) == 0)):
            bboxes_it.append([int(x) for x in box])
    bboxes.append(bboxes_it)

assert(len(bboxes) == len(img_paths))

class DataGenerator(tf.keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, path_list, bboxes_list, batch_size=32, dim=(448,448,3),
                 divisions=7, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.path_list = path_list
        self.S = divisions
        self.bboxes_list = bboxes_list
```

```
    self.shuffle = shuffle
    self.on_epoch_end() #triggered at beginning and end of each epoch
    self.cell_size = dim[0]/divisions

def __len__(self):
    'Denotes the number of batches per epoch'
    return int(np.floor(len(self.path_list) / self.batch_size))

def __getitem__(self, index):
    'Generate one batch of data'
     # Generate indexes of the batch
    indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

    # Generate data
    X, Y = self.__data_generation(indexes)

    return X, Y

def on_epoch_end(self):
    'Updates indexes after each epoch'
    self.indexes = np.arange(len(self.path_list))
    if self.shuffle == True: # For more robust data
        np.random.shuffle(self.indexes)

def __data_generation(self, indexes):
    'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
    # Initialization
    X = np.empty((self.batch_size, *self.dim))
    Y = np.empty((self.batch_size,self.S,self.S,5))

    batch_num = 0
    # Generate data
    for i in indexes:
        original_img = load_img(self.path_list[i])
```

```
        width, height = original_img.size
        # load the image with the required size and calculate scale factors
        image = load_img(self.path_list[i], target_size=(448, 448))
        scale_w = 448 / width
        scale_h = 448 / height
        image = img_to_array(image)
        # scale pixel values to [0, 1]
        image = image.astype('float32')
        image /= 255.0
        y_img = np.zeros((self.S,self.S,5))
        for box in self.bboxes_list[i]:

            xleft = int(box[0] * scale_w)
            yleft = int(box[1] * scale_h)
            b_width = int(box[2] * scale_w)
            b_height = int(box[3] * scale_h)
            ox = xleft + b_width/2
            oy = yleft + b_height/2
            # Calculate the coordinates of the cell in the grid that contains the center
            grid_col = trunc(ox/self.cell_size)
            grid_row = trunc(oy/self.cell_size)
            # Calculate the coordinates of the center of the bbox w.r.t the associated cell; (0,0) top left and
(1,1) bottom right corners of the cell
            ox_cell = (ox - (grid_col)*self.cell_size)/self.cell_size
            oy_cell = (oy - (grid_row)*self.cell_size)/self.cell_size
            # Calculate the width and height of the bbox in terms of cell size, a bbox of width 448/S(cell
size) will have grid_width = 1
            grid_width = b_width/448
            grid_heigth = b_height/448
            # Put the results into y; 1 represent the probability of the class
            y = [1,ox_cell,oy_cell,grid_width,grid_heigth]
            #print(i)
            #print(y)
```

```
            y_img[grid_row][grid_col] = y
        #print(image)
        # Store sample
        X[batch_num,] = image

        # Store grid
        Y[batch_num,] = y_img

        batch_num += 1

    return X, Y

def custom_loss(y_true, y_pred):
    # y_true (Batch size, 7, 7, 5)
    # y_pred (Batch size, 7, 7, 5)

    mse = tf.keras.losses.MeanSquaredError(reduction = "sum") # Define the SUM squared error loss
    predictions = tf.reshape(y_pred,(-1,7,7,5)) # The predictions are a tensor, need some reshaping to manipulate it

    exists_box = tf.expand_dims(y_true[...,0], 3) # A box exists if the first entry of the cell is equal to 1


    pred_box = exists_box*predictions[...,1:5] #Calculate only loss for the cells that contain a box
    target_box = exists_box*y_true[...,1:5] #Target boxes

    epsilon = tf.fill(tf.shape(pred_box[..., 2:4]), 1e-6) #Needed to avoid divergence of square root derivatives in back propagation

    # width and height are penalyzed using the square root, however predictions can be negative so multiply by sign in order to obtain positive
    # and take absoulte value in the square root
    wh_pred = tf.math.sign(pred_box[...,3:5]) * tf.math.sqrt(tf.math.abs(pred_box[...,3:5] + epsilon))
    wh_targ = tf.math.sqrt(target_box[...,3:5] + epsilon)
```

```
    # Get also centers
    xy_pred = pred_box[...,1:3]
    xy_true = target_box[...,1:3]

    # Concatenate the new xy and wh in order to calculate sum squared root
    final_pred_box = tf.concat([xy_pred,wh_pred], axis = 3)
    final_true_box = tf.concat([xy_true,wh_targ], axis = 3)
    box_loss = mse(tf.reshape(final_pred_box, (-1, tf.shape(final_pred_box)[-
1])),tf.reshape(final_true_box, (-1, tf.shape(final_true_box)[-1])))


    # Take only the first entry of each box corresponding to the probability that there's an object
    pred_obj = predictions[...,0:1]
    true_obj = y_true[...,0:1]

    #Calculate object loss as in the paper
    object_loss = mse(tf.reshape(exists_box*pred_obj, (-1, )), tf.reshape(exists_box*true_obj, (-1, )) )


    # Calculate the loss for cells that don't have objects
    non_exists_box = 1 - exists_box
    no_object_loss = mse(tf.reshape(non_exists_box*pred_obj, (-1, )),
tf.reshape(non_exists_box*true_obj, (-1, )))


    # Penalize more the box loss and less the no object loss
    total_loss = 5*box_loss + object_loss + 0.5*no_object_loss
    return total_loss

tf.keras.backend.clear_session()

yolo = Sequential()
```

```
yolo.add(tf.keras.layers.Conv2D(32, (3, 3), padding="same", input_shape=(448,448,3), use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides = (2,2)))

yolo.add(tf.keras.layers.Conv2D(64, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides = (2,2)))

yolo.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(64, (1, 1), padding="valid", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(128, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides = (2,2)))

yolo.add(tf.keras.layers.Conv2D(256, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(128, (1, 1), padding="valid", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
```

```
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(256, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides = (2,2)))

yolo.add(tf.keras.layers.Conv2D(1024, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(512, (1, 1), padding="valid", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.Conv2D(1024, (3, 3), padding="same", use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))
yolo.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides = (2,2)))

yolo.add(tf.keras.layers.Conv2D(1024, (3, 3), padding="same", strides = (2,2), use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))

yolo.add(tf.keras.layers.Conv2D(512, (1, 1), padding="valid", strides = (2,2), use_bias = False))
yolo.add(tf.keras.layers.BatchNormalization()) #Batch normalization needs to be executed before lrelu apparently
yolo.add(LeakyReLU(alpha=0.1))

yolo.add(tf.keras.layers.Reshape((8192,), input_shape=(4,4,512)))

yolo.add(Dense(245, activation = 'sigmoid'))
```

```
yolo.add(tf.keras.layers.Reshape((7,7,5), input_shape=(245,)))

yolo.summary()

from tensorflow.keras.optimizers.legacy import Adam

training_generator = DataGenerator(img_paths, bboxes)
checkpoint_filepath = 'C:/Users/Asus/Univercity 4 lvl/Diploma/YOLOv1/yolo_best.h5'
adam = Adam(learning_rate=0.5e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath, save_weights_only=True)

yolo.compile(loss=custom_loss, optimizer=adam)

history = yolo.fit(x = training_generator, epochs=7, workers = 1, verbose=1, callbacks=[model_checkpoint_callback])

from scipy.io import savemat

#Save the matrices for the plots
mdic = {"loss": history.history['loss'] }
savemat(r'C:\Users\Asus\Univercity 4 lvl\Diploma\YOLOv1\loss_biggest_model.mat', mdic)

yolo.load_weights('C:/Users/Asus/Univercity 4 lvl/Diploma/YOLOv1/yolo_best_4k.h5')

def decode_net_out(out,confidence=0.6):
    # IN : out [7x7x5]
    # RETURNS : bboxes [list] of the best training boxes
    cell_size = 448/7
    bboxes = []
    confidences = []
    for i in range(0,7):
        for j in range(0,7):
```

```
            if out[i][j][0] > confidence: #CONFIDENCE THRESHOLD (P of the cell)
                bbox = np.zeros((4))
                ox_cell = out[i][j][1]
                oy_cell = out[i][j][2]
                w_cell = out[i][j][3]
                h_cell = out[i][j][4]
                if(ox_cell != 0 and oy_cell !=0):
                    ox = int(ox_cell*cell_size + cell_size*j)
                    oy = int(oy_cell*cell_size + cell_size*i)
                    w = w_cell*448
                    h = h_cell*448
                    lx = ox - w/2
                    ly = oy - h/2
                    bbox = [out[i][j][0],int(lx),int(ly),int(w),int(h)]
                    bboxes.append(bbox)
    return bboxes

def calc_iou(prop_bbox, ground_truth):
    #This function takes as input a single prop_bbox and a single ground_truth bbox and return their intersection
    #over union

    P,p_x, p_y, p_w, p_h = prop_bbox
    g_x, g_y, g_w, g_h = ground_truth

    #Non overlapping cases then IoU = 0.0:
    #-Bottom_right smaller than top left
    if((p_x + p_w) < g_x):
        return 0.0
    if((p_y + p_h) < g_y):
        return 0.0
    if((g_x + g_w) < p_x):
        return 0.0
    if((g_y + g_h) < p_y):
```

```
        return 0.0

    #Top left corner of intersection
    x_in = max(p_x, g_x)
    y_in = max(p_y, g_y)
    #Bottom right corner of intersection
    x1_in = min(p_x + p_w, g_x + g_w)
    y1_in = min(p_y + p_h, g_y + g_h)
    w_in = x1_in - x_in
    h_in = y1_in - y_in
    #Area intersection
    area_in = w_in * h_in

    #Area union
    area_un = (p_w * p_h) + (g_w * g_h) - area_in

    #Intersection Over Union
    IoU = area_in / area_un

    return IoU


def single_img_results(prop_bboxes, true_bboxes):
    #This function returns the average IOU for a single image given its proposed bboxes and corresponding
    #ground truth

    if(len(prop_bboxes)== 0):
        return 0.0

    max_Ious = []

    #For each ground truth find the prop bbox that overlaps the most
    for true_bbox in true_bboxes:
        Iou_results = []
```

```
        for prop_bbox in prop_bboxes:
            Iou_results.append(calc_iou(prop_bbox,true_bbox))
        max_Ious.append(max(Iou_results))

    #If there are more ground truth than len of max_Ious fill the list with 0s until their lenght are the same
    if(len(true_bboxes)>len(max_Ious)):
        app = [0.0]*len(len(true_bboxes)-len(max_Ious))
        max_Ious.append(app)

    avg_Iou = sum(max_Ious)/len(max_Ious)
    return(avg_Iou)



def calculate_results():
    X=True
    while X:
        a = input('Input 1 if you want to run modelon test images.\nInput 2 if you want add your own image.\nIf you want to qout print X.\nInput number:')
        if a == 'X':
            X=False
            break
        elif int(a)==1:

            #READING DATA WITH OBJECTS
            test_bboxes_paths_1 = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\1_labels'))
            test_img_paths_1 = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\1_img'))

            #Read all the text files and create a list of list of bounding boxes, one list per image
            test_bboxes_1 = []*len(test_bboxes_paths_1)
            for path in test_bboxes_paths_1:
                bboxes_it = []
                file = open(path, 'r')
                Lines = file.readlines()
```

```
    for line in Lines:
        bboxes_it.append([int(x) for x in line.split()]) #Separated by white spaces
    test_bboxes_1.append(bboxes_it)

#Creation of the scaled down 448x448 version of the ground truth bboxes
test_bboxes_scaled_1 = []
for i in range(0,len(test_img_paths_1)):
    bboxes_it = []
    original_img = load_img(test_img_paths_1[i])
    width, height = original_img.size
    scale_w = 448 / width
    scale_h = 448 / height

    for box in test_bboxes_1[i]:
        xleft = int(box[0] * scale_w)
        yleft = int(box[1] * scale_h)
        b_width = int(box[2] * scale_w)
        b_height = int(box[3] * scale_h)
        new_bbox = [xleft,yleft,b_width,b_height]
        bboxes_it.append(new_bbox)
    test_bboxes_scaled_1.append(bboxes_it)

 #Predicting


for confidence in [0.52]:

    true_true = 0
    false_true = 0
    avg_Ious = []
    for i in range(0, len(test_img_paths_1)):
        loaded_img = load_img(test_img_paths_1[i], target_size=(448, 448))
        image = img_to_array(loaded_img)
        # scale pixel values to [0, 1]
```

```
image = image.astype('float32')
image /= 255.0
in_net = np.expand_dims(image, axis=0) #Expand dimension because the network works with batches as input

out_net = yolo.predict(in_net)

prop_bboxes = decode_net_out(out_net[0],confidence)
if len(prop_bboxes)>0:
   true_true +=1
elif len(prop_bboxes)==0:
   false_true += 1
#print(prop_bboxes)
avg_Iou = single_img_results(prop_bboxes, test_bboxes_scaled_1[i])
print("Image average IOU : " + str(avg_Iou))
avg_Ious.append(avg_Iou)
#print(true_true,false_true,confidence)


#DRAW THE IMAGE
image = cv2.imread(test_img_paths_1[i])
width,height,channels = image.shape
scale_w = 448/width
scale_h = 448/height
image_show = cv2.resize(image,(448,448))
# Drawing the regions in the Image
for (P,x, y, w, h) in prop_bboxes:
   cv2.rectangle(image_show, (x, y),
                 (x + w, y + h),
                  (0, 0, 255), 2)
   cv2.putText(image_show, 'Tank: ' + str(P), (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, w/448 + 0.2, (36,255,12), 1)
for (x, y, w, h) in test_bboxes_scaled_1[i]:
   cv2.rectangle(image_show, (x, y),
```

```
                    (x + w, y + h),
                     (0, 255, 0), 2)
            im_show_rgb = cv2.cvtColor(image_show, cv2.COLOR_BGR2RGB)
            window_name = 'image'
            plt.imshow(im_show_rgb)
            plt.show()

        print("TEST DATASET AVERAGE IOU = " + str(sum(avg_Ious)/len(avg_Ious)))


    #------------------------------------------------------------------------


    #READING DATA WITHOUT OBJECTS
    test_bboxes_paths_2 = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\2_labels'))
    test_img_paths_2 = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\2_img'))
    #Read all the text files and create a list of list of bounding boxes, one list per image
    test_bboxes_2 = []*len(test_bboxes_paths_2)
    for path in test_bboxes_paths_2:
        bboxes_it = []
        file = open(path, 'r')
        Lines = file.readlines()
        for line in Lines:
            bboxes_it.append([int(x) for x in line.split()]) #Separated by white spaces
        test_bboxes_2.append(bboxes_it)

    #Creation of the scaled down 448x448 version of the ground truth bboxes
    test_bboxes_scaled_2 = []
    for i in range(0,len(test_img_paths_2)):
        bboxes_it = []
        original_img = load_img(test_img_paths_2[i])
        width, height = original_img.size
        scale_w = 448 / width
```

```
        scale_h = 448 / height

        for box in test_bboxes_2[i]:
            xleft = int(box[0] * scale_w)
            yleft = int(box[1] * scale_h)
            b_width = int(box[2] * scale_w)
            b_height = int(box[3] * scale_h)
            new_bbox = [xleft,yleft,b_width,b_height]
            bboxes_it.append(new_bbox)
        test_bboxes_scaled_2.append(bboxes_it)




    print('--------------------------------------------------------------------------------')
    for confidence in [0.52]:
        false_false = 0
        true_false = 0
        avg_Ious = []
        for i in range(0, len(test_img_paths_2)):
            loaded_img = load_img(test_img_paths_2[i], target_size=(448, 448))
            image = img_to_array(loaded_img)
            # scale pixel values to [0, 1]
            image = image.astype('float32')
            image /= 255.0
            in_net = np.expand_dims(image, axis=0) #Expand dimension because the network works with batches as input

            out_net = yolo.predict(in_net)

            prop_bboxes = decode_net_out(out_net[0],confidence)
            if len(prop_bboxes)>0:
                false_false +=1
            elif len(prop_bboxes)==0:
```

```
                true_false += 1
            #print(prop_bboxes)
            avg_Iou = single_img_results(prop_bboxes, test_bboxes_scaled_2[i])
            print("Image average IOU : " + str(avg_Iou))
            avg_Ious.append(avg_Iou)
            #print(false_false,true_false,confidence)

            #DRAW THE IMAGE
            image = cv2.imread(test_img_paths_2[i])
            width,height,channels = image.shape
            scale_w = 448/width
            scale_h = 448/height
            image_show = cv2.resize(image,(448,448))
            # Drawing the regions in the Image
            for (P,x, y, w, h) in prop_bboxes:
                cv2.rectangle(image_show, (x, y),
                              (x + w, y + h),
                               (0, 0, 255), 2)
                cv2.putText(image_show, 'Tank: ' + str(P), (x, y-10), cv2.FONT_HERSHEY_SIMPLEX,
w/448 + 0.2, (36,255,12), 1)
            for (x, y, w, h) in test_bboxes_scaled_2[i]:
                cv2.rectangle(image_show, (x, y),
                              (x + w, y + h),
                               (0, 255, 0), 2)
            im_show_rgb = cv2.cvtColor(image_show, cv2.COLOR_BGR2RGB)
            window_name = 'image'
            plt.imshow(im_show_rgb)
            plt.show()
        print("TEST DATASET AVERAGE IOU = " + str(sum(avg_Ious)/len(avg_Ious)))
    elif int(a)==2:
        b = str(input('Specify path to your file with image:'))

        test_bboxes_paths_1 = natsorted(listdir_fullpath(r'C:\Users\Asus\Univercity 4 lvl\Diploma\1'))
        test_img_paths_1 = natsorted(listdir_fullpath(r'{}'.format(b)))
```

```
#Read all the text files and create a list of list of bounding boxes, one list per image
test_bboxes_1 = []*len(test_bboxes_paths_1)
for path in test_bboxes_paths_1:
    bboxes_it = []
    file = open(path, 'r')
    Lines = file.readlines()
    for line in Lines:
        bboxes_it.append([int(x) for x in line.split()]) #Separated by white spaces
    test_bboxes_1.append(bboxes_it)

#Creation of the scaled down 448x448 version of the ground truth bboxes
test_bboxes_scaled_1 = []
for i in range(0,len(test_img_paths_1)):
    bboxes_it = []
    original_img = load_img(test_img_paths_1[i])
    width, height = original_img.size
    scale_w = 448 / width
    scale_h = 448 / height

    for box in test_bboxes_1[i]:
        xleft = int(box[0] * scale_w)
        yleft = int(box[1] * scale_h)
        b_width = int(box[2] * scale_w)
        b_height = int(box[3] * scale_h)
        new_bbox = [xleft,yleft,b_width,b_height]
        bboxes_it.append(new_bbox)
    test_bboxes_scaled_1.append(bboxes_it)

 #Predicting


for confidence in [0.52]:
```

```
true_true = 0
false_true = 0
avg_Ious = []
for i in range(0, len(test_img_paths_1)):
    loaded_img = load_img(test_img_paths_1[i], target_size=(448, 448))
    image = img_to_array(loaded_img)
    # scale pixel values to [0, 1]
    image = image.astype('float32')
    image /= 255.0
    in_net = np.expand_dims(image, axis=0) #Expand dimension because the network works with batches as input

    out_net = yolo.predict(in_net)

    prop_bboxes = decode_net_out(out_net[0],confidence)
    if len(prop_bboxes)>0:
        true_true +=1
    elif len(prop_bboxes)==0:
        false_true += 1
    #print(prop_bboxes)
    avg_Iou = single_img_results(prop_bboxes, test_bboxes_scaled_1[i])
    print("Image average IOU : " + str(avg_Iou))
    avg_Ious.append(avg_Iou)
    #print(true_true,false_true,confidence)


    #DRAW THE IMAGE
    image = cv2.imread(test_img_paths_1[i])
    width,height,channels = image.shape
    scale_w = 448/width
    scale_h = 448/height
    image_show = cv2.resize(image,(448,448))
    # Drawing the regions in the Image
    for (P,x, y, w, h) in prop_bboxes:
```

```
                cv2.rectangle(image_show, (x, y),
                              (x + w, y + h),
                               (0, 0, 255), 2)
                cv2.putText(image_show, 'Tank: ' + str(P), (x, y-10), cv2.FONT_HERSHEY_SIMPLEX,
w/448 + 0.2, (36,255,12), 1)
            for (x, y, w, h) in test_bboxes_scaled_1[i]:
                cv2.rectangle(image_show, (x, y),
                              (x + w, y + h),
                               (0, 255, 0), 2)
            im_show_rgb = cv2.cvtColor(image_show, cv2.COLOR_BGR2RGB)
            window_name = 'image'
            plt.imshow(im_show_rgb)
            plt.show()

        print("TEST DATASET AVERAGE IOU = " + str(sum(avg_Ious)/len(avg_Ious)))

    else:
        print('Invalid input:')
```

Додаток Б

Ілюстративний матеріал

Рисунок Б.1 – Слайд 1

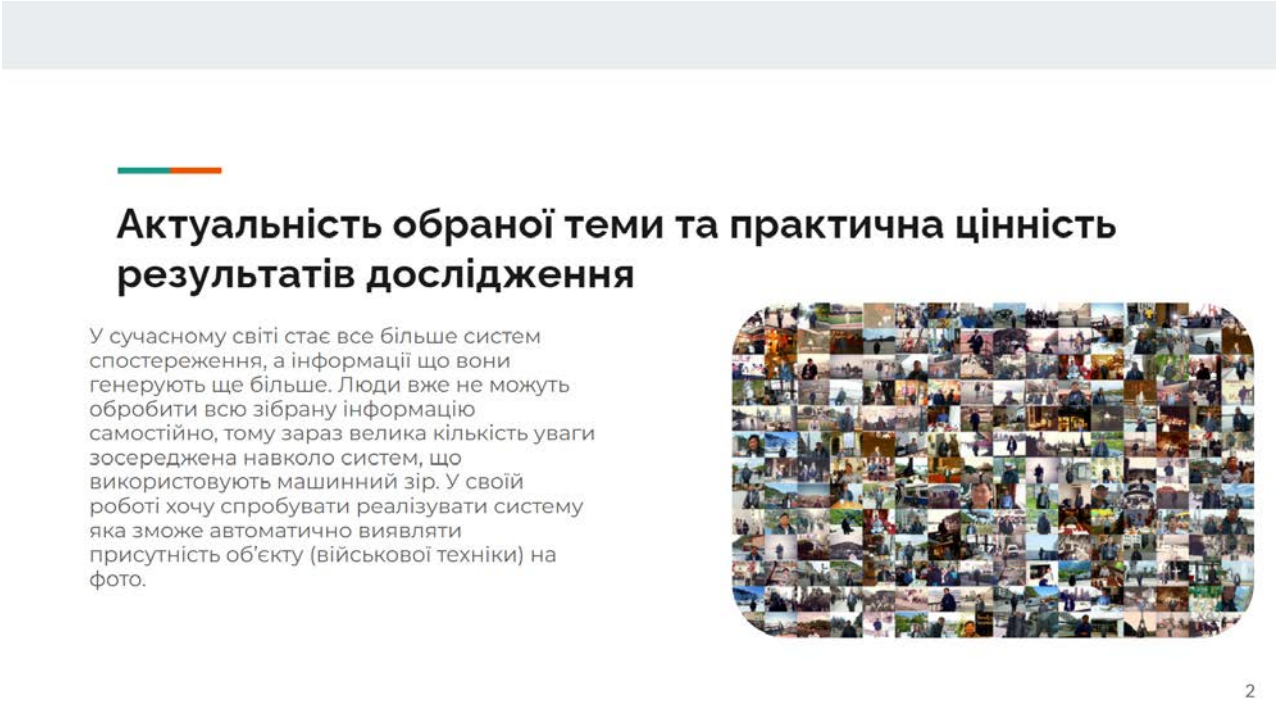


Рисунок Б.2 – Слайд 2

Постановка задачі

Об'єкт дослідження: методи виявлення об'єктів (військової техніки) на зображеннях, методи роботи з візуальними даними, методи машинного навчання для виявлення об'єктів (техніки).

Предмет дослідження: модель системи виявлення об'єктів (техніки) на зображенні на основі машинного навчання, інструменти та методи для забезпечення функціонування моделі.

Мета дипломної роботи полягає в тому, щоб дослідити існуючі рішення для виявлення об'єктів (техніки) на фото, проаналізувати обрані методи для вирішення проблеми, розробити конкурентоспроможну систему виявлення об'єктів на фото.

3

Рисунок Б.3 – Слайд 3

Огляд існуючих рішень

Глибоке навчання:
Згорткові нейронні мережі (Convolutional Neural Networks - CNN) та похідні (R-CNN, Fast CNN, YOLO).

Плюси:
- Точність
- Універсальність
- Швидкість

Мінуси:
- Велика кількість даних
- Кількість обчислювальних ресурсів (за виключенням YOLO)

Машинне навчання:
Гістограма орієнтованих градієнтів (Histogram of Oriented Gradients - HOG);

Метод Віоли-Джонса (Viola–Jones object detection);

Метод масштабно-інваріантної трансформації ознак (Scale-invariant Feature Transform - SIFT)

Плюси:
- Мала кількість даних
- Точність у вузьких сфера
- Швидкість

Мінуси:
- Низька точність

4

Рисунок Б.4 – Слайд 4

Модель системи виявлення техніки на фото

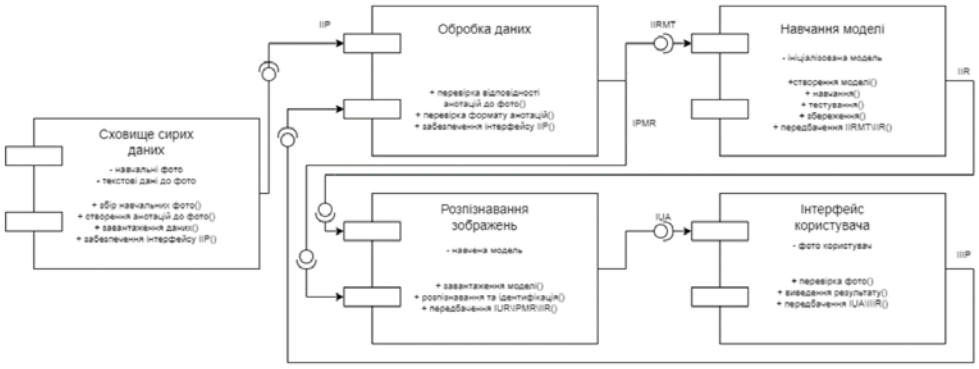


Рис. 1. Діаграма компонентів у нотації UML

5

Рисунок Б.5 – Слайд 5

Вхідні дані

На вхід модель отримує зображення та файл анотацій у якому знаходиться інформація про обмежуючу рамку в якій знаходиться об'єкт.

6

Рисунок Б.6 – Слайд 6

Опис архітектури програмного продукту, що проектується

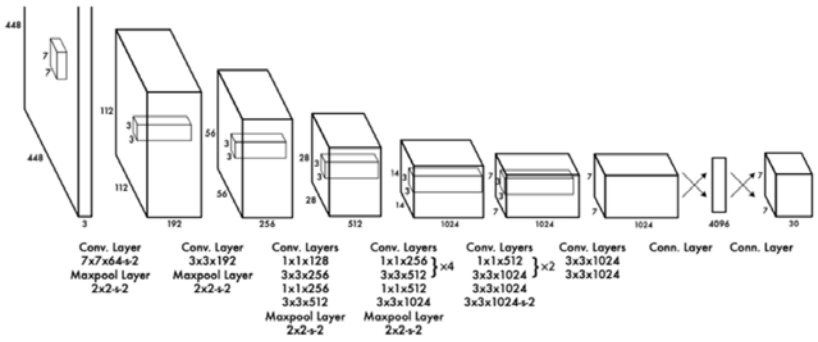


Рис. 2. Архітектура згорткової нейронної мережі YOLOv1 [1]

7

Рисунок Б.7 – Слайд 7

Обчислення помилки моделі

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Рис. 3. Формула обчислення помилки моделі [1]

8

Рисунок Б.8 – Слайд 8

Модифікація алгоритму

На вході були змінені розміри початково ковзаючого вікна для збільшення кількості ознак що будуть захоплюватись моделлю.

Для покращення швидкості навчання було додано нормалізацію вагових коефіцієнтів на кожному шарі згортки.

Функція активації була змінена з ReLU на LeakedReLU для уникнення повних затухань активації нейронів. Це допомогло підвищити реакцію моделі на помилкові результати.

Були змінені розміри вихідних тензорів відповідно до вимоги задачі. Оскільки ми маємо лише 1 клас об'єктів 1 обмежувальну рамку вихід нашої нейронної моделі виглядає наступним чином [p,x,y,w,h].

9

9

Рисунок Б.9 – Слайд 9

Метрика оцінки якості моделі

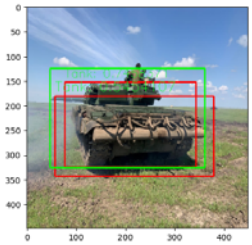
Матриця невідповідностей — це таблиця, що часто використовується для опису ефективності моделі класифікації (або "класифікатора") для набору тестових даних, для яких відомі справжні значення.

Accuracy (точність) - здатність алгоритму відрізняти потрібний об'єкт

Precision (влучність) - здатність алгоритму виявляти потрібний об'єкт узагалі на фото

Intersection over Union (IoU) використовується для оцінки ефективності виявлення об'єктів шляхом порівняння обмежувальної рамки правдивості землі з прогнозованою обмежувальною рамкою

N = 679	True	False
True	307	76
False	159	157



10

Рисунок Б.10 – Слайд 10

Програмна реалізація YOLOv1

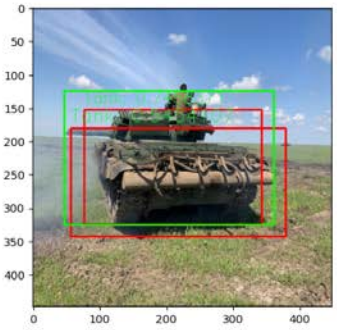
Бібліотеки Python:

Keras, NumPy, MatPlotLib, Pandas, CV2, Tensorflow

Навчання виконувалось на 3806 зображеннях з анотаціями.

Валідація виконувалось на 4481 зображеннях з анотаціями.

Тестування було проведене на 679 зображеннях.



11

Рисунок Б.11 – Слайд 11

Верифікація та валідація

Верифікація. У процесі написання роботи було створено систему виявлення військової техніки на зображенні. Проведено аналіз методів машинного навчання, за допомогою яких розв'язувалась задача виявлення: згорткова нейронна мережа, згорткові, об'єднуючі та повнозв'язні шари нейронної мережі. Найкращим результатом виявлення об'єктів військової техніки на валідаційній вибірці створеною моделлю була точність у 80,7% при рівні впевненості 0.55.

Валідація. Найкращі результати виявлення об'єктів на тестовій вибірці становлять 68.3% при рівні впевненості 0.52. Середнє значення IoU при цьому становить 0.5413. Також слід відзначити що при рівні впевненості 0.4 точність методу менша і становить 64.1%, натомість таку модель можна використовувати у випадках коли ми хочемо відсіяти частину зображень де ми точно впевнені що техніка відсутня. В такому випадку кількість True Negative становить від 10 до 20 % в залежності від вибірки. .

Система може бути допоміжним інструментом для військових аналітиків та систем відеоспостереження як інструмент автоматизації частини роботи людини.

12

Рисунок Б.12 – Слайд 12

Аналіз отриманих результатів

Найкращі результати були досягнуті при рівні впевненості моделі 0.52.

Accuracy (точність) - 68.3%

Precision (влучність) - 84.6%

Intersection over Union - 0.5413746095489592

N = 679	True	False
True	307	76
False	159	157

Матриця невідповідностей

13

Рисунок Б.13 – Слайд 13

Висновки

Розроблено архітектуру нейронної мережі для виявлення військової техніки на фото.

Реалізовано програмне забезпечення системи виявлення військової техніки на фото. Описано програмне забезпечення використаних методів машинного навчання розв'язання задачі, а саме: згорткової нейронної мережі та її компонентів, згорткового шару, об'єднуючого шару, повнозв'язного шару, функції активації та функції обчислення помилки моделі.

Проведено верифікацію та валідацію системи. Результати показали, що найкращою точністю моделі було точність у 68.3%. Система розроблена для використання в автоматизованих системах відеоспостереження і здатна автоматизувати частину роботи.

Перспективи подальших досліджень: формування більшого датасету зображень техніки, яка застосовуються в Україні, розширення системи математичними інструментами ідентифікації об'єктів військової техніки на фото.

14

Рисунок Б.14 – Слайд 14

Перелік посилань

1. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

15

Рисунок Б.15 – Слайд 15

Дякую за увагу!

16

Рисунок Б.16 – Слайд 16