НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

ДО ЗАХИСТУ ДОПУЩЕНО
Завідувач кафедри
_____ Олена ЧУМАЧЕНКО
«___»_____2023 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Виявлення пошкоджених будівель на супутникових зображеннях»

Виконав:
студент IV курсу, групи КІ-91
Бірук Сергій Володимирович _____

Керівник:
Доцент, к.ф.-м.н., доцент
Барановська Л.В. _____

Консультант з економічного розділу:
Доцент, к.е.н.,
Рощина Н.В. _____

Консультант з нормоконтролю:
Фахівець першої категорії кафедри ШІ
Гончарук М.М. _____

Рецензент:
Доцент, к.т.н.,
Харченко К.В. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
завідувач кафедри
_____ Олена ЧУМАЧЕНКО
«___» _____ 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Біруку Сергію Володимировичу

1. Тема роботи «Виявлення пошкоджених будівель на супутникових зображеннях», керівник роботи Барановська Леся Валеріївна, затверджені наказом по університеті від «30» травня 2023 р. №2065-с.

2. Термін подання студентом роботи 08.06.2023

3. Вихідні дані до роботи: посилання будь-яких сайтів.

4. Зміст роботи: дослідження предметної області, методи штучного інтелекту для виявлення пошкоджених будівель на супутникових зображеннях. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В,		

7. Дата видачі завдання: _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи.	13.03.2023 – 23.04.2023	Виконано
2.	Підготовка першого розділу.	24.04 2023 – 30.04.2023	Виконано
3.	Підготовка другого розділу.	01.05.2023 – 07.05.2023	Виконано
4.	Розробка програмного продукту.	19.04.2023 – 01.06.2023	Виконано
5.	Підготовка третього розділу	25.05.2023 - 01.06.2023	Виконано
6.	Підготовка економічної частини	27.05.2023 - 30.05.2023	Виконано
7.	Оформлення розділів відповідно до правил нормоконтролю.	04.05.2023-08.06.2023	Виконано
8.	Підготовка презентації доповіді.	08.06.2023-09.06.2023	Виконано
9.	Оформлення дипломної роботи.	09.06.2023	Виконано

Студент Сергій БІРУК

Керівник Леся БАРАНОВСЬКА

РЕФЕРАТ

Тема: “Виявлення пошкоджених будівель на супутникових зображеннях”

Дипломну роботу виконано на 97 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 25 найменувань. У роботі наведено 27 рисунків та 9 таблиць.

МАШИННЕ НАВЧАННЯ, ГЛИБИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, ВИЯВЛЕННЯ ОБ’ЄКТІВ

Актуальність теми полягає в тому, що пошкоджені будинки можуть служити індикаторами загального впливу природних або техногенних катастроф на інфраструктуру. Аналіз супутникових зображень дають уявлення про розуміння масштабів шкоди та забезпечення цілеспрямованої відповіді, що зрештою сприяє захисту та необхідної підтримки для цивільного населення у зонах катастрофи.

Метою дослідження є переглянути існуючу літературу з аналізу супутникових зображень для оцінки пошкоджень будівель, алгоритмів виявлення об’єктів і згорткові моделі для класифікації зображень руйнувань будинків; по-друге, розробити метод виявлення пошкоджень будівель на супутникових зображеннях за допомогою згоркових мереж, який включає алгоритм виявлення об’єктів та класифікатор; і по-третє, оцінити ефективність запропонованого методу на наборі даних супутникових зображень, що містять будівлі з різним ступенем пошкодження.

Об’єктом дослідження є супутникові знімки та будинки на цих зображеннях.

Предмет дослідження – різні моделі виявлення об’єктів та класифікації зображень.

ABSTRACT

The thesis is completed on 97 sheets, it contains 4 appendices and a list of references to the used sources from 25 names. There are 27 figures in the work and 9 tables.

MACHINE LEARNING, DEEP LEARNING, CLASSIFICATION, OBJECT DETECTION

The relevance of the topic is that damaged houses can serve as indicators of the general impact of natural or man-made disasters on the infrastructure. Analysis of satellite imagery provides insight into understanding the extent of damage and enabling a targeted response, ultimately contributing to the protection and necessary support for civilians in disaster areas.

The aim of the study is to review the existing literature on satellite image analysis for building damage assessment, object detection algorithms and convolutional models for classification of building damage images; secondly, to develop a method for detecting building damage on satellite images using convolutional networks, which includes an object detection algorithm and a classifier; and third, to evaluate the effectiveness of the proposed method on a dataset of satellite images containing buildings with different degrees of damage.

The object of research is satellite images and the houses on these images.

Various models of object detection and image classification became the subject of research.

Зміст

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ШІ – штучний інтелект

CNN (convolutional neural network) – згорткові нейронні мережі

YOLO - You Only Look Once

NMS (Non Maximum Suppression) - немаксимальне придушення

IoU - Intersection over Union

ВСТУП

Будинки — це критично важлива інфраструктура, необхідна для функціонування будь-якого сучасного суспільства. Вони є основою наших міст і служать наріжним каменем людської цивілізації. Однак будівлі вразливі до широкого спектру загроз, як природних, так і створених людиною. Стихійні лиха, такі як землетруси, урагани та повені, створені людиною загрози, такі як війни, тероризм і вандалізм, можуть завдати значної шкоди будівлям, що призведе до проблем безпеки, величезних економічних втрат і людських страждань.

В останні роки використання супутникових зображень стало важливим інструментом для моніторингу навколишнього середовища на планеті Земля. За допомогою зображень з космосу ідентифікують місця великих пожеж [1] чи утворення ураганів, для попередження про можливі загрози. Але навіть для аналізу таких типів стихійних лих використовують ШІ, що дозволяє позбавити трудомісткого процесу переглядати всі знімки вручну.

Метою будь-який попереджень стихійних чи техногенних лих є збереження якнайбільше життів. Проте деякі катастрофи неможливо передбачити, такі як: землетруси, ракетні чи артилерійські прильоти на території військових дій. Єдиним варіантом проаналізувати весь масштаб катастрофи - це прослідкувати за пошкодженою інфраструктурою. Можливість спостерігати за наслідками таких катастроф дають спосіб проаналізувати оцінки збитку та визначення найкращого курсу дій для відновлення будинків. Однак аналіз цих зображень вручну є трудомістким і складним завданням. Ось тут і вступає в дію використання нейронних мереж.

Основною метою цієї дипломної роботи є розробка методології побудови виявлення пошкоджень за допомогою супутникових зображень і згорткових нейронних мереж. Методологія включає два основних етапи: пошук будинків за допомогою моделі виявлення об’єктів для локалізації будівель та класифікація пошкоджень. Алгоритм YOLO використовуватиметься для ідентифікації будівель на

супутникових зображеннях, а Siamense model – для класифікації пошкоджень цих будівель.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мотивація дослідження

У розпал військових дій чи землетрусу ідентифікація та оцінка пошкоджених будинків на території набувають першочергового значення. Аналіз супутникових зображень є цінним інструментом для вирішення цього завдання, дозволяючи збройним силам і гуманітарним організаціям отримувати важливу інформацію щодо масштабу збитку та терміновості необхідного реагування для рятування людей. Розуміння важливості пошуку пошкоджених будинків за супутниковими знімками на території бойових дій вимагає розгляду таких ключових факторів:

По-перше, безпека та благополуччя цивільного населення, які проживають в зонах конфлікту, викликають найбільше занепокоєння. Військові дії завжди призводять до побічних збитків, у тому числі до руйнування будинків. Виявляючи та документуючи пошкоджені будинки за допомогою супутникових зображень дозволяють ідентифікувати райони, які потребують негайної допомоги, сприяти наданню цільової гуманітарної допомоги та підтримки постраждалого населення.

По-друге, пошкоджені будинки можуть служити індикаторами загального впливу конфлікту на цивільну інфраструктуру. Масштаби руйнувань у житлових районах дають уявлення про гостроту бойових дій, що тривають, і рівень захисту, необхідний для цивільного населення. Аналіз супутникових зображень допомагає створити точну картину ситуації на місці, інформуючи військове командування та осіб, які приймають рішення, про ефективність їхніх стратегій та потенційну потребу коригування оперативних планів.

Крім того, виявлення пошкоджених будинків із супутникових знімків допомагає оцінити пропорційність і дотримання міжнародного гуманітарного права під час військових операцій. Аналізуючи масштаби та характер руйнувань, юридичні

експерти та правозахисні організації можуть оцінити збиток, завданий цивільним спорудам. Ця інформація є важливою для притягнення до відповідальності осіб, винних у будь-яких можливих порушеннях, та забезпечення дотримання міжнародних правових стандартів.

Також пошук пошкоджених будинків за допомогою супутникових знімків дозволяє ефективно здійснити постконфліктну реконструкцію та реабілітацію. Зацікавлені сторони, такі як державні органи, агенції з надання допомоги та неурядові організації, визначивши масштаб і розподіл збитку, можуть розробити комплексні плани реконструкції, розподілу ресурсів і відновлення життєво важливих послуг. Цей цілеспрямований підхід максимізує ефективне використання ресурсів, мінімізує ризик подальшої шкоди та полегшує процес відновлення громад, які постраждали від конфлікту.

Крім того, аналіз супутникових зображень сприяє ситуаційній обізнаності військових сил, які діють у зонах конфлікту. Виявляючи пошкоджені будинки, військові командири можуть отримати уявлення про тактику противника та потенційні проблемні зони. Здатність розпізнавати моделі та тенденції руйнування житлових споруд може допомогти зрозуміти поведінку та стратегії протиборчих сил. Ці знання підвищують ефективність військових операцій, та дозволяють приймати більш обґрунтовані рішення, що потенційно зменшують жертви серед цивільного населення.

Отже, пошук пошкоджених будинків за супутниковими знімками на території бойових дій має багато життєво важливих цілей. Він забезпечує безпеку та благополуччя цивільного населення, дає уявлення про вплив конфлікту на інфраструктуру, оцінює відповідність міжнародному гуманітарному праву, сприяє зусиллям із постконфліктної реконструкції та покращує обізнаність збройних сил про ситуацію. Аналіз супутникових зображень є цінним інструментом для розуміння масштабів шкоди та забезпечення цілеспрямованої відповіді, що зрештою сприяє захисту та підтримці постраждалих людей у зонах конфлікту.

1.2. Довідкова інформація щодо виявлення пошкоджень будівель і згорткових нейронних мереж

Згорткові нейронні мережі змінили сферу аналізу зображень, дозволивши машинам «бачити» та інтерпретувати візуальні дані з надзвичайною точністю. Натхненні складною організацією зорової кори людини, ці мережі чудово розпізнають шаблони та особливості в зображеннях.

Концепція згорткових мереж з'явилася після дослідження Хабела та Візеля в 1950-х і 1960-х роках, які показали ієрархічну структуру зорової кори . Вони виявили, що нейрони зорової кори мають рецептивні поля, які вибірково реагують на специфічні зорові стимули [2]. Ці відкриття стали основою для розробки штучних нейронних мереж, які могли б імітувати механізми візуальної обробки мозку.

І в 1980 році Куніхіко Фукусіма, був натхненний роботами Хабеля та Візеля і запропонував модель неокогнітрон, в якій було представлено згорткові шари та шари з пониженою вибіркою як два основні типи шарів у ConvNets. Згорткові шари використовують перекриття рецептивних полів і вагові вектори фільтрів [2]. Що представляло собою найпростішу нейронну мережу для розпізнавання зображень.

З розвитком технологій та збільшенням обчислювальної потужності стало можливим створювати більш складні структури згорткових мереж, що дає можливість обробляти більш складну інформацію та більшу кількість інформації.

Однак, навіть маючи приголомшливі можливості, CNN не позбавлені обмежень і ці алгоритми не безпомилкові. Вони залежать від якості навчальних даних, відповідного вибору архітектурних конфігурацій і ретельного калібрування параметрів моделі.

Ефективність згорткових мереж у виявленні пошкоджень у будівлях полягає в їхній здатності навчатися на величезній кількості позначених даних. Навчання CNN включає в себе показ його безлічі анотованих зображень, що зображують різні типи та ступені пошкодження будівлі. За допомогою процесу, відомого як контрольоване навчання, мережа вчиться розпізнавати та класифікувати конкретні моделі, пов'язані

з різними типами пошкоджень. Цей етап навчання дозволяє мережі розвинути складне розуміння візуальних сигналів, що вказують на пошкодження будівлі, дозволяючи їй узагальнювати свої знання та точно ідентифікувати пошкодження на зображеннях.

Ключовою віхою у виявленні об’єктів за допомогою CNN стала розробка алгоритму You Look Only Once. YOLO представила новаторський підхід до виявлення об’єктів, який зробив революцію в галузі. На відміну від попередніх алгоритмів, які покладалися на трудомісткі методи ковзного вікна, YOLO використовує єдину нейронну мережу для прямого прогнозування обмежувальних рамок і ймовірностей класу в реальному часі. Цей новий підхід значно покращив швидкість і ефективність виявлення об’єктів, що робить його особливо придатним для великомасштабних застосувань, таких як виявлення будівель на супутникових зображеннях. Поєднання супутникових зображень і CNN дозволяє швидко й точно ідентифікувати пошкоджені будівлі, надаючи людям, які приймають рішення, гуманітарним організаціям і службам екстреного реагування, важливу інформацію для планування та виконання цілеспрямованих заходів з надання допомоги.

Важливість цього дослідження полягає в його потенціалі для підвищення точності та швидкості виявлення пошкоджень будівель за допомогою супутникових зображень. Запропонована методологія може бути використана для надання точної та своєчасної інформації про масштаби шкоди, заподіяної стихійними лихами чи техногенними загрозами, дозволяючи органам влади вживати відповідних заходів для пом’якшення шкоди та надання необхідної допомоги постраждалому населенню.

1.3. Цілі та питання дослідження

Ця дипломна присвячена розробці надійного методу виявлення пошкоджень будівель на супутникових зображеннях за допомогою згорткових нейронних мереж. Запропонований підхід складається з двох основних етапів: по-перше, визначення будівель на супутникових зображеннях за допомогою алгоритму виявлення об’єктів, по-друге, класифікація виявлених будівель на основі ступеня пошкодження за допомогою класифікатора на основі Siamese моделі. Мета полягає в тому, щоб створити систему, яка зможе швидко й точно оцінити пошкодження будівель, надаючи безцінну інформацію тим, кому доручено координувати зусилля з реагування на стихійні лиха.

Таким чином, проблема полягає в наступному: як ми можемо розробити метод виявлення пошкоджень будівель на супутникових зображеннях за допомогою CNN, який був би точним і ефективним? Щоб відповісти на це питання, ми вивчимо використання алгоритму виявлення об’єктів для ідентифікації будівель на супутникових зображеннях і дослідимо застосування класифікаторів на основі CNN для оцінки ступеня пошкодження.

Цілі цієї роботи: по-перше, переглянути існуючу літературу з аналізу супутникових зображень для оцінки пошкоджень будівель, алгоритмів виявлення об’єктів і CNN для класифікації зображень; по-друге, розробити метод виявлення пошкоджень будівель на супутникових зображеннях за допомогою методів машинного навчання, який включає алгоритм виявлення об’єктів YOLO та класифікатор на базі Siamese; і по-третє, оцінити ефективність запропонованого методу на наборі даних супутникових зображень, що містять будівлі з різним ступенем пошкодження.

Обсяг цієї дипломної обмежується виявленням і класифікацією пошкоджень будівель на супутникових зображеннях за допомогою машинного навчання. Хоча інші методи дистанційного зондування та алгоритми машинного навчання також можуть бути застосовані до цієї проблеми, це дослідження зосереджено на

використанні CNN та алгоритму виявлення об’єктів YOLO. Крім того, ця робота не розглядатиме ширших питань реагування на катастрофи та відновлення, таких як розподіл ресурсів і логістика, а натомість зосередиться на розробці методу виявлення пошкоджень будівель на супутникових зображеннях.

Дипломна робота організована таким чином: Розділ 2 містить огляд літератури з аналізу супутникових зображень для оцінки пошкоджень будівель, алгоритмів виявлення об’єктів і CNN для класифікації зображень. Розділ 3 представляє аналіз результатів оцінки запропонованого методу. Розділ 4 представляє функціонально-вартісний аналіз програмного продукту.

Зрештою, ця робота має на меті розробити метод, який є водночас точним і ефективним, ми сподіваємося надати цінний інструмент для тих, кому доручено координувати зусилля з реагування на катастрофи, що зрештою допоможе врятувати не одне життя.

1.4. Висновок

Підсумовуючи, мотивація для проведення досліджень щодо виявлення пошкоджень будівель за допомогою супутникових зображень і згорткових нейронних мереж обумовлена гострою потребою посилити реагування на стихійні лиха, допомогти гуманітарним зусиллям і підтримати міське планування після катастрофічних подій. Здатність швидко й точно оцінювати пошкодження будівель дистанційно за допомогою супутникових зображень потенційно може врятувати життя, ефективно розподілити ресурси та прискорити процеси відновлення. У регіонах, які постраждали від військових дій або збройних конфліктів, ідентифікація та класифікація пошкоджених будівель за допомогою супутникових зображень стає ще більш важливою. Військові операції часто призводять до масштабних руйнувань,

через що службам швидкого реагування та гуманітарним організаціям важко оцінити масштаби збитку та визначити пріоритети свого втручання.

Дослідження в цій галузі сприяють розвитку комп’ютерного зору та методів глибокого навчання. Згорткові нейронні мережі зробили революцію в задачах аналізу зображень, продемонструвавши надзвичайні можливості виявлення, класифікації та сегментації об’єктів. Застосовуючи ці методи у сфері виявлення пошкоджень будівель, дослідники розширюють межі того, що можливо у дистанційному зондуванні та оцінці катастроф. Дослідження різних архітектур CNN, таких як сіамські мережі та моделі виявлення об’єктів, такі як YOLO, ще більше розширюють потенціал точного та ефективного виявлення та класифікації пошкоджених будівель.

РОЗДІЛ 2. ОГЛЯД СУЧАСНИХ АРХІТЕКТУР

2.1. План дослідження виявлення пошкоджень будівель з супутникових зображень

Запропонована методологія для виявлення пошкоджень будівлі передбачає двоетапний підхід. По-перше, моделі виявлення використовуються для ідентифікації та локалізації будівель на супутникових зображеннях шляхом створення обмежувальних рамок. Згодом для класифікації пошкоджень використовується згорткові нейронна мережа. Ця комбінована структура спрямована на прискорення аналізу зображень, одночасно забезпечуючи точну та ефективну ідентифікацію та класифікацію пошкоджень будівель.

Початковий крок аналізу передбачає використання моделей виявлення для визначення місцезнаходження будівель на супутникових знімках. Ці моделі, часто засновані на таких алгоритмах виявлення об’єктів, як Faster R-CNN, RetinaNet, YOLO або SSD. Ці алгоритми спеціально розроблені для ідентифікації та локалізації об’єктів, що цікавлять, на зображенні. Використовуючи можливості цих моделей виявлення, запропонована методологія точно визначає обмежувальні рамки навколо будівель, надаючи точну інформацію про локалізацію для подальшого аналізу.

Після фази виявлення будівлі використовуємо згорткові мережі для класифікації пошкоджень. Згорткові мережі відомі своєю майстерністю в задачах аналізу зображень завдяки своїй здатності вивчати та витягувати значущі характеристики з візуальних даних. Використовуючи вивчені представлення в CNN, методологія має на меті класифікувати будівлі як пошкоджені, сильно пошкоджені, помірно пошкоджені або непошкоджені. Цей процес класифікації дає змогу ретельно оцінити ступінь і серйозність пошкодження будівлі.

Інтеграція згорткових мереж для класифікації пошкоджень забезпечує потужну основу для визначення та класифікації пошкоджень будівель. Використовуючи

методи глибокого навчання, CNN можуть автоматично вивчати дискримінаційні особливості, які відрізняють пошкоджені та непошкоджені структури. Це дає змогу методології точно класифікувати будівлі на основі вивчених уявлень, що полегшує прийняття обґрунтованих рішень щодо серйозності та терміновості необхідних втручань.

Крім того, поєднання моделей виявлення об’єктів і згорткових моделей для класифікації підвищує загальну ефективність процесу аналізу зображення. Моделі виявлення об’єктів ефективно ідентифікують і локалізують будівлі, зменшуючи простір пошуку для подальшої класифікації пошкоджень. Цей цілеспрямований підхід дозволяє класифікатору зосередити свій аналіз на відповідних областях інтересу, що спрощує подальший аналіз і зменшує обчислювальні витрати, сприяючи швидшому часу обробки та прискореному аналізу зображень. У результаті модель досягає балансу між швидкістю та точністю, що дозволяє своєчасно та надійно виявляти та класифікувати пошкодження будівель.

2.2. Виявлення об’єктів

Виявлення об’єктів – це техніка комп’ютерного зору, яка працює для ідентифікації та визначення місцезнаходження об’єктів на зображенні чи відео. Зокрема, виявлення об’єктів малює обмежувальні рамки навколо цих виявлених об’єктів, що дозволяє нам визначити, де ці об’єкти знаходяться (або як вони рухаються) у певній сцені [3].

Виявлення об’єктів і класифікація зображень є двома пов’язаними, але різними концепціями комп’ютерного зору. Важливо розуміти відмінності між ними, щоб уникнути плутанини.

Розпізнавання зображень передбачає присвоєння однієї мітки всьому зображенню. Наприклад, якщо на зображенні зображено пташку, система розпізнавання зображення просто позначить її як «bird». Подібним чином, якщо на

зображенні кілька пташок, вони всі будуть позначені як "bird". По суті, розпізнавання зображень зосереджено на класифікації всього зображення на основі його вмісту.

З іншого боку, виявлення об’єктів не лише ідентифікують самі об’єкти, але й надають додаткову інформацію про їх розташування на зображенні. А також, замість того, щоб призначати одну мітку всьому зображенню, моделі виявлення об’єктів малюють рамки навколо кожного окремого об’єкта інтересу, наприклад кожної пташки на зображенні. Потім ці рамки позначаються як "bird". Таким чином, виявлення об’єктів не тільки ідентифікує об’єкти, присутні на зображенні, але й надає просторову інформацію про їх положення.

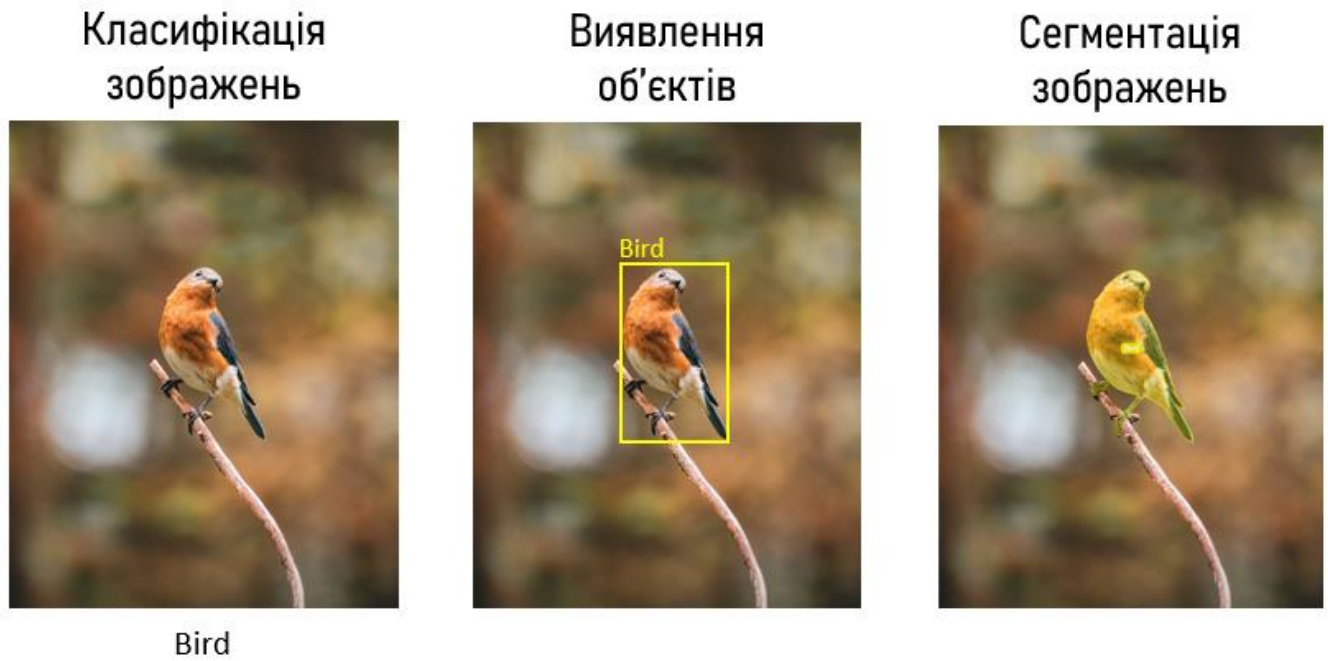

Рис.2.1. Різниця між класифікацією зображень, виявленням об’єктів та сегментацією зображень[4]

У завданні класифікації зображень мережа призначає клас кожному вхідному зображенню. Виявлення об’єктів ідентифікує об’єкт та локалізує його на зображенні. Однак припустімо, що ви хочете знати форму цього об’єкта, який піксель якому об’єкту належить тощо. У цьому випадку потрібно призначити клас кожному пікселю зображення — це завдання відоме як сегментація. Модель сегментації повертає набагато більш детальну інформацію про зображення. Сегментація зображень має багато застосувань у медичних зображеннях, безпілотних автомобілях і супутникових зображеннях[5].

Така особливість моделей виявлення об’єктів та сегментація дозволяє точно ідентифікувати та локалізувати об’єкти на одному зображенні, забезпечуючи широкий спектр застосувань застосувань у медичних зображеннях, безпілотних автомобілях, супутникових зображеннях та систем спостереження.

2.3. Порівняння різних архітектур виявлення об’єктів

Початок в розвитку виявлення об’єктів розпочався із класичних, на цей час, методів машинного навчання. До цих методів належав детектор Віоли-Джонса[6], що в 2001 поклав початок розробці традиційних методів виявлення об’єктів, масштабно-інваріантне перетворення ознак (SIFT) [7], гістограма орієнтованих градієнтів (HOG)[8].

Ці класичні підходи мали на меті ідентифікувати відмінні риси в зображенні, а потім класифікувати їх за допомогою різних алгоритмів машинного навчання, таких як логістична регресія, кольорові гістограми або випадкові ліси [9]. Ці методи працювали шляхом виявлення конкретних візерунків або характеристик, таких як краї, кути або текстури, і використання цих сигналів для виявлення об’єктів.

Однак із появою глибокого навчання в 2013 році [9] досліджуване поле виявлення об’єктів зазнав значних змін. Методи глибокого навчання, зокрема згорткові нейронні мережі, зробили революцію в цій галузі. CNN чудово вміють автоматично вивчати складні ієрархічні представлення з необроблених піксельних даних, що дозволяє їм фіксувати складні шаблони та залежності в зображеннях.

Використання моделей глибокого навчання для виявлення об’єктів призвело до значного підвищення точності та продуктивності. Ці моделі здатні одночасно локалізувати та класифікувати об’єкти на зображеннях, перевершуючи можливості класичних методів машинного навчання. Завдяки використанню великомасштабних анотованих наборів даних і потужності паралельної обробки графічних процесорів

алгоритми виявлення об’єктів на основі глибокого навчання досягли безпрецедентного рівня точності та швидкості.

Сучасні методи виявлення об’єктів можна умовно розділити на дві основні категорії: одноступеневі та двоступеневі детектори об’єктів. Ці методи, які використовують методи глибокого навчання, спрямовані на вилучення відповідних функцій із вхідних зображень або відеокадрів для точного виявлення об’єктів [10].

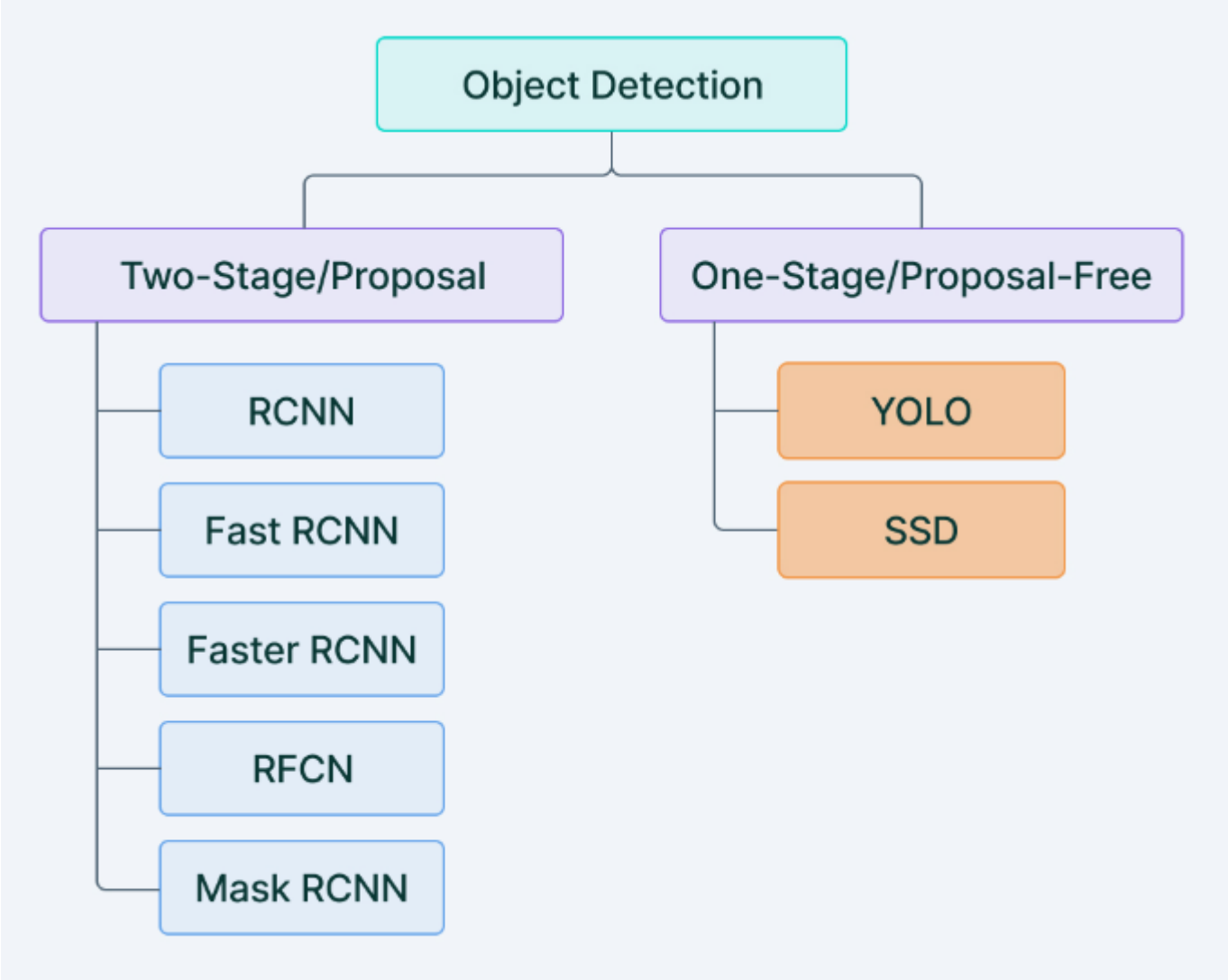

Рис.2.2. Одноступеневі та двоступеневі детектори об’єктів [9].

Загалом детектор об’єктів вирішує два послідовних завдання. Перше завдання передбачає ідентифікацію та локалізацію змінної кількості об’єктів на зображенні, яка може бути навіть нульовою, якщо об’єктів немає. Друге завдання полягає в класифікації кожного виявленого об’єкта та оцінці його точного розташування за допомогою обмежувальних рамок.

Щоб спростити цей процес, деякі методи виявлення об’єктів поділяють ці завдання на окремі етапи. У двоетапному підході перший етап зосереджений на створенні набору потенційних пропозицій об’єктів. Ці пропозиції служать регіонами-кандидатами інтересу, які ймовірно містять об’єкти. Потім другий етап класифікує кожну пропозицію та уточнює координати обмежувальної рамки, забезпечуючи точну локалізацію та класифікацію об’єктів.

Різноманітні двоступінчасті детектори включають region convolutional neural network (RCNN) з еволюціями Faster R-CNN або Mask R-CNN. Останньою розробкою є Granulated RCNN (G-RCNN) [10].

Двоступеневі детектори об’єктів спочатку знаходять область інтересу та використовують цю обрізану область для класифікації. Однак такі багатоступеневі детектори зазвичай не піддаються наскрізному навчанню, оскільки кадрування є недиференційованою операцією [10].

Натомість одноступінчасті детектори безпосередньо передбачають наявність і розташування об’єктів на всьому зображенні. Основні переваги виявлення об’єктів за допомогою одноетапних алгоритмів включають загалом більшу швидкість виявлення та більшу структурну простоту й ефективність порівняно з багатоступеневими детекторами.

Щоб знайти баланс між швидкістю й точністю, дослідники постійно розробляють і вдосконалюють методи виявлення об’єктів. Ці досягнення передбачають удосконалення мережевих архітектур, методів виділення функцій і стратегій навчання, спрямованих на розширення меж продуктивності виявлення об’єктів у різних програмах. Мета даної роботи допрацювати метод класифікації, який виконаний на архітектурі Siamese моделі. Таке доповнення допоможе порівняти два зображення, до та після катастрофи, та виконати класифікацію ступеня пошкодженості. Саме тому для нас актуально використати одноступеневі детектори.

2.3.1. SSD

Single Shot MultiBox Detector (SSD) — це алгоритм комп’ютерного зору, який використовує згорточні нейронні мережі для створення кількох обмежувальних прямокутників із фіксованими розмірами. Потім ці обмежувальні прямокутники оцінюються, щоб визначити наявність у них примірника класу об’єктів. Щоб отримати остаточне виявлення, застосовується немаксимальний крок придушення. Модель SSD працює шляхом поділу кожного вхідного зображення на сітки різного розміру. У кожній сітці виявлення виконується для різних класів об’єктів і пропорцій. Кожній сітці присвоюється оцінка, що вказує на ступінь відповідності між об’єктом і цією конкретною сіткою. Щоб отримати остаточні результати виявлення, використовується немаксимальне придушення для обробки виявлень, що накладаються [11].

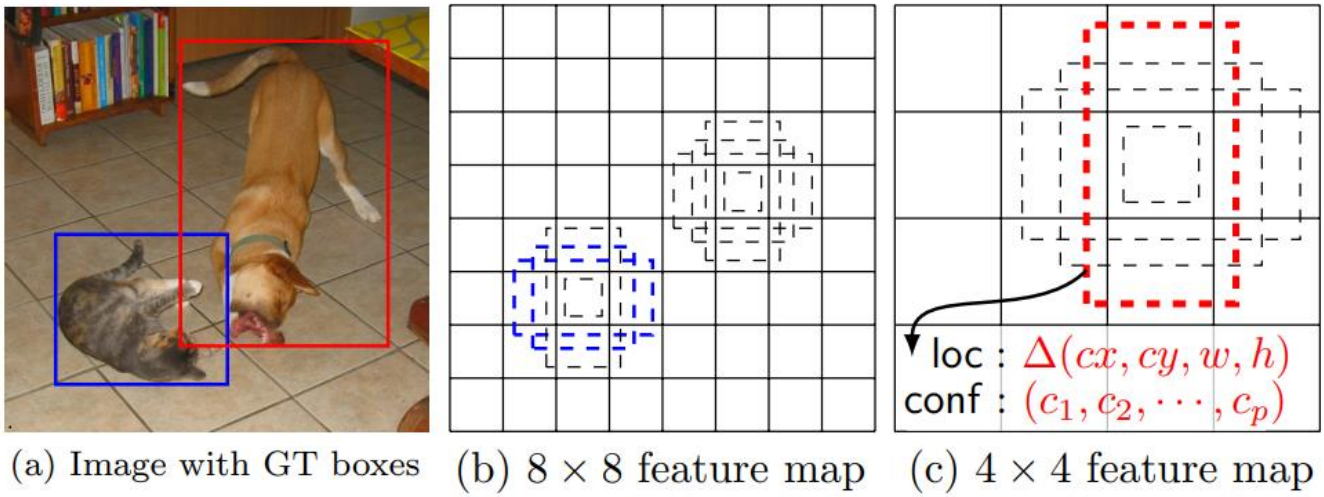

Рис.2.3. SSD framework [12]

Більш детально, модель SSD починається з поділу вхідного зображення на структуру сітки, причому кожна сітка представляє певну область всередині зображення. У кожній сітці мережа SSD генерує набір обмежувальних прямокутників попередньо визначених розмірів і пропорцій, що охоплює діапазон потенційних місць і форм об’єктів. Ці обмежувальні прямокутники служать регіонами-кандидатами для виявлення об’єктів.

Далі мережа SSD обчислює бали для кожної обмежувальної рамки, щоб визначити ймовірність містити об’єкт певного класу. Мережа застосовує згорткові операції та

виділення ознак для оцінки візуальної інформації в межах кожної обмежувальної рамки, виробляючи оцінки достовірності, пов’язані з різними класами об’єктів. Ці бали представляють ймовірність присутності даного класу у відповідній обмежувальній рамці.

Після процесу оцінки застосовується немаксимальне придушення для уточнення набору виявлених об’єктів. Цей крок спрямований на усунення зайвих і перекриваючих обмежувальних рамок для отримання найбільш точних і незайвих виявлень. Немаксимальне придушення передбачає порівняння оцінок достовірності суміжних обмежувальних рамок і видалення тих із нижчими оцінками, коли відбувається значне перекриття. Решта обмежувальних прямокутників із найвищими оцінками достовірності вважаються остаточними виявленнями, наданими моделлю SSD.

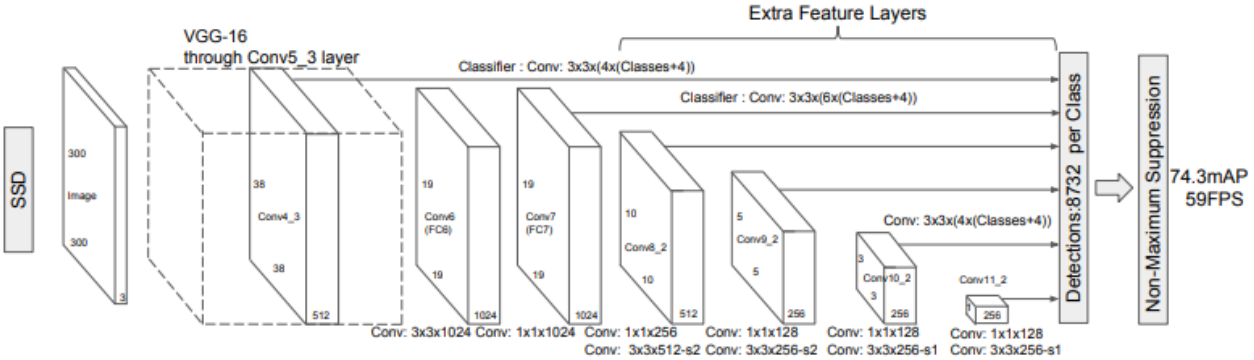

Рис.2.4. Архітектура SSD на базі VGG-16 [12]

Підводячи підсумок, модель SSD використовує згорткові мережі для створення кількох обмежувальних прямокутників із заздалегідь визначеними розмірами та співвідношеннями сторін у структурі на основі сітки. Оцінюючи присутність екземплярів класу об’єктів у цих обмежувальних прямокутниках і застосовуючи немаксимальне придушення, модель SSD забезпечує надійну продуктивність виявлення об’єктів.

2.3.2. YOLO

Підхід You Only Look Once (YOLO) пропонує використання наскрізної нейронної мережі, яка дозволяє одночасно прогнозувати обмежувальні прямокутники та ймовірності класу. Ця парадигма пропонує кілька переваг, зокрема виняткову швидкість. Формулюючи задачу виявлення об'єктів як проблему регресії, YOLO позбавляє потреби в складних pipelines.

Основною характеристикою фреймворку YOLO є його здатність виконувати виявлення об’єктів із надзвичайною ефективністю. На відміну від традиційних методів, які передбачають багатоетапні процеси, що включають локалізацію об’єктів і подальшу класифікацію, YOLO охоплює уніфіковану архітектуру, яка обходить ці проміжні етапи. Цей унікальний підхід забезпечує суттєве прискорення швидкості виявлення.

Щоб досягти виявлення об’єктів у реальному часі, YOLO формулює завдання як задачу регресії, де координати обмежувальної рамки та пов’язані ймовірності класу прогнозуються одночасно. Нейронна мережа YOLO обробляє все зображення за один прохід, безпосередньо виробляючи прогнози обмежувальної рамки та їхні ймовірності відповідного класу. Цей уніфікований процес прогнозування усуває потребу в кількох ітераціях і обчислювально інтенсивних операціях, що робить YOLO особливо добре підходячим для програм, які вимагають швидкого та точного виявлення об’єктів.

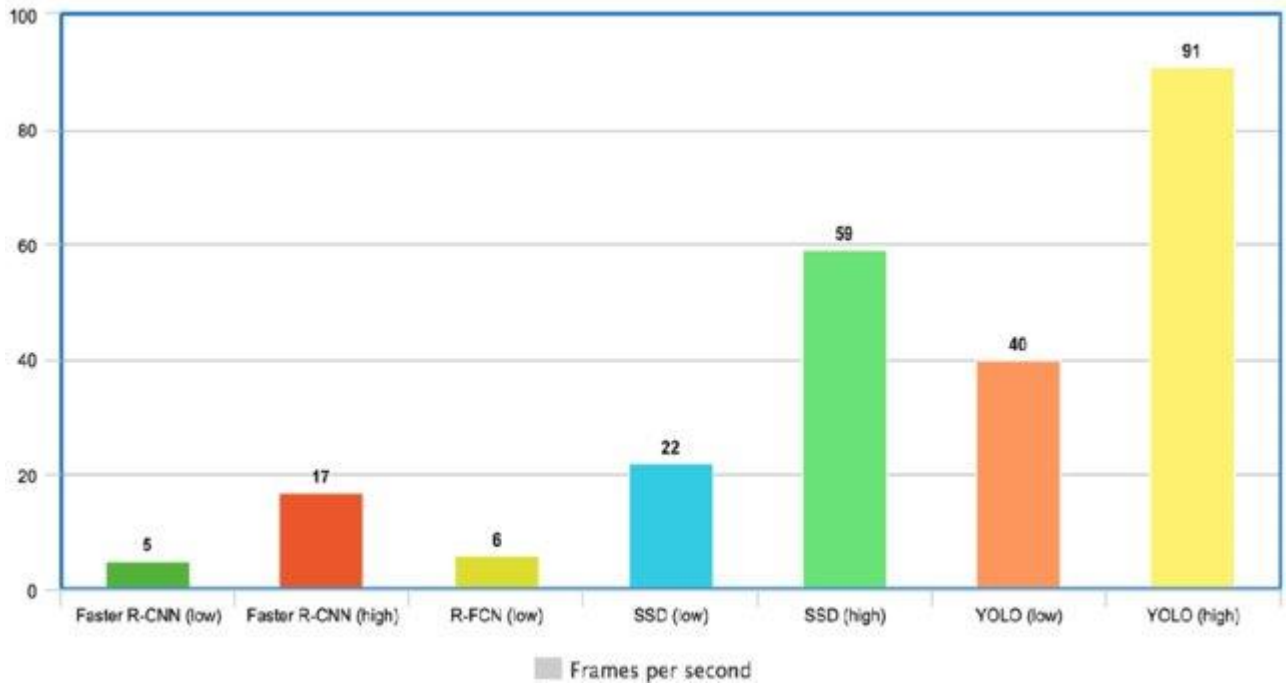

Рис.2.5. Порівняння кількості оброблених кадрів за секунду (FPS) при реалізації моделей Faster R-CNN, R-FCN, SSD та YOLO з використанням вхідних зображень з різною роздільною здатністю [13].

Алгоритм YOLO приймає зображення як вхідні дані, а потім використовує просту глибоку згорткову нейронну мережу для виявлення об’єктів на зображенні. Архітектура моделі CNN, яка є основою YOLO, показана нижче.

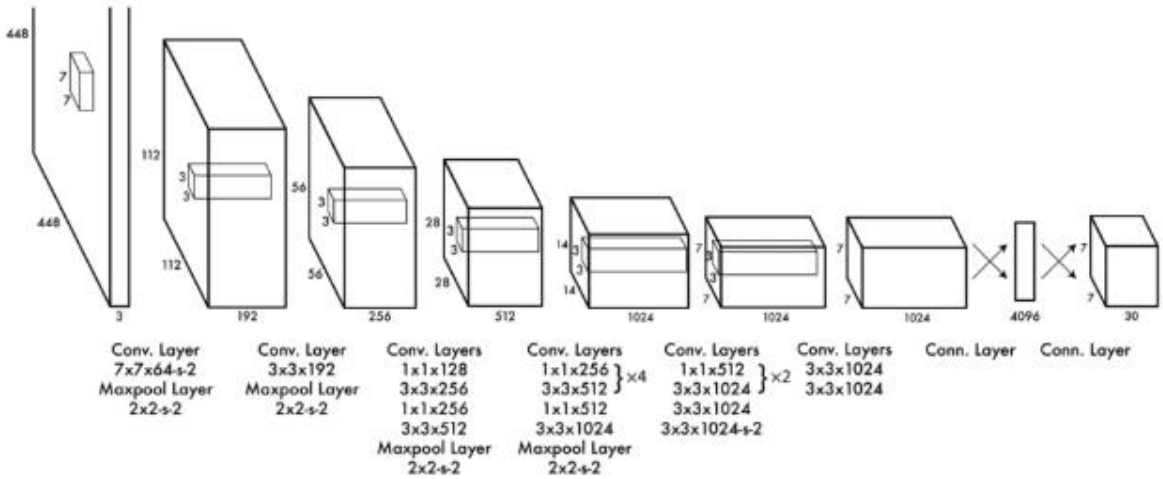

Рис.2.6. Архітектура YOLO[14]

Перші 20 шарів згортки моделі попередньо навчені за допомогою ImageNet шляхом підключення тимчасового середнього пулу та повністю підключеного шару. Потім ця попередньо навчена модель перетворюється для виконання виявлення, оскільки попередні дослідження показали, що додавання згортки та підключених шарів до попередньо навченої мережі підвищує продуктивність. Останній повністю пов’язаний рівень(fully convolution layer) YOLO передбачає як імовірності класу, так і координати обмежувальної рамки. [14]

YOLO ділить вхідне зображення на сітку S × S (рис.2.7). Клітинка, що аналізується, відповідає за виявлення цього об’єкта, якщо центр об’єкта потрапляє в цю клітинку сітки. Кожна клітинка сітки передбачає обмежувальні прямокутники B і оцінки достовірності для цих прямокутників.

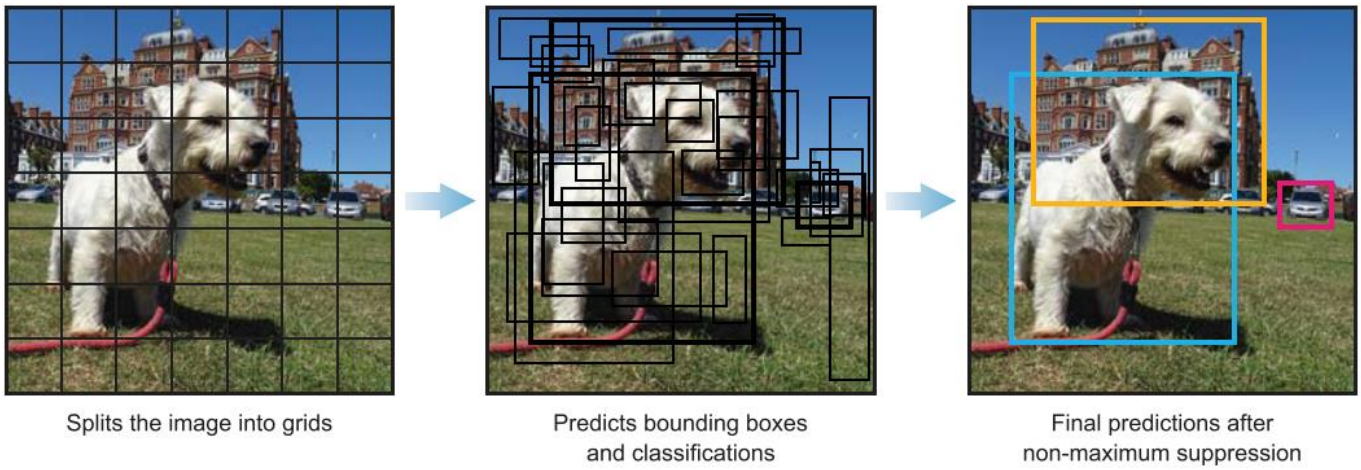

Рис. 2.7. Підхід YOLO до виявлення об’єктів [20]

Координати обмежувальних рамок B, подібно до усіх детекторів, YOLO передбачає чотири координати для кожної обмежувальної рамки (b_x, b_y, b_w, b_h), де x і y встановлено як зміщення розташування комірки.

Оцінка об’єктності — вказує ймовірність того, що клітинка містить об’єкт. Оцінка об’єктності передається через сигмоїдну функцію, яка розглядається як ймовірність із діапазоном значень від 0 до 1 [20]. Ці оцінки достовірності відображають, наскільки модель впевнена в тому, що бокс містить об’єкт, і наскільки точну, на її думку, передбачений бокс. Оцінка об’єктності обчислюється таким чином:

$$\mathrm{P}_0 = P_r(containing\ an\ object) \times IoU(pred, truth) \qquad (2.1)$$

Одним із ключових методів, що використовується в моделях YOLO, є немаксимальне придушення. NMS — це етап постобробки, який використовується для підвищення точності та ефективності виявлення об’єктів. Під час виявлення об’єктів зазвичай для одного об’єкта на зображенні генеруються кілька обмежувальних рамок. Ці обмежувальні рамки можуть перекриватися або розташовуватися в різних положеннях, але всі вони представляють той самий об’єкт. NMS використовується для ідентифікації та видалення зайвих або неправильних обмежувальних рамок і для виведення окремої обмежувальної рамки для кожного об’єкта на зображенні.

З моменту першого випуску YOLO в 2015 році було запропоновано кілька нових версій однієї моделі, кожна з яких базується на своїй попередниці та вдосконалює її [13]. На рисунку 2.7 зображена хронологія, яка демонструє розвиток YOLO за останні роки.

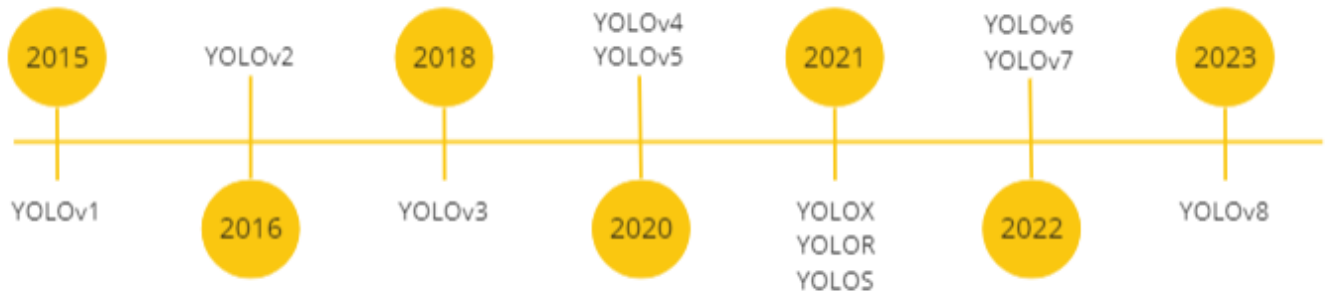

Рис.2.8. Еволюція YOLO

Дослідимо, як саме відрізняються моделі один від одного:

Таблиця 2.1 - Порівняння моделей YOLO[17]

Модель	Переваги	Недоліки
YOLOv1	Швидший, ніж інші існуючі рішення; Порівняно менше помилкових спрацьовувань виявляється на зображеннях зі складним фоном.	Якщо клітинка сітки містить більше одного об’єкта, модель не зможе виявити їх усі, це проблема виявлення близьких об'єктів, від якої страждає YOLO; Кількість об'єктів, які можна виявити, дорівнює кількості сіток, тобто в сітці 5x5 можна виявити максимум 25 об'єктів.
YOLOv2	Покращена швидкість і точність; використовує зображення для навчання класифікаторів, а також	Навчання моделі є тривалим, оскільки спочатку нам потрібно навчити мережу класифікаторів, а

	використовує зображення для виявлення об’єктів. Це полегшує навчання детектора та покращує mAP.	потім замінити підключені шари згортковими шарами та перенавчити модель.
YOLOv3	Втричі швидший, ніж попередні методи; Краще виявлення менших об'єктів на зображеннях; Виявляє виходи на трьох різних етапах, а не на кінцевому етапі виведення.	Вищі помилки позиціонування, через які низькі оцінки AP
YOLOv4	Модифіковані найсучасніші методи роблять їх більш ефективними та придатними для навчання на одному GPU; Висока швидкість виявлення при 65 FPS у системах реального часу	Введення такої кількості рівнів збільшує накладні витрати на визначення компромісу між mAP і швидкістю навчання та логічного висновку, щоб модель могла працювати у вбудованих системах.
YOLOv5	Він створений на основі PyTorch, що робить процес навчання та висновків дуже легким і швидким і допомагає досягти кращих результатів.	
YOLOv7	Покращена точність та швидкість виявлення при 160 FPS	
YOLOv8	Дуже висока точність виявлення об’єктів	має нижчий FPS, ніж YOLOv7 чи YOLOv5

Модель YOLOv8 являє собою значний прогрес у порівнянні з попередніми ітераціями як з точки зору точності, так і ефективності. Завдяки ретельним дослідженням і розробкам, YOLOv8 містить удосконалення, які призводять до

значного покращення продуктивності виявлення об’єктів. Ці вдосконалення охоплюють різні аспекти моделі, включаючи проектування архітектури, вилучення функцій і методи оптимізації.

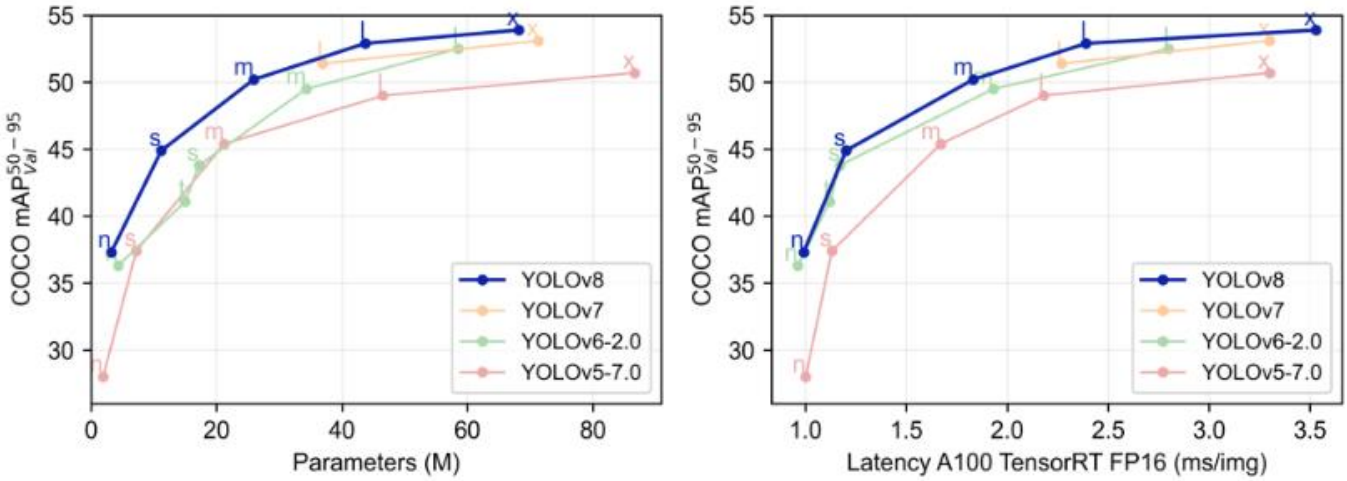

Рис.2.9. порівняння YOLOv8 з іншими версіями YOLO[13]

Архітектурні вдосконалення в YOLOv8 сприяють його чудовій продуктивності. Дослідники внесли цілеспрямовані вдосконалення в магістральну мережу, запровадивши нові архітектурні елементи, які отримують більш складну просторову та контекстну інформацію з вхідних даних. Використовуючи глибші та складніші структури нейронної мережі, YOLOv8 демонструє покращені можливості розпізнавання та локалізації об’єктів.

2.4. Порівняння архітектурних моделей класифікації пошкоджень

Після фази виявлення будівлі використовується згорткові мережі для класифікації пошкоджень. На попередньому етапі було описано двоступеневі архітектури і саме другим кроком для цих мереж було виконання класифікації локалізованого об’єкту.

Особливість класифікації пошкоджених об’єктів полягає в тому, що потрібно зрозуміти чи дійсно цей об’єкт був пошкоджений чи він вже мав такий вигляд до моменту катастрофи.

У дослідженні [21] проведено аналіз архітектур:

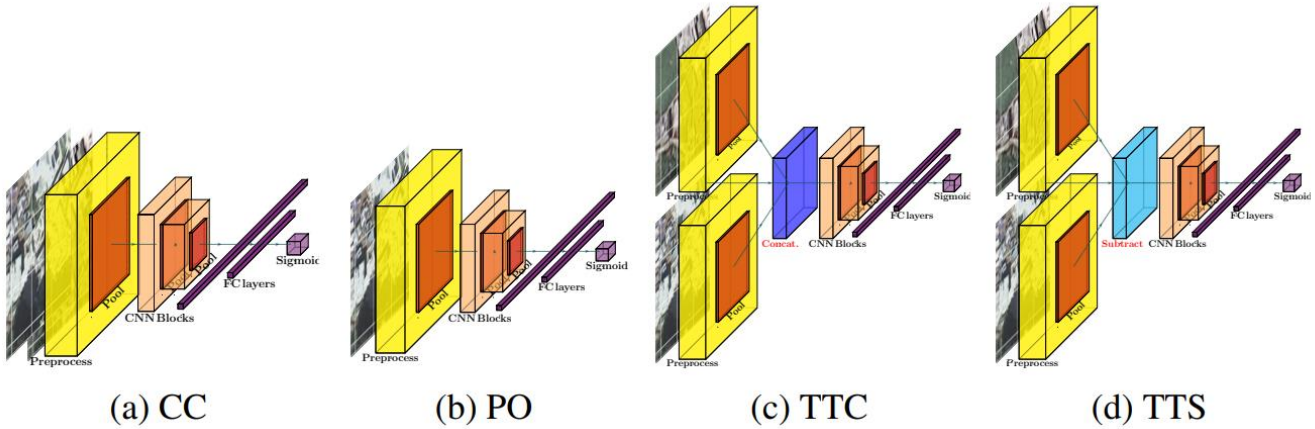

Рис.2.10. Архітектури класифікацій пошкоджень

Різниця між чотирма архітектурами полягає в тому, як обробляються вхідні зображення:

- Модель Concatenated Channel (CC). Об’єднує зображення до та після катастрофи в одне 6-канальне зображення.
- Модель Post-image Only (PO). Використовуйте лише 3-канальне зображення після катастрофи як вхідні дані. Ця модель втрачає інформацію із зображення до катастрофи, але уникає таких проблем, як зміщення та відмінності яскравості на зображеннях до та після катастрофи.
- Модель Twin-tower Concatenate (TTC). Попередня обробка зображень до та після катастрофи за допомогою окремих екстракторів згорткових функцій, а потім об’єднує витягнуті функції. Ця архітектура розроблена для порівняння зображень до та після катастрофи на основі абстрактних характеристик, виділених згортковими шарами, замість прямого порівняння пікселів. Це робить модель більш стійкою до нерівномірності зображень до та після катастрофи.
- Модель Twin-tower Substract (TTS). Те саме, що й TTC, за винятком того, що витягнуті ознаки комбінуються шляхом віднімання їх по елементах замість

об’єднання. Ця архітектура розроблена для більш точного відображення відмінностей у зображеннях до та після катастрофи, що є хорошим показником пошкодження будівлі [21].

Результати експерименту в [21] показують, що моделі twin-towers перевершують моделі з одним зображенням на вході, а модель TTS досягає найкращої продуктивності на валідаційному датасеті із показником AUC 0,830. Краща продуктивність моделей twin-towers вказує на те, що корисну інформацію можна отримати, порівнюючи будівлі та їх оточення на зображеннях після катастрофи з тими, що на зображеннях до катастрофи. Теоретично модель TTC є більш загальною і повинна мати можливість перевершувати рівень віднімання в моделі TTS і досягати рівного або кращого AUC, якщо це найкращий спосіб використання вхідних функцій. В роботі [21] цього не сталося, тому що навчальний набір був занадто малий, а модель TTC перенавчається на наявних даних. Ще один цікавий результат полягає в тому, що модель PO перевершує модель CC, яка містить значно більше інформації. Це свідчить про те, що просте об’єднання зображень до та після катастрофи без попереднього вилучення характеристик високого рівня не дозволяє моделі компенсувати відмінності між зображеннями, такі як зміщення об’єктів, відмінності кутів камери тощо [21].

2.5. Siamese класифікатор

У контексті класифікації пошкоджень будівель Siamese мережа відіграє ключову роль у розрізненні неушкоджених і пошкоджених будівель. Siamese модель належить до TTC типу. Використовуючи можливості порівняльного аналізу для двох зображень до та після катастрофи, модель може ефективно розрізняти характерні закономірності та особливості, що вказують на пошкодження будівель. Siamese мережа дозволяє виділяти дискримінаційні ознаки, дозволяючи приймати точні рішення щодо класифікації.

Архітектура сіамської мережі складається з двох ідентичних підмереж, які мають однакову вагу та архітектуру. Кожна підмережа обробляє вхідне зображення, витягуючи релевантні функції та генеруючи представлення високого рівня. Потім ці представлення порівнюються, щоб оцінити подібність або відмінність між парами зображень.

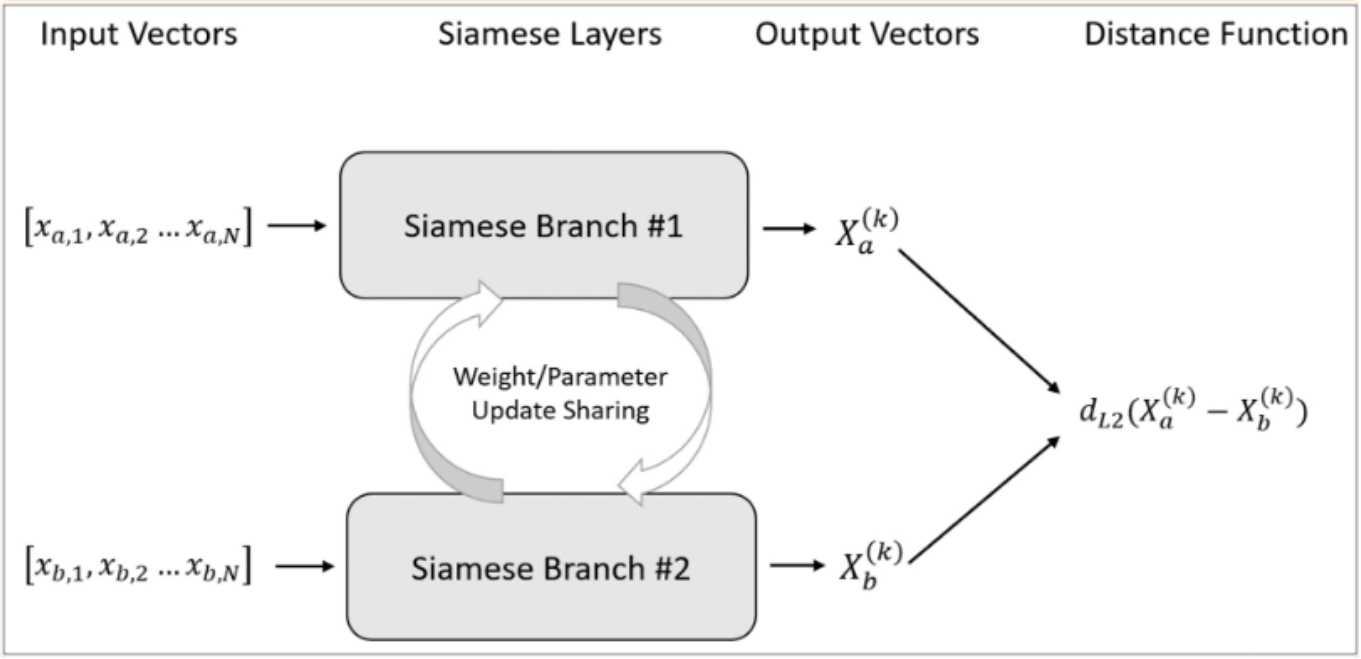

Рис.2.11. Стандартна Siamese модель[15]

Створені вектори після Siamese шарів використовуються для обчислення метричного вектора відстані, зазвичай L2-відстані, який використовується як вихідний сигнал мережі та може розумітися як вивчена відстань між двома входами. Існують різноманітні функції втрати, які можна використовувати з виходами сіамської мережі, але всі вони в основному обертаються навколо максимізації або мінімізації відстані між векторами[15].

Стандартну модель не можна застосувати для класифікації зображень. Але є можливість проаналізувати наскільки сильно схожі зображення до і після катастрофи. Поточне дослідження спрямоване для покращеної діагностики та класифікації кількох класів пошкодженості будинків. Модель повністю ідентична до стандартної моделі Siamese, але із використанням конкатенації вихідних шарів Siamese моделі для подальшої класифікації з використанням softmax функції. Це дасть змогу

класифікувати декілька ступенів пошкодженості будинків, такі як: "no-damage "minor-damage", "major-damage" та "destroyed".

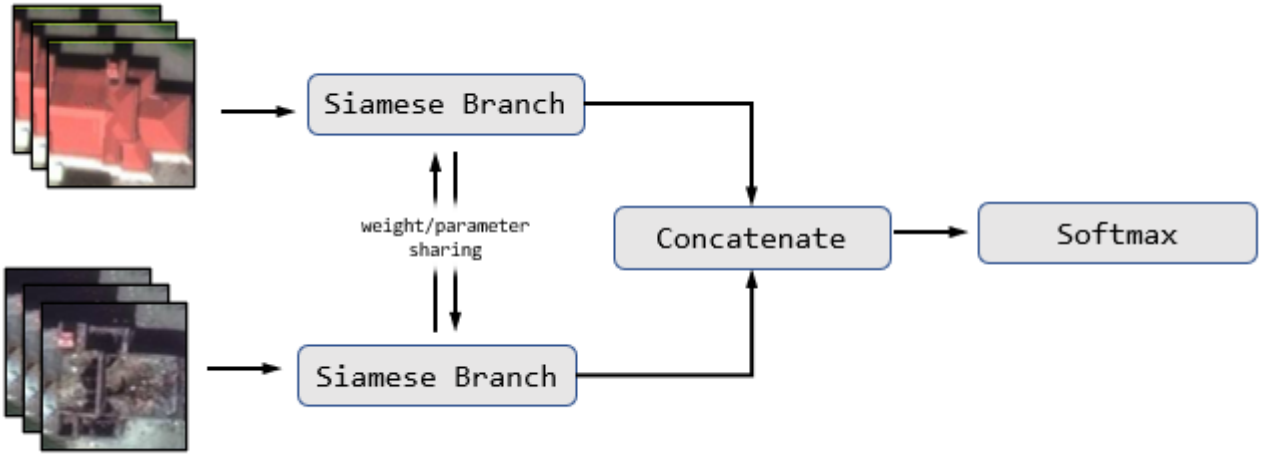

Рис.2.12. Модифікована Siamese модель

Інтеграція моделей виявлення для локалізації будівель і подальше використання згорткових мереж для класифікації пошкоджень пропонують помітні переваги. По-перше, застосування моделей виявлення дозволяє ефективно та точно ідентифікувати будівлі на супутникових зображеннях, уможливлюючи цілеспрямований аналіз, зосереджений виключно на відповідних регіонах інтересу. Це прискорює загальний процес аналізу зображення, оскільки нерелевантні ділянки ігноруються, що призводить до підвищення ефективності обчислень і скорочення часу обробки.

По-друге, включення сіамської мережі полегшує точну класифікацію пошкоджень. Порівняльний аналіз, проведений сіамською мережею, дозволяє детально оцінити візуальні характеристики непошкоджених і пошкоджених будівель. Використовуючи потужність глибокого навчання та можливості вилучення функцій CNN, методологія забезпечує точні та надійні результати класифікації, підвищуючи загальну ефективність структури.

Крім того, поєднання моделей виявлення та Siamese мережі використовує сильні сторони кожного компонента, створюючи синергетичну структуру для виявлення пошкоджень будівель. Моделі виявлення надають точну інформацію

обмежувальної рамки, звужуючи аналіз до відповідних областей інтересу. Згодом сіамська мережа використовує можливості порівняльного аналізу CNN, дозволяючи приймати точні та точні рішення щодо класифікації пошкоджень на основі вивчених представлень характеристик.

На завершення запропонована методологія включає в себе моделі виявлення для локалізації будівель і згорткову мережу, як Siamese мережу для класифікації пошкоджень. Цей двоетапний підхід прискорює процес аналізу зображення, одночасно забезпечуючи точну ідентифікацію та класифікацію пошкоджень будівлі. Використання моделей виявлення для точної локалізації будівлі, а також можливості порівняльного аналізу CNN дозволяють приймати ефективні та точні рішення в оцінці пошкоджень будівлі. Використовуючи сильні сторони обох компонентів, методологія досягає балансу між швидкістю та точністю, сприяючи ефективному аналізу зображень у контексті виявлення пошкоджень будівель.

2.6. Висновок

В цьому розділі наведено огляд існуючих підходів для виявлення об’єктів. Детально розглянуто одноступеневі методи виявлення об’єктів їх види та еволюції розвитку. YOLO, найсучасніша модель виявлення об’єктів, демонструє виняткову продуктивність у реальному часі і ефективно локалізує будівлі на супутникових зображеннях, забезпечуючи попередній набір регіонів-кандидатів для подальшого аналізу.

Siamese модель пропонує потужні можливості для класифікації пошкоджень. Розроблена для вимірювання подібності між двома вхідними даними, siamese мережа чудово розрізняє пошкоджені та непошкоджені ділянки в ідентифікованих будівлях. Навчаючись на парах фрагментів зображень, що містять пошкоджені та непошкоджені ділянки, сіамська мережа вивчає метрику подібності, яка дозволяє їй точно класифікувати ступінь пошкодження.

Загалом, моделі YOLO та Siamese служать важливими компонентами двоетапної моделі навчання, пропонуючи додаткові переваги у виявленні будівель та класифікації пошкоджень. Їх інтеграція дає змогу моделі точно ідентифікувати та класифікувати пошкоджені будівлі на основі супутникових зображень, таким чином підтримуючи ефективне реагування на стихійні лиха, гуманітарні зусилля та міське планування. У міру подальших досліджень і розробок у цій галузі очікується, що можливості та ефективність цих моделей зростатимуть, що призведе до ще більш надійних систем виявлення пошкоджень будівель у майбутньому.

РОЗДІЛ 3. РЕЗУЛЬТАТИ РОБОТИ

3.1. Збір і підготовка даних

Для того щоб змоделювати складний і динамічний характер пошкоджень будівель, потрібні зображення, що містять багато типів пошкоджень і таких зображень має бути у великій кількості. Саме з цією метою був представлений великомасштабний набір даних супутникових зображень xBD. xBD охоплює різноманітний набір катастроф і географічних місць із понад 800 000 анотацій будівель на понад 45 000 км^2 зображень. Щоб підготувати цей набір даних, працювали експерти з реагування на катастрофи з усього світу, які спеціалізуються на різних типах катастроф, щоб створити шкалу анотацій, яка точно представляє реальні умови шкоди [16].

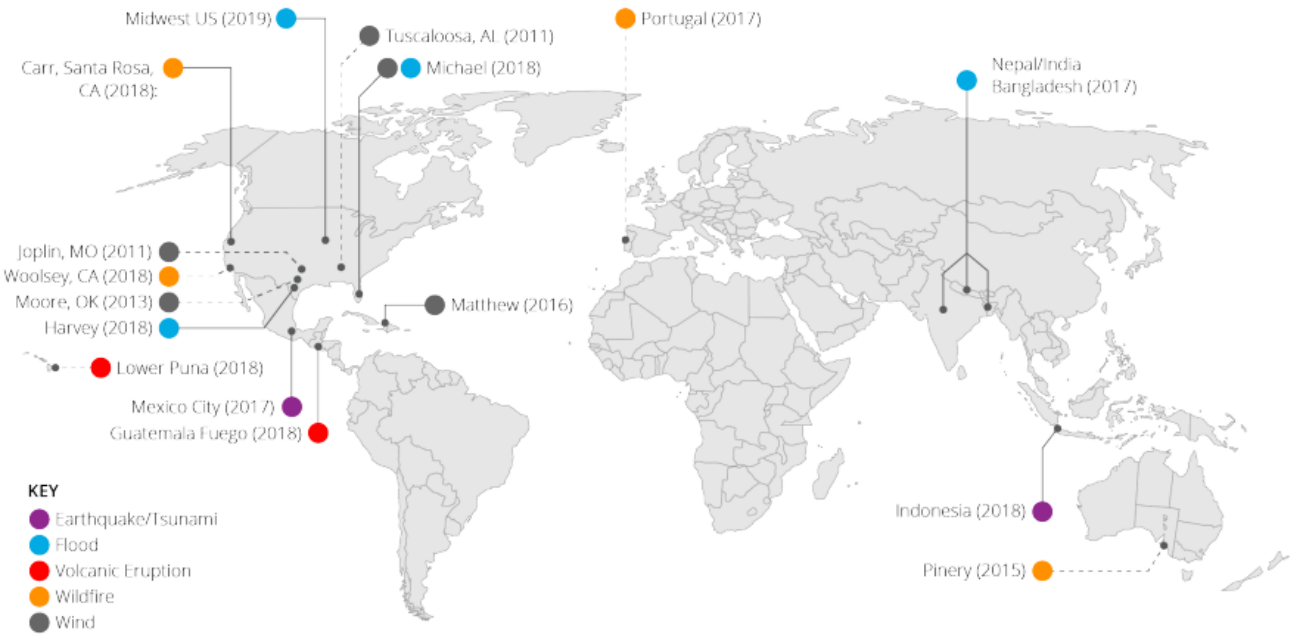

Рис.3.1 Типи катастроф і катастрофи, представлені в xBD по всьому світу [16]

Набір даних xBD використовується в призовому конкурсі xView2 [17], метою якого є стимулювання створення точних і ефективних моделей машинного навчання,

які оцінюють пошкодження будівель на основі супутникових зображень до та після катастрофи.

Однією з цілей завдання xView2 є створення моделей, які широко застосовуються для великої кількості катастроф. Це дозволить кільком агентствам із реагування на катастрофи потенційно зменшити своє робоче навантаження за допомогою однієї моделі з відомим циклом розгортання. xBD мав би бути репрезентативним для багатьох типів катастроф, а не лише для тих, для яких велика кількість даних була загальнодоступною.

Метою даної роботи побудувати модель для того, щоб ефективно аналізувати пошкодженні або зруйновані будинки в результаті бойових дій на території України. Тому для попереднього навчання було використано лише зображення із таких місць-трагедій як: 'socal-fire', 'santa-rosa-wildfire', 'mexico-earthquake', 'palu-tsunami', 'hurricane-michael'. Тому що будинки зруйновані або пошкодженні на цих зображення є схожими, що в Україні.

Розглянемо в таблиці 3.1 як саме оцінюється ступінь пошкодження та які будинки належать до кожного класу [16].

Таблиця 3.1 - ступінь пошкодження

Ступінь пошкодження	Опис
0. No Damage	Жодних слідів структурних пошкоджень або пошкоджень черепиці чи слідів горіння.
1. Minor Damage	Частково обгоріла будівля, елементи даху відсутні або є видимі тріщини.
2. Major Damage	Часткове обвалення стіни чи даху
3. Destroyed	Повністю зруйнована чи обгоріла будівля

Інша частина супутникових даних, використаних в дипломні роботі, були отримані з двох основних джерел: веб-сайту Maxar [18] і Google Maps Pro. Ці джерела

надали супутникові знімки високої роздільної здатності, які були зібрані над територією деокупованих територій на півночі України та Маріуполя. Використання супутникових даних із цих авторитетних джерел забезпечує надійність і якість зображень, які використовуються для виявлення та класифікації пошкоджень будівель.

Веб-сайт Maxar пропонує повну колекцію супутникових знімків, отриманих із різних супутників, оснащених передовими датчиками зображення. Ці супутники фіксують зображення високої роздільної здатності, що дозволяє детально аналізувати цільові області. Використання знімків Maxar забезпечує доступ до актуальних і точних супутникових даних, що полегшує оцінку пошкоджень будівель у вказаних регіонах.

Окрім веб-сайту Maxar, супутникові дані також збиралися з Google Maps Pro. Google Maps Pro надає платформу, яка пропонує доступ до широкого спектру геопросторових даних, включаючи супутникові зображення високої роздільної здатності. Супутникові зображення, доступні на цій платформі, отримані від багатьох супутникових постачальників, що сприяє різноманітності та охопленню даних.

Після того, як супутникові дані були зібрані, вони пройшли подальшу обробку та анотації. Зібрані зображення були позначені та анотовані на платформі Roboflow [19], широко використовуваному інструменті для маркування та попередньої обробки даних у програмах комп’ютерного зору. Roboflow сприяв ефективній анотації супутникових зображень, надаючи інтуїтивно зрозумілі інструменти анотації та робочі процеси.

Процес анотації передбачав позначення будівель, на супутникових зображеннях. Цей крок був вирішальним для навчання моделей згорткової нейронної мережі для точного виявлення та класифікації пошкоджень будівель. Окресливши межі будівель і позначивши їх відповідним чином, анотовані дані надали базову правдиву інформацію, необхідну для навчання та оцінки моделей CNN.

Roboflow дозволив систематичну та послідовну анотацію зібраних супутникових даних. Платформа підтримувала створення обмежувальних рамок навколо цікавих будівель, забезпечуючи точну локалізацію та розмежування.

Використання Roboflow для анотації даних підвищило ефективність і точність загального процесу дослідження. Інструменти анотації платформи оптимізували робочий процес анотації, скоротивши час і зусилля, необхідні для анотації вручну. Крім того, здатність Roboflow обробляти великомасштабні набори даних забезпечила масштабованість процесу анотації, враховуючи значний обсяг супутникових зображень, зібраних з територій України.

3.2. Оцінювальні метрики, що використовуються для виявлення пошкоджень будівлі

При оцінці продуктивності моделі виявлення пошкоджень важливо враховувати конкретні оціночні метрики як для виявлення будівель, так і для класифікації пошкоджень. Ці показники надають кількісні показники для оцінки точності, точності, запам’ятовування та загальної ефективності процесів виявлення та класифікації. У цьому розділі розглядаються метрики оцінювання, які зазвичай використовуються у сфері виявлення пошкоджень, розділені на метрики для виявлення будівель і метрики для класифікації пошкоджень.

Для виявлення будівель метрики оцінювання зосереджені на оцінці здатності системи точно ідентифікувати та локалізувати будівлі на супутникових зображеннях. Зазвичай використовуються такі показники:

Intersection over Union (IoU): IoU вимірює перекриття між передбачуваною обмежувальною рамкою та обмежувальною рамкою правди на землі будівлі. Він обчислюється як відношення площі перетину до об’єднаної площі двох обмежувальних рамок. Вищі значення IoU вказують на кращу точність локалізації.

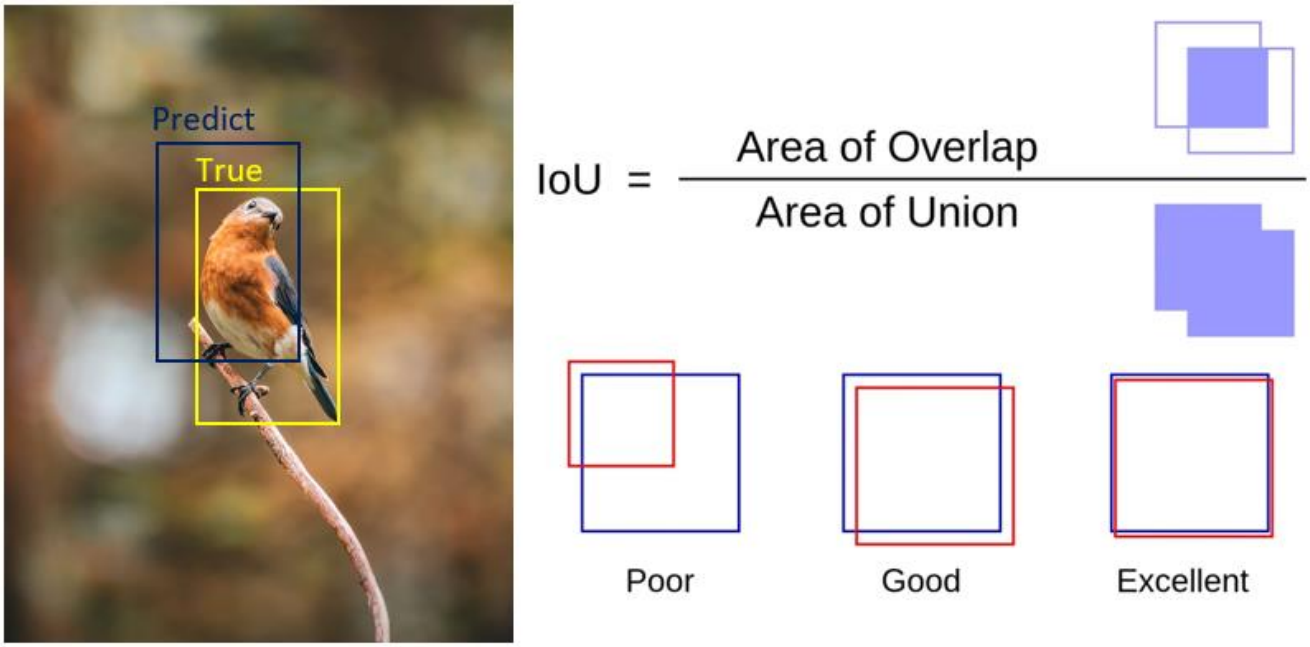

Рис.3.2. Оцінка IoU – це перекриття між істинною обмежувальною рамкою та прогнозованою обмежувальною рамкою

Intersection over the union коливається від 0, коли загалом немає перекриття до 1, коли дві обмежувальні рамки перекривають одна одну на 100%. Чим вище перекриття між двома обмежувальні рамки, тим краще

Precision and Recall: precision вимірює частку правильно виявлених будівель серед усіх прогнозованих будівель, тоді як recall вимірює частку правильно виявлених будівель серед усіх наземних істинних будівель. Ці показники дають уявлення про здатність системи мінімізувати помилкові спрацьовування (precision) і помилкові негативи (recall) при виявленні будівель.

$$Recall = \frac{TP}{TP+FN} \tag{3.1}$$

$$Precision = \frac{TP}{TP+FP} \tag{3.2}$$

Average Precision: AP підсумовує криву точності-пригадування шляхом обчислення середніх значень точності на різних рівнях пригадування. Він забезпечує загальну оцінку здатності системи точно виявляти будівлі за різними порогами.

Так само оцінка F1 є особливо корисною для визначення оптимальної впевненості, яка балансує значення точності та запам’ятовування для даної моделі.

$$F_1 = 2\frac{precision \cdot recall}{precision+recall} \quad (3.3)$$

Що стосується класифікації пошкоджень, показники оцінки зосереджені на оцінці продуктивності системи щодо точної класифікації виявлених будівель на пошкоджені та непошкоджені категорії. Зазвичай використовуються такі показники:

Accuracy: accuracy вимірює частку правильно класифікованих будівель серед усіх класифікованих будівель. Він забезпечує загальний показник ефективності класифікації системи.

Precision, Recall: precision, recall оцінюють здатність системи правильно класифікувати пошкоджені та непошкоджені будівлі. Precision являє собою частку правильно класифікованих пошкоджених будівель серед усіх прогнозованих пошкоджених будівель. Recall вимірює частку правильно класифікованих пошкоджених будівель серед усіх фактично пошкоджених будівель. AUC-score: ROC – це графік, який відображає зв’язок між істинним позитивним показником (TP) і хибним позитивним показником (FP), показуючи TP, який ми можемо очікувати отримати за певного компромісу з FP. Оцінка AUC — це площа під цією кривою ROC, що означає, що отримана оцінка в широкому сенсі відображає здатність моделі правильно передбачати класи. AUC не є прямим показником точності моделі, це ймовірність того, що модель присвоїть більшу ймовірність випадковому позитивному прикладу, ніж випадковому негативному . Однак це можна інтерпретувати як форму точності, де чим вищий бал, тим більша ймовірність того, що будуть зроблені правильні прогнози[22].

Confusion Matrix: confusion matrix відображає розподіл прогнозованих і обґрунтованих класифікацій правдивості, що дозволяє детально проаналізувати справжні позитивні, справжні негативні, помилкові позитивні та помилкові негативні результати. Він забезпечує повне розуміння ефективності класифікації та потенційних областей для покращення.

Ці оціночні метрики забезпечують кількісні показники для оцінки продуктивності систем виявлення пошкоджень за допомогою CNN. Розглядаючи показники як для виявлення будівель, так і для класифікації пошкоджень, що

допомагають отримати уявлення про precision, recall, accuracy та загальну ефективність системи. Ці показники служать цінними інструментами для порівняння різних моделей, точного налаштування параметрів і порівняння з встановленими стандартами в області виявлення пошкоджень.

3.3. Модель навчання

Як і обговорювалося в попередніх розділах, запропонована модель для виявлення пошкоджень будівлі передбачає двоетапний підхід. По-перше, моделі виявлення використовуються для ідентифікації та локалізації будівель на супутникових зображеннях шляхом створення обмежувальних рамок. Моделі виявлення надають точну інформацію обмежувальної рамки, звужуючи аналіз до відповідних областей інтересу (рис.3.1).

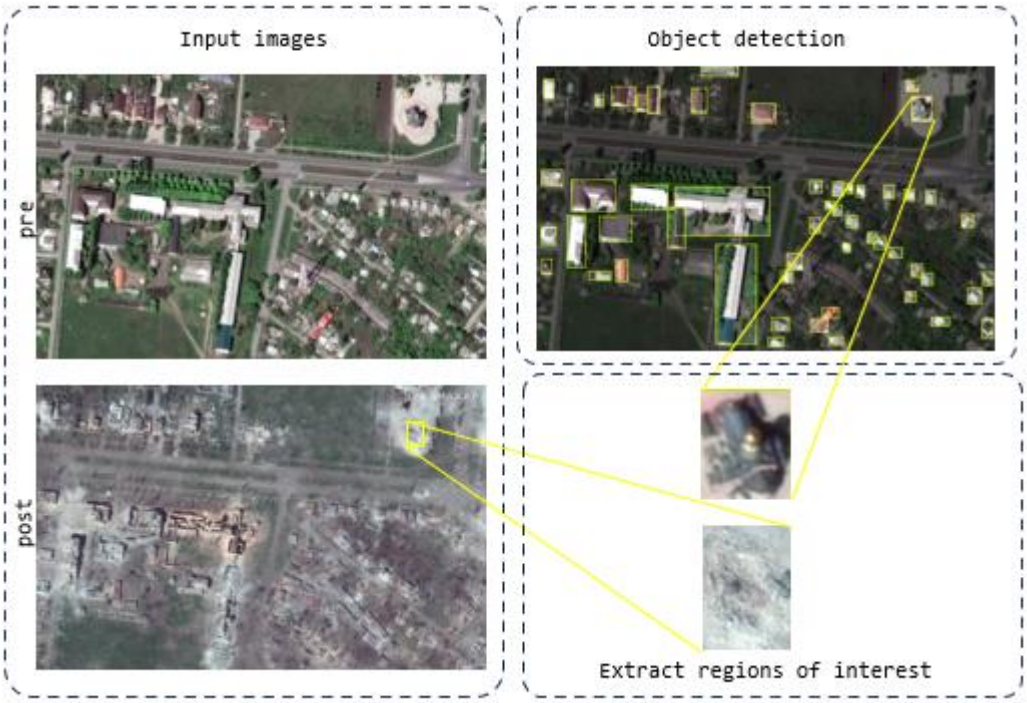

Рис.3.1. Знаходження областей інтересу

Для виявлення будівель на супутникових зображеннях використовується модель YOLOv8s. Дана модель забезпечує швидке й точне виявлення об’єктів. Адаптований для виявлення будівель, YOLO доводить свою важливу роль у ефективній локалізації будівель на супутникових зображеннях, таким чином забезпечуючи початковий набір регіонів-кандидатів для подальшого аналізу.

Після того як визначили області інтересу, для класифікації пошкоджень використовується згортка нейронної мережі типу Siamese моделі. Ця комбінована структура спрямована на прискорення аналізу зображень, одночасно забезпечуючи точну та ефективну ідентифікацію та класифікацію пошкоджень будівель.

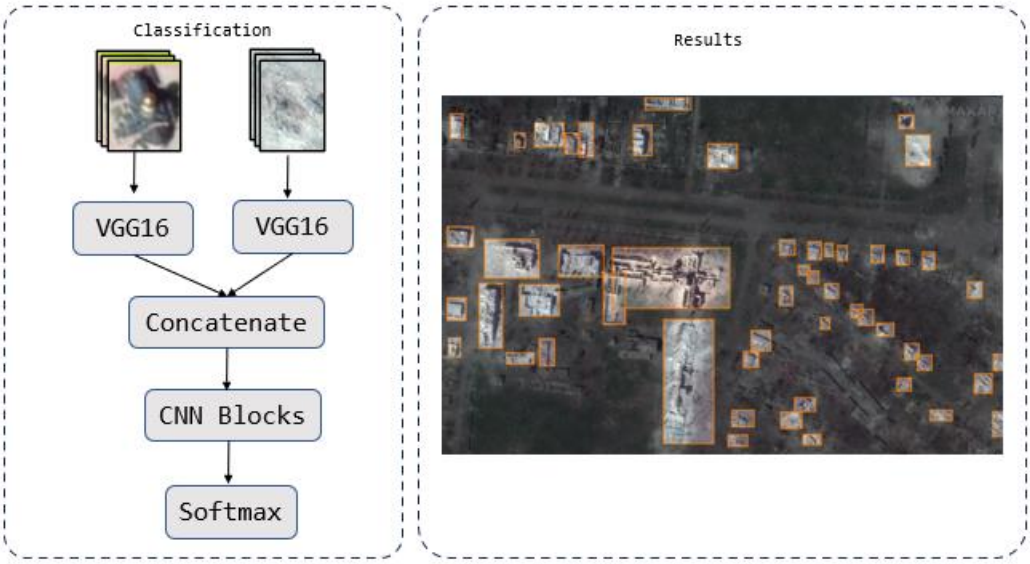

Рис.3.2. Модель класифікації пошкоджених будинків

Для реалізації мережі класифікації було використано Tensorflow та Keras. А в якості моделі для вилучення ознак Siamese моделі було використано VGG16.

VGGNet був розроблений у 2014 році Visual Geometry Group в Оксфордському університеті (звідси назва VGG). VGG16, складається з 16 вагових шарів: 13 згорткових шарів і 3 повністю зв'язаних шарів [20]. Його уніфікована архітектура

робить його привабливим для розробників глибинного навчання, оскільки його дуже легко зрозуміти.

Детальний граф моделі класифікації зображено на рисунку 3.3.

Рис.3.3. Граф моделі класифікації

3.4. Результати

На рисунку 3.4. зображено результати роботи моделі.

Рис.3.4. Результати моделі

no damage, minor damage, major damage, destroyed

3.5. Оцінка ефективності моделей виявлення об'єктів і класифікації пошкоджень будівель

Щоб модель виявлення пошкоджень була практично корисною, вона повинна добре працювати під час майбутніх катастроф. Іншими словами, модель повинна добре узагальнювати катастрофи, на яких її не навчали. Типовим способом покращення узагальнення в моделях ML є збільшення розміру та варіації навчальних даних.

Це проблема, оскільки існує лише невелика кількість минулих катастроф, для яких доступні супутникові зображення високої роздільної здатності та ручна оцінка збитків. Це означає, що існує обмежена варіація характеристик будівлі, умов освітлення, типів рельєфу, якості супутникового зображення та ракурсів камери в навчальних даних. Відсутність мінливості навчальних даних означає, що модель може легко перенавчатися на навчальних даних та працювати погано поза навчальною вибіркою.

У цьому розділі проведено оцінювання, наскільки добре модель може узагальнюватися в умовах обмеженої мінливості навчальних даних.

Аналіз ефективності моделі YOLOv8s:

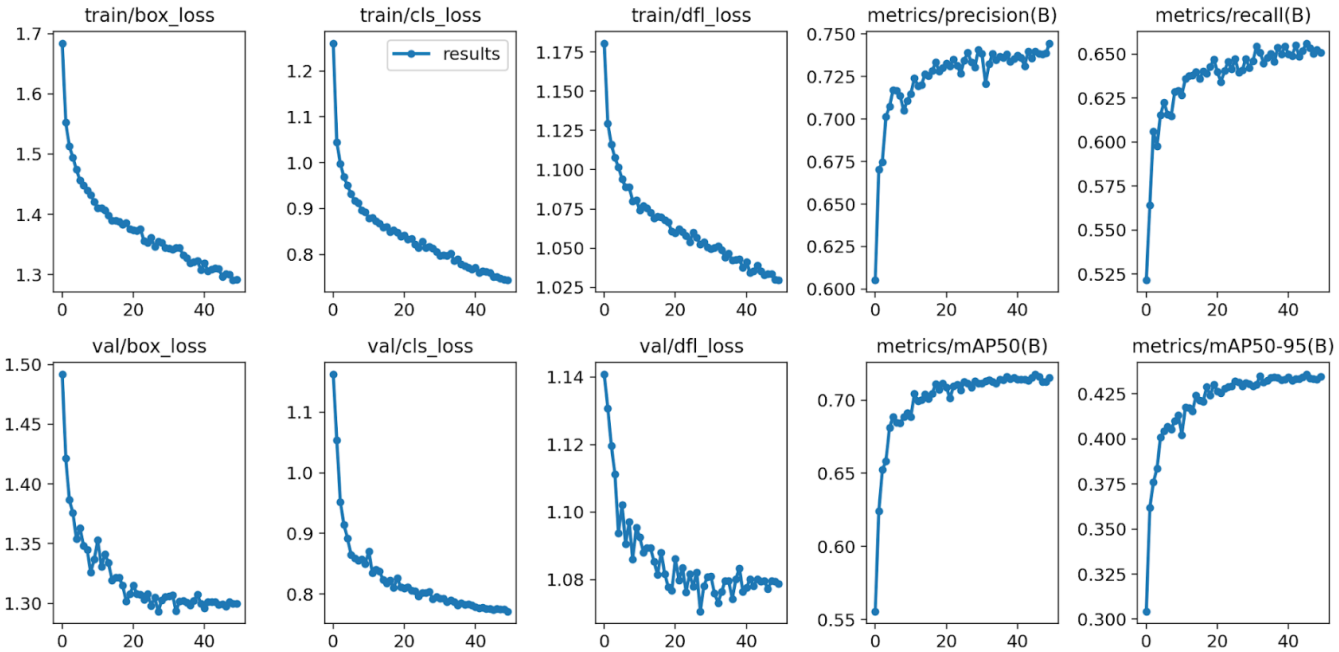

Рис.3.5. метрики ефективності визначення будинків моделі YOLO

Для YOLO дуже добре працюють метрики precision і recall під час оцінювання моделі за різних значень довіри, надаючи цінну інформацію про те, як працює модель і які значення оптимізують продуктивність моделі. Як правило, коли ви збільшуєте поріг достовірності, precision зростатиме, а recall зменшуватиметься, як показано в результатах наведених на рисунку 3.6:

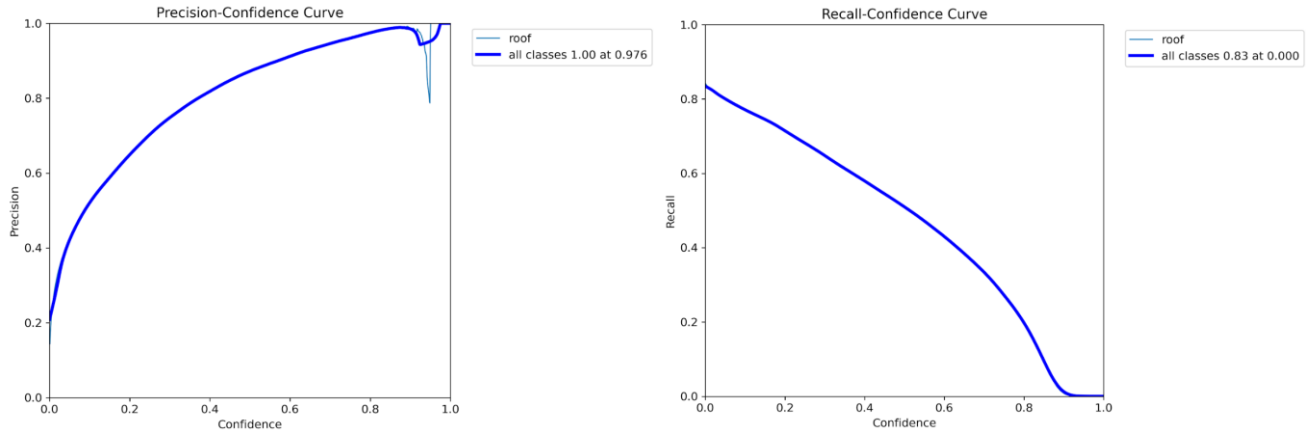

Рис.3.6. Метрики precision і recall

За допомогою кривої показників F1 можна візуалізувати баланс між метриками precision і recall і визначити точку проектування за допомогою рисунку нижче:

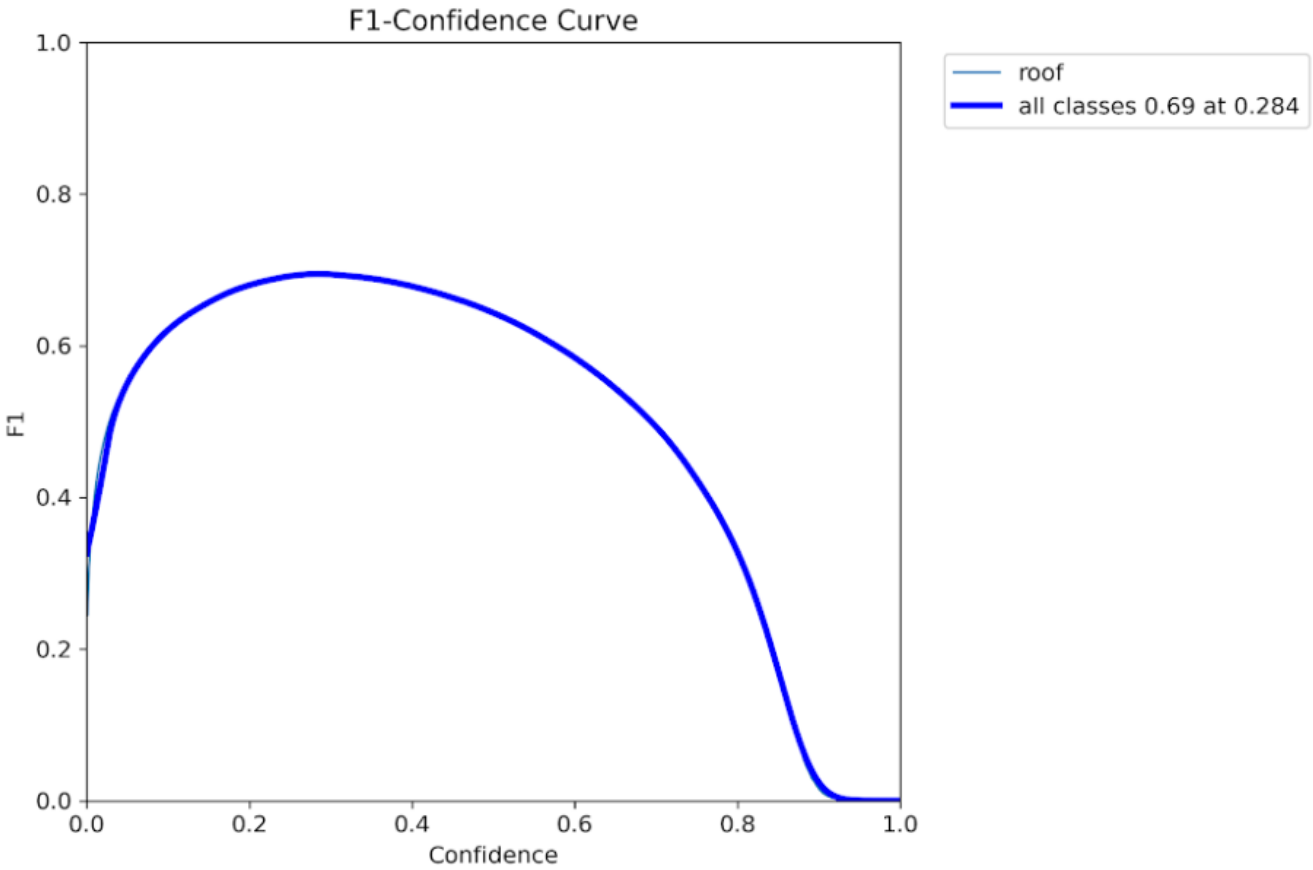

Рис.3.7. F1-Confidence curve

З кривої F1 достовірне значення, яке оптимізує precision і recall, становить 0,284 з максимальним значенням f1 0.69.

Щоб проаналізувати результати моделі класифікації пошкоджень будівлі, було використано кілька оціночних метрик для оцінки її ефективності. Ці показники дають уявлення про загальну ефективність моделі в класифікації пошкоджених і непошкоджених будівель. Аналіз результатів має на меті оцінити ефективність моделі, визначити сильні та слабкі сторони, а також виділити сфери потенційного вдосконалення.

У нас дані більш-менш збалансовані тому можна використовувати для аналізу ефективності, як accuracy так і auc метрики.

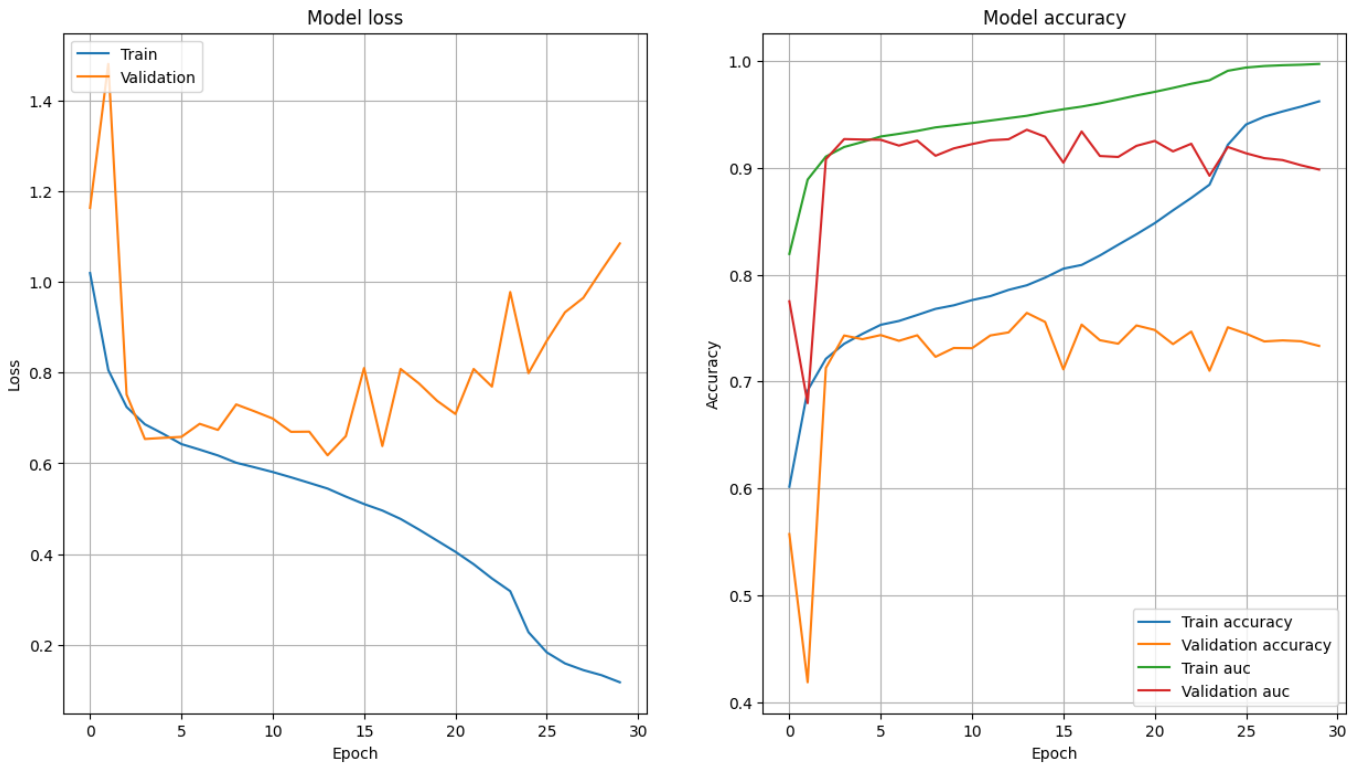

Рис.3.8. графік навчання моделі класифікації

Найкраще значення accuracy 76.42%, f1 score 68.76, а найкраще значення AUC 93.56%

3.6. Порівняння запропонованого методу з існуючими підходами

Для першого етапу метода використовувалася звичайна модель YOLO, і дана модель немає особливих відмінностей від уже існуючих моделей.

Порівнюючи модель класифікації для виявлення пошкоджень будівлі з іншими існуючими моделями, важливо враховувати їхні сильні сторони, обмеження та продуктивність на подібних наборах даних. Уже було запропоновано кілька моделей для виявлення пошкоджень будівель, і аналіз їх ефективності може дати розуміння ефективності даної моделі класифікації.

Важливо зазначити, що продуктивність цих моделей може відрізнятися залежно від набору даних, якості зображення та конкретних характеристик виявленого пошкодження будівлі. Такі фактори, як розмір набору даних, дисбаланс класів і доступність позначених навчальних даних, також можуть впливати на продуктивність моделі. Тому вкрай важливо враховувати ці фактори під час порівняння моделі класифікації з іншими існуючими моделями та інтерпретації результатів.

Так найкраща модель Google AI була досягнута шляхом створення валідаційного набору даних із того самого міста, на даних якого тренувалася модель.

Таблиця 3.2 – порівняння існуючих підходів класифікаціїї пошкоджених будівель

Model	Accuracy	AUC	F1-score
Ours	0.7642	**0.9356**	0.6876
Google AI [21] (average)	0.6825	0.74	-
Google AI [21] (best)	**0.78**	0.86	-
ResNet18 [23]	-	-	0.68
Se-ResNext50 [23]	-	-	0.7776
xView2 [24]	-	-	0.41
BDANet [25]	-	-	**0.782**

3.7. Висновок

В даному розділі було проведено опис даних, які використовувалися для навчання моделі. Опис основних метрик, які використовувалися для аналізу продуктивності моделі. І запропонована сама модель для виявлення пошкоджень будівлі, що передбачає двоетапний підхід. По-перше, моделі виявлення використовуються для ідентифікації та локалізації будівель на супутникових зображеннях шляхом створення обмежувальних рамок. Моделі виявлення надають точну інформацію обмежувальної рамки, звужуючи аналіз до відповідних областей інтересу. Згодом для класифікації пошкоджень використовується згортка нейронної мережі типу Siamese моделі. Ця комбінована структура спрямована на прискорення аналізу зображень, одночасно забезпечуючи точну та ефективну ідентифікацію та класифікацію пошкоджень будівель.

Поєднання YOLO та Siamese моделі в двоетапній моделі навчання пропонує кілька переваг. По-перше, він підвищує точність виявлення пошкоджень будівель, використовуючи точність локалізації будівель YOLO та дискримінаційну силу класифікації пошкоджень. По-друге, модель досягає балансу між точністю та обчислювальною ефективністю, оскільки продуктивність YOLO в реальному часі дозволяє швидко ідентифікувати регіони-кандидати, зменшуючи простір пошуку для класифікації пошкоджень. По-третє, модель демонструє високий ступінь узагальненості, що робить її застосовною до різноманітних географічних місць і різних сценаріїв лиха.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі представлено комплексну оцінку ключових аспектів, що стосуються розробки програмного продукту, спрямованого на вирішення складної проблеми визначення пошкоджених будинків на супутникових зображень за допомогою моделі нейронної мережі, створеної в попередньому розділі.

Щоб вибрати найбільш оптимальний підхід до впровадження, було проведено ретельний аналіз різних варіантів, враховуючи як економічні фактори, так і характеристики продукту, які безпосередньо впливають на ефективність роботи та сумісність апаратного забезпечення. Для цього використовується методологія функціонально-вартісного аналізу (ФВА), потужна техніка, яка дає змогу оцінити справжню вартість продукту чи послуги, незалежно від організаційної структури відповідної компанії.

Функціональний аналіз витрат виконується з метою визначення можливого зниження витрат шляхом вивчення більш ефективних альтернатив виробництва та досягнення покращеного балансу між споживчою вартістю продукту та витратами на його виробництво. Цей аналітичний процес включає економічну, технічну та проектну інформацію для отримання глибоких висновків.

Алгоритм функціонально-вартісного аналізу включає кілька етапів, включаючи визначення послідовності розробки продукту, розрахунок загальних річних витрат і необхідної кількості робочих годин, визначення джерел витрат і в кінцевому підсумку розрахунок загальної вартості програмного продукту. Ретельно дотримуючись цього алгоритму, можна досягти повного розуміння фінансових наслідків і розподілу ресурсів, пов’язаних із розробкою програмного продукту.

4.1 Постановка задачі проектування

У цій роботі використовується методологія функціонального аналізу вартості для проведення комплексної технічної та економічної оцінки розробки послуги, спеціально розробленої для ідентифікації пошкоджених будівель за допомогою супутникових зображень. Враховуючи, що рішення, що стосуються проектування та реалізації окремих компонентів, мають значний вплив на всю систему, кожна підсистема повинна відповідати певним критеріям. Тому аналіз в першу чергу зосереджений на функціональних можливостях програмного продукту, призначеного для навчання нейронної мережі та подальшого його застосування для вирішення конкретних практичних завдань, пов’язаних із сегментацією супутникових знімків.

Технічні вимоги до програмного продукту викладені таким чином:

1. Сумісність: програмний продукт повинен безперебійно працювати на персональних комп'ютерах, оснащених стандартним набором компонентів. Це гарантує, що його можна легко розгорнути в різних обчислювальних середовищах, не вимагаючи спеціальних конфігурацій обладнання.
2. Зручність для користувача: підкреслюючи зручність і зрозумілість, програмний продукт повинен мати інтуїтивно зрозумілий і зручний інтерфейс. Це дозволяє користувачам, незалежно від їхнього технічного досвіду, ефективно взаємодіяти з системою та з легкістю використовувати її функції.
3. Обробка даних у режимі реального часу: програмний продукт має мати ефективні можливості обробки даних, забезпечуючи швидкий доступ до інформації в режимі реального часу. Це забезпечує своєчасний аналіз і прийняття рішень на основі супутникових зображень, що дозволяє швидко реагувати для точної ідентифікації пошкоджених будівель.
4. Масштабованість і обслуговування: важливо, щоб програмний продукт підтримував зручну масштабованість, дозволяючи потенційне розширення або адаптацію для роботи з більшими наборами даних або розміщення

додаткових функцій у майбутньому. Крім того, система має бути розроблена з урахуванням простоти обслуговування, мінімізації зусиль і ресурсів, необхідних для оновлення, налагодження та підвищення продуктивності з часом.

5. Економічна ефективність: програмний продукт має бути спрямований на мінімальні витрати на впровадження, шукаючи оптимальний розподіл ресурсів без шкоди для функціональності чи продуктивності. Завдяки оптимізації економічної ефективності продукт стає більш доступним і життєздатним для розгортання в широкому діапазоні застосувань і галузей.

Дотримуючись цих технічних вимог, розроблений програмний продукт дозволить ефективно вирішувати завдання ідентифікації пошкоджених будівель на супутникових зображеннях, відповідаючи бажаним стандартам продуктивності, зручності використання, масштабованості та доступності.

4.2. Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу пошкоджених будинків на супутникових зображень. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція $F1$:

а) Python;

б) C++.

Функція F_2:

а) Tensorflow;

б) Pytorch.

Функція F_3:

а) Google Colab;

б) PyCharm.

Варіанти реалізації основних функцій наведені у морфологічній карті. Рисунок 4.1 демонструє морфологічну карту.

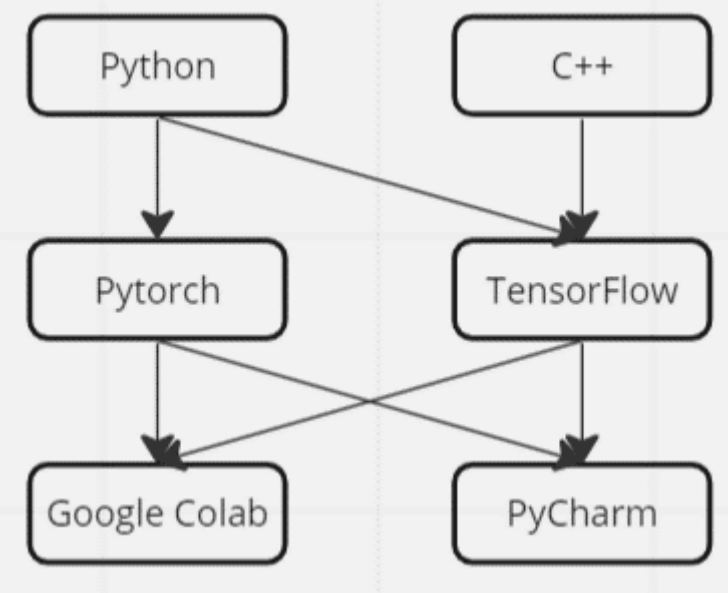

Рис.4.1. Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.1. Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Інтуїтивна та швидка розробка програми, велика кількість бібліотек	Низька швидкість виконання

	Б	Висока швидкість виконання	На розробку програмного коду витрачається багато часу
F_2	А	Краще будувати великі та складні проєкти, є велика спільнота користувачів	Складно зрозуміти для новачків
	Б	Зручний та простий у використанні	Не використовується багатьма мовами програмування
F_3	А	Не потребує встановлення бібліотек, має великі потужності	Немає розширених функцій, обмежений час використання GPU, для використання потрібен інтернет
	Б	Зручний у використані та має багато інструментів	Для використання потрібні потужні обчислювальні ресурси

Функція F_1:

Перевага тут однозначно на стороні Python. Тому що дана мова, хоч і працює повільніше, ніж C++, але за допомогою своїх бібліотек, які часто написані на C++, дає змогу швидко із меншою кількістю строк коду писати програми. Для спрощення роботи варіант Б відкидаємо.

Функція F_2:

TensorFlow і PyTorch — два популярних фреймворки для глибокого та машинного навчання. Хоча вони мають схожі цілі та функції, між ними є деякі помітні відмінності. PyTorch часто вважають зручнішим і легшим для розуміння завдяки його інтуїтивно зрозумілому та динамічному обчислювальному графіку. Це дозволяє користувачам визначати та змінювати свої моделі на льоту, що робить його зручним для досліджень та експериментів. З іншого боку, TensorFlow має більш статичне

визначення графа, що може бути складнішим для початківців, але надає можливості оптимізації для розгортання виробничого рівня.

Зрештою, вибір між TensorFlow і PyTorch часто залежить від індивідуальних уподобань, вимог до проекту та наявного набору навичок команди розробників. Обидва фреймворки мають свої сильні та слабкі сторони, і вони продовжують розвиватися з новими функціями та вдосконаленнями з часом. Тому модна використовувати два варіанти.

Функція F_3:

Google Colab і PyCharm є популярними інструментами для програмування та розробки, але вони служать різним цілям і мають відмінні функції. Ось коротке порівняння:

Середовище та налаштування: Google Colab — це онлайн-платформа, яка забезпечує середовище Jupyter Notebook у хмарі. Він не потребує локального встановлення та підтримує спільну роботу, що робить його зручним для швидкого створення прототипів і спільного використання коду. З іншого боку, PyCharm — це повнофункціональне інтегроване середовище розробки (IDE), яке потрібно інсталювати локально на вашій машині. Він забезпечує більш комплексне середовище розробки з розширеними функціями для великомасштабних проектів.

PyCharm дозволяє запускати код безпосередньо на вашій локальній машині, використовуючи її обчислювальні ресурси. Він забезпечує ефективні можливості налагодження та виконання для локальної розробки. Google Colab виконує код на віддалених серверах, використовуючи інфраструктуру Google. Це може бути корисним під час роботи з великими наборами даних або інтенсивних обчислювальних завдань, які вимагають більше ресурсів, ніж доступно на локальній машині.

Для виконання навчання моделі визначення пошкоджених будинків на супутникових зображень потрібно багато обчислювальних ресурсів, саме тому було використано варіант А - Google Colab.

Таким чином маємо наступний план реалізації ПП:

$F_1a – F_2a – F_3a$

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3. Обгрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для обчислень та збереження даних;
- X3 – час навчання даних;
- X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	40	80	120
Об’єм пам’яті	X2	Мб	50	25	10

Час попередньої обробки даних	X3	мс	100	80	60
Потенційний об’єм програмного коду	X4	кількість рядків коду	1500	1000	800

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

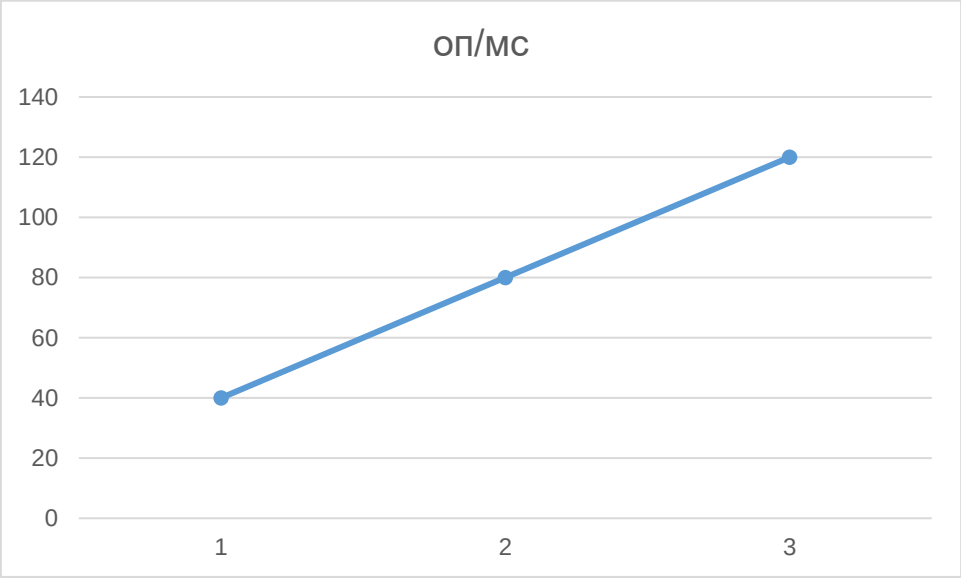

Рисунок 4.2 – X1, швидкодія мови програмування

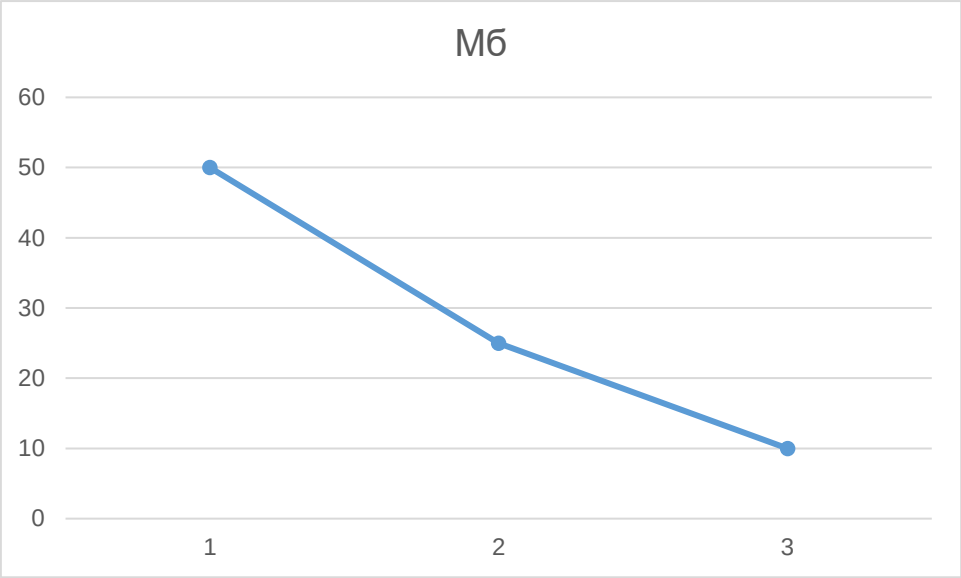

Рисунок 4.3 – X2, об’єм пам’яті

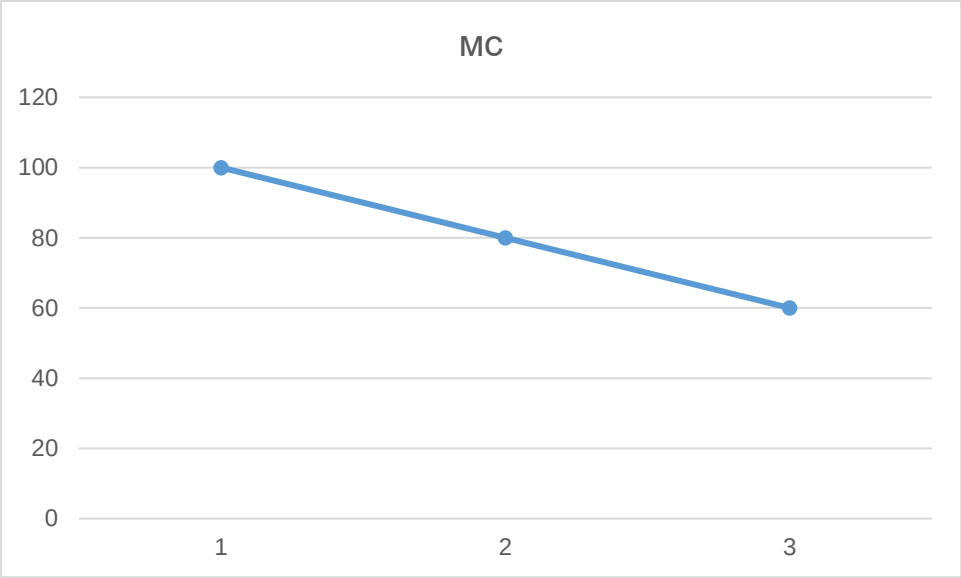

Рисунок 4.4 – X3, час попередньої обробки даних

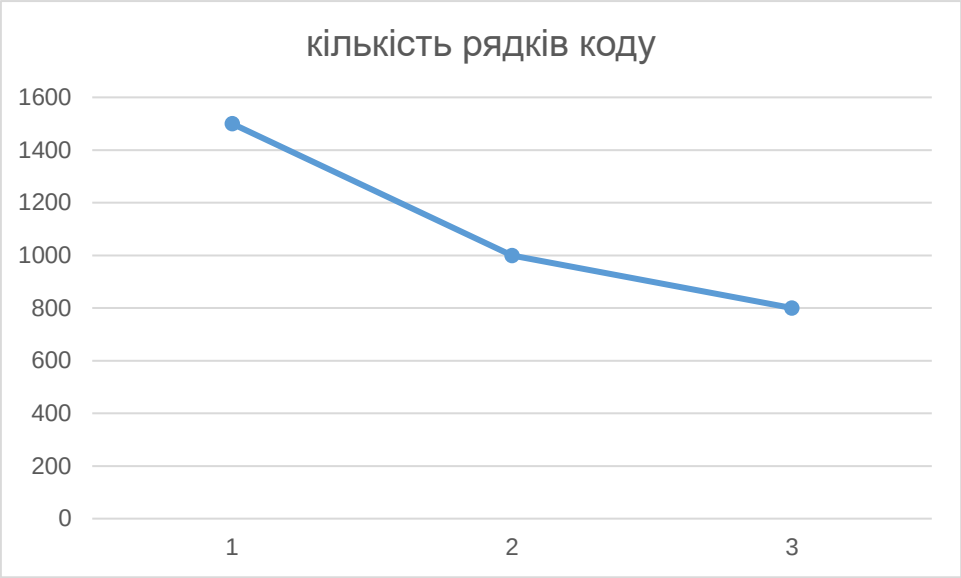

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порів¬няння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

параметр	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	2	2	1	1	11	-6.5	42.25
X2	Об’єм пам’яті	Mb	3	3	1	4	3	3	3	20	2.5	6.25
X3	Час попередньої обробки даних	мс	2	1	3	1	1	2	2	12	-5.5	30.25

X4	Потенційний об'єм програмного коду	Кількість рядків коду	4	4	4	3	4	4	4	27	9.5	90.25
	Разом		10	10	10	10	10	10	10	70	0	169

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \tag{4.1}$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 169. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 169}{7^2(4^3 - 4)} = 0{,}6897 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	>	<	<	<	<	<	0,5
X1 і X3	<	>	>	>	>	<	<	>	1,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	<	>	>	>	>	>	1,5
X2 і X4	<	<	<	>	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1	0,5	1,5	0,5	3,5	0,22	12,25	0,21	44,88	0,21
X2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,13	0,27
X3	0,5	0,5	1	0,5	2,5	0,16	9,25	0,16	34,13	0,16
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,36	77,88	0,36
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \,, \tag{4.11}$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	90	3	0,21	0,63
F2	А	X2	30	9	0,27	2,43
	Б	X3	80	2	0,16	0,32
F3	А	X4	1000	10	0,36	3,6

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0{,}63 + 2{,}43 + 3{,}6 = 6{,}66\ ;$$

$$K_{K2} = 0{,}63 + 0{,}32 + 3{,}6 = 4{,}55.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 70 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для

всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 70 \cdot 1{,}8 \cdot 0{,}9 = 113{,}4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (113.4 + 20.88 + 4.8 + 20.88) \cdot 8 = 1280 \text{ людино-годин.}$$

$$T_{II} = (113.4 + 20.88 + 6.91 + 20.88) \cdot 8 = 1297 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн., один аналітик в області даних з окладом 23000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.,} \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{20000 + 20000 + 23000}{3 \cdot 21 \cdot 8} = 125 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{ЗП} = C_ч \cdot T_i \cdot K_Д, \quad (4.16)$$

де $C_ч$– величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_{ЗП} = 125 \cdot 1280 \cdot 1.2 = 192000$ грн.

II. $C_{ЗП} = 125 \cdot 1297 \cdot 1.2 = 194550$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_{ВІД} = C_{ЗП} \cdot 0.22 = 192000 \cdot 0.22 = 42240$ грн.

II. $C_{ВІД} = C_{ЗП} \cdot 0.22 = 194550 \cdot 0.22 = 42801$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($C_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$C_Г = 12 \cdot М \cdot K_З = 12 \cdot 20000 \cdot 0,2 = 48000$ грн.

З урахуванням додаткової заробітної плати:

$C_{ЗП} = C_Г \cdot (1 + K_З) = 48000 \cdot (1 + 0.2) = 57600$ грн.

Відрахування на соціальний внесок:

$C_{ВІД} = C_{ЗП} \cdot 0.22 = 57600 \cdot 0,22 = 12672$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$C_А = K_{ТМ} \cdot K_А \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 10000 = 3000$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 10000 \cdot 0.1 = 1200 \quad \text{грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.8 =$$

$$= 1491 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491 \cdot 0,2 \cdot 0,3 \cdot 4,87 = 435,7 \quad \text{грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \quad грн.$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$$С_{ЕКС} = 57600 + 12672 + 3000 + 1200 + 435,7 + 6700 = 81607,7 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 81607,7 / 1491 = 54.73 \quad грн/год.$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \qquad (4.18)$$

I. $С_М = 54,73 \cdot 1280 = 70054,4$ грн.

II. $С_М = 54,73 \cdot 1297 = 70984,8$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \qquad (4.19)$$

I. $С_Н = 192000 \cdot 0,67 = 128640$ грн.

II. $С_Н = 194550 \cdot 0,67 = 130348,5$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \qquad (4.20)$$

I. $С_{ПП} = 192000 + 42240 + 70054,4 + 128640 = 432\,934,4$ грн.

II. $С_{ПП} = 194550 + 42801 + 70984,8 + 130348,5 = 438\,684,3$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 6,66 \ / \ 432\ 934,4 = 1,54 \cdot 10^{-5},$$

$$К_{ТЕР2} = 4,55 \ / \ 438\ 684,3 = 1,04 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,54 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,54 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Мова програмування – Python;
- Фреймворк для глибокого та машинного навчання – TensorFlow;
- Google Colab як інструмент для програмування.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми, доступний функціонал та забезпечення оптимальною обчислювальною потужністю для роботи.

ВИСНОВКИ

В ході виконання дипломної роботи було виконано дослідження моделей для розпізнавання пошкоджених будинків на супутникових знімках. Досліджено існуючі архітектури виявлення об’єктів. А також було запропонована власна архітектура, що включає в себе моделі виявлення для локалізації будівель і згорткову мережу на основі Siamese моделі для класифікації пошкоджень. Ця двоетапна навчальна модель, що поєднує YOLO для виявлення будівель і сіамську мережу для класифікації пошкоджень, пропонує потужний підхід для виявлення пошкоджень будівель із супутникових зображень. Підвищена точність, швидкість, ефективність і можливість узагальнення моделі роблять її цінним інструментом у реагуванні на катастрофи, гуманітарних зусиллях і міському плануванні. Використовуючи сильні сторони YOLO та сіамської мережі, двоетапна модель демонструє свою ефективність у точному визначенні та класифікації пошкоджених будівель.

Використали методологія функціонально-вартісного аналізу, щоб вибрати найбільш оптимальний підхід до впровадження, було проведено ретельний аналіз різних варіантів, враховуючи як економічні фактори, так і характеристики продукту, які безпосередньо впливають на ефективність роботи та сумісність апаратного забезпечення.

Майбутні дослідження мають бути зосереджені на подальшому вдосконаленні архітектури моделі, вивченні методів навчання передачі та інтеграції мультимодальних джерел даних для покращення можливостей виявлення пошкоджень будівель.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Online fire map. Fire Information for Resource Management System. URL: https://firms.modaps.eosdis.nasa.gov/map/#d:24hrs;@36.3,46.4,6z
2. Daksh B. History of CNN & its impact in the field of Artificial Intelligence [Електронний ресурс] / Bhatnagar Daksh // Medium. – 2 січня 2023. – URL: https://medium.com/international-school-of-ai-data-science/history-of-cnn-its-impact-in-the-field-of-artificial-intelligence-2b1efb7d99e5.
3. Object Detection Guide [Електронний ресурс] // FRITZ LABS INCORPORATED. – 2021. – Режим доступу до ресурсу: https://www.fritz.ai/object-detection/.
4. Daniel S. Зображення пташки // Unsplash. - 1 квітня 2023. - URL: https://unsplash.com/photos/TIVnungUUG0
5. Image segmentation. URL: https://www.tensorflow.org/tutorials/images/segmentation
6. Viola–Jones object detection framework. // Wikipedia. - 14 квітня 2023. - URL: https://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework
7. Scale Invariant Feature Transform. // Scolarpedia. - 5 серпня 2012. - URL: http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform
8. Mallick S. Histogram of Oriented Gradients explained using OpenCV [Електронний ресурс] / Satya Mallick // LearnOpenCV. – 6 грудня 2016. – URL: https://learnopencv.com/histogram-of-oriented-gradients/.
9. Rizzoli A. The Ultimate Guide to Object Detection [Електронний ресурс] / Alberto Rizzoli // V7. – 2021. – URL: https://www.v7labs.com/blog/object-detection-guide.

10. Boesch G. Object Detection in 2023: The Definitive Guide [Електронний ресурс] / Gaudenz Boesch // Viso.ai – URL: https://viso.ai/deep-learning/object-detection/.
11. Adith N. T. Single Shot Detector (SSD) + Architecture of SSD. // OpenGenus. URL: https://iq.opengenus.org/single-shot-detector/
12. Wei L., Dragomir A., Dumitru E., Christian S., Scott R., Cheng-Yang F., Alexander C. B. SSD: Single Shot MultiBox Detector. [Електронний ресурс] / Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg // arXiv. - 29 грудня 2016. - URL: https://doi.org/10.48550/arXiv.1512.02325
13. Sanchez S. A. A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework [Електронний ресурс] / S. A. Sanchez, A. D. Morales, H. J. Romero // IOPScience. – 2020. – Режим доступу до ресурсу: http://dx.doi.org/10.1088/1757-899X/844/1/012024.
14. Sovit R. YOLOv8 Ultralytics: State-of-the-Art YOLO Models. LearnOpenCV. 10 січня 2023. URL: https://learnopencv.com/ultralytics-yolov8/
15. Siamese neural networks for the classification of high-dimensional radiomic features [Електронний ресурс] / M.Abhishaike, J. Dormer, Q. Li, D. Chen // National Library of Medicine. – 2020. – Режим доступу до ресурсу: https://doi.org/10.1117%2F12.2549389.
16. Ritwik G., Richard H., Sandra S., Nirav P., Bryce G., Jigar D., Eric H., Howie C., Matthew G., xBD: A Dataset for Assessing Building Damage from Satellite Imagery. 21 лютого 2019. URL: https://arxiv.org/pdf/1911.09296.pdf
17. Sanket C. P., Prof. Pinal S. Survey on Different Object Detection and Segmentation Methods. International Journal of Innovative Science and Research Technology. 1 січня 2021. URl: https://ijisrt.com/assets/upload/files/IJISRT21JAN357.pdf
18. Maxar Technologies. URL: https://www.maxar.com/
19. Roboflow. URL: https://roboflow.com/

20. ELGENDY M. Deep Learning for Vision Systems / MOHAMED ELGENDY. – NY: Manning Publications Co., 2020. – 480 с.
21. Building Damage Detection in Satellite Imagery Using Convolutional Neural Networks [Електронний ресурс] / Z. X.Joseph, L. Wenhan, L. Zebo, V. Zaytseva // arXiv. – 2019. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.1910.06444.
22. Allwright S. How to interpret AUC score (simply explained) [Електронний ресурс] / Stephen Allwright // Stephen Allwright. – 2022. – Режим доступу до ресурсу: https://stephenallwright.com/interpret-auc-score/.
23. Khvedchenya E. Fully convolutional Siamese neural networks for buildings damage assessment from satellite images [Електронний ресурс] / E. Khvedchenya, T. Gabruseva // arXiv. – 2021. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.2111.00508.
24. Mehmet F. D. Post-Disaster Building Damage Assessment [Електронний ресурс] / F. D. Mehmet, M. Dutson, S. S. Shri. – 2022. – Режим доступу до ресурсу: https://github.com/mattdutson/xview2.
25. BDANet: Multiscale Convolutional Neural Network with Cross-directional Attention for Building Damage Assessment from Satellite Images [Електронний ресурс] / Y.Shen, S. Zhu, T. Yang, C. Chen // arXiv. – 2021. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.2105.07364.

ДОДАТОК А. ВИХІДНИЙ КОД ДЛЯ ВИЯВЛЕННЯ БУДІВЕЛЬ ЗА ДОПОМОГОЮ YOLO

```
!pip install ultralytics
```

```
import os
import glob
import numpy as np
import pandas as pd
import random
from pathlib import Path
import tqdm.notebook

import cv2
import torch
from osgeo import gdal
import matplotlib.pyplot as plt
from PIL import Image
from IPython import display
from sklearn.model_selection import train_test_split

import yaml
from ultralytics import YOLO
```

```
# checking availability of GPU resources
print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else
'CPU'})")
```

```
base_path = "/content/drive/MyDrive"
folder_name = "Colab Notebooks/DamageBuilding"
df = pd.read_csv(f'{base_path}/{folder_name}/annotations.csv')
df
```

```
df.info()
```

```
def getBounds(bounds):
    try:
        bounds = tuple(map(int, bounds[1:-1].split(', ')))
        return bounds
    except:
        return np.nan

def getWidth(bounds):
    try:
```

```
        (xmin, ymin, xmax, ymax) = bounds
        return np.abs(xmax - xmin)
    except:
        return np.nan

def getHeight(bounds):
    try:
        (xmin, ymin, xmax, ymax) = bounds
        return np.abs(ymax - ymin)
    except:
        return np.nan

# Create bounds, width and height
df.loc[:,'bounds'] = df.loc[:,'bounds'].apply(getBounds)
df.loc[:,'width'] = df.loc[:,'bounds'].apply(getWidth)
df.loc[:,'height'] = df.loc[:,'bounds'].apply(getHeight)
df
```

```
folder_img_name = "Colab Data/DamageBuilding"

image_path = f"{base_path}/{folder_img_name}/images"
pre_image = sorted(glob.glob(os.path.join(image_path, "*pre*")))
post_image = sorted(glob.glob(os.path.join(image_path, "*post*")))
```

```
print(f"Found {len(post_image)} images files in {folder_name}")
pickone = random.choice(post_image)
img = Image.open(pickone)
IMAGE_HEIGHT, IMAGE_WIDTH = img.size
num_channels = len(img.mode)
print("Image size: {}".format((IMAGE_HEIGHT, IMAGE_WIDTH)))
print("Num channels: {}".format(num_channels))
```

```
display.Image(pickone)
```

```
IMAGE_HEIGHT, IMAGE_WIDTH = 1024, 1024
TILE_WIDTH = 512
TILE_HEIGHT = 512
TRUNCATED_PERCENT = 0.3
_overwriteFiles = True

TILES_DIR = {'train': Path('/content/drive/MyDrive/Colab
Notebooks/DamageBuilding/PostImages/train/images/'),
             'valid': Path('/content/drive/MyDrive/Colab
Notebooks/DamageBuilding/PostImages/valid/images/')}

for _, folder in TILES_DIR.items():
```

```
    if not os.path.isdir(folder):
        os.makedirs(folder)

LABELS_DIR = {'train': Path('/content/drive/MyDrive/Colab
Notebooks/DamageBuilding/PostImages/train/labels/'),
              'valid': Path('/content/drive/MyDrive/Colab
Notebooks/DamageBuilding/PostImages/valid/labels/')}

for _, folder in LABELS_DIR.items():
    if not os.path.isdir(folder):
        os.makedirs(folder)
```

```
damage_dict = {
    "no-damage": 0,
    "minor-damage": 1,
    "major-damage": 2,
    "destroyed": 3,
    "un-classified": 0
}
```

```
# Save one line in .txt file for each tag found inside the tile
def tag_is_inside_tile(bounds, x_start, y_start, damage, width, height,
truncated_percent):
    x_min, y_min, x_max, y_max = bounds
    x_min, y_min, x_max, y_max = x_min - x_start, y_min - y_start, x_max -
x_start, y_max - y_start

    if (x_min > width) or (x_max < 0.0) or (y_min > height) or (y_max < 0.0):
        return None

    x_max_trunc = min(x_max, width)
    x_min_trunc = max(x_min, 0)
    if (x_max_trunc - x_min_trunc) / (x_max - x_min) < truncated_percent:
        return None

    y_max_trunc = min(y_max, width)
    y_min_trunc = max(y_min, 0)
    if (y_max_trunc - y_min_trunc) / (y_max - y_min) < truncated_percent:
        return None

    x_center = (x_min_trunc + x_max_trunc) / 2.0 / width
    y_center = (y_min_trunc + y_max_trunc) / 2.0 / height
    x_extend = (x_max_trunc - x_min_trunc) / width
    y_extend = (y_max_trunc - y_min_trunc) / height

    return (damage_dict[damage], x_center, y_center, x_extend, y_extend)
```

```
train, valid = train_test_split(pre_image, train_size=0.8)
# train, test = train_test_split(train, train_size=0.95)

print("Train test:", len(train))
print("Valid test:", len(valid))
# print("Test test:", len(test))
```

```
def cut_images(images, folder='train', time='pre'):
  for img_path in tqdm.notebook.tqdm(images):
      np_img = np.array(Image.open(img_path, mode='r'), dtype=np.uint8)

      img_name = img_path.split('/')[-1]
      if time=='post':
        img_labels = df[df["image_id"] == img_name]
      else:
        img_labels = df[df["image_id"] == img_name.replace('pre', 'post')]

      X_TILES = IMAGE_WIDTH  // TILE_WIDTH
      Y_TILES = IMAGE_HEIGHT // TILE_HEIGHT

      for i, x in enumerate(range(X_TILES)):
          for j, y in enumerate(range(Y_TILES)):

              x_end = min((x + 1) * TILE_WIDTH, IMAGE_WIDTH)
              x_start = x_end - TILE_WIDTH
              y_end = min((y + 1) * TILE_HEIGHT, IMAGE_HEIGHT)
              y_start = y_end - TILE_HEIGHT

              # print(f"{x_start}_{x_end}_{y_start}_{y_end}")

              save_tile_path = TILES_DIR[folder].joinpath(img_name[:-13] + "_" +
str(i) + "_" + str(j) + ".jpg")
              save_label_path = LABELS_DIR[folder].joinpath(img_name[:-13] + "_"
+ str(i) + "_" + str(j) + ".txt")


              if time=='post':
                found_boxs = [
                    tag_is_inside_tile(bounds, x_start, y_start, damage,
TILE_WIDTH, TILE_HEIGHT, TRUNCATED_PERCENT)
                    for bounds, damage in zip(img_labels.bounds,
img_labels.damage_type)]
              else:
                damage = 'no-damage'
                found_boxs = [
                    tag_is_inside_tile(bounds, x_start, y_start, damage,
TILE_WIDTH, TILE_HEIGHT, TRUNCATED_PERCENT)
                    for bounds in img_labels.bounds]
```

```
                found_boxs = [box for box in found_boxs if box is not None]

                if found_boxs:
                  with open(save_label_path, 'w+') as f:
                    for tags in found_boxs:
                        f.write(' '.join(str(x) for x in tags) + '\n')

                  if _overwriteFiles or not os.path.isfile(save_tile_path):
                    cut_tile = np.zeros(shape=(TILE_WIDTH, TILE_HEIGHT, 3),
dtype=np.uint8)
                    cut_tile[0:TILE_HEIGHT, 0:TILE_WIDTH, :] =
np_img[y_start:y_end, x_start:x_end, :]
                    cut_tile_img = Image.fromarray(cut_tile)
                    cut_tile_img.save(save_tile_path)
```

```
cut_images(pre_image, time='pre')
cut_images(valid, folder='valid')
```

```
from pathlib import Path

base_path = "/content/drive/MyDrive/Colab Notebooks"

# Change the current working directory to the Google Drive directory
os.chdir(f'{base_path}')

# Change directory to cloned repository
folder_name = "DamageBuilding"
Path(f'{base_path}/{folder_name}').mkdir(parents=True, exist_ok=True)
os.chdir(f'{base_path}/{folder_name}')

!pwd

!ls
```

```
CONFIG = """names: ['roof']
nc: 1
train: train/images
val: valid/images
"""

with open("data.yaml", "w") as f:
    f.write(CONFIG)
```

```
print("yaml file parameters:\n")

with open('data.yaml') as stream:
```

```
    data_loaded = yaml.safe_load(stream)
data_loaded
```

```
model_name = 'yolov8s.pt'
data = f'{base_path}/{folder_name}/data.yaml'
epochs = 50
resume = False  # resume training from last checkpoint
imgsz = 512
optimizer = "Adam"
lr0 = 3E-04

model = YOLO(model_name)
results = model.train(data=data, epochs=epochs, resume=resume, imgsz=imgsz,
optimizer=optimizer, lr0=lr0)
```

```
results = model.val()
```

```
model = YOLO("runs/detect/train6/weights/best.pt")
```

```
def plot_random_images_with_boxes(image_file):
    f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize=(30, 15))

    image_path = os.path.join(image_folder, image_file)
    annotation_path = os.path.join(annotation_folder,
os.path.splitext(image_file)[0] + ".txt")

    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    with open(annotation_path, "r") as f:
        annotations = f.readlines()

    ax1.imshow(image)
    ax1.set_title('Image', fontsize=24)

    for annotation in annotations:
        class_id, x, y, width, height = map(float, annotation.split())
        left = int((x - width / 2) * image.shape[1])
        top = int((y - height / 2) * image.shape[0])
        right = int((x + width / 2) * image.shape[1])
        bottom = int((y + height / 2) * image.shape[0])

        cv2.rectangle(image, (left, top), (right, bottom), (255, 0, 0), 2)

    ax2.imshow(image)
    ax2.set_title('True label', fontsize=24)
```

```
    pred_image = cv2.imread(f'runs/detect/predict/1/{image_file}')
    pred_image = cv2.cvtColor(pred_image, cv2.COLOR_BGR2RGB)
    ax3.imshow(pred_image)
    ax3.set_title('Predict label', fontsize=24)

    ax1.axis('off')
    ax2.axis('off')
    ax3.axis('off')
    plt.show()
```

```
# Example usage
image_folder = 'valid/images'
annotation_folder = 'valid/labels'
num_images = 5

image_files = os.listdir(image_folder)
random.shuffle(image_files)
image_files = image_files[:num_images]

for image_file in image_files:
  results = model.predict(source=f'valid/images/{image_file}', name='predict/1',
show_labels=False, save=True, retina_masks=True)
  plot_random_images_with_boxes(image_file)
```

ДОТАТОК Б. ВИХІДНИЙ КОД ДЛЯ КЛАСИФІКАЦІЇ ПОШКОДЖЕНЬ БУДІВЕЛЬ ЗА ДОПОМОГОЮ CNN

```
!pip install tensorflow-addons

import os
import glob
import pickle
import tqdm.notebook
import datetime
from pathlib import Path

import cv2
import numpy as np
import pandas
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers, Input
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image_dataset_from_directory
from keras.layers import concatenate
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
import tensorflow_addons as tfa
```

```
base_path = "/content/drive/MyDrive/Colab Notebooks"
folder_name = "DamageBuilding"

damage_dict = {
    0: "no-damage",
    1: "minor-damage",
    2: "major-damage",
    3: "destroyed",
}
```

```
images_dir = f"{base_path}/{folder_name}/PostImages/train/images"
labels_dir = f"{base_path}/{folder_name}/PostImages/train/labels"

# pre_images = sorted(glob.glob(os.path.join(images_dir, "*pre*")))
post_images = sorted(glob.glob(os.path.join(images_dir, "*post*")))

# pre_label = sorted(glob.glob(os.path.join(labels_dir, "*pre*")))
# post_label = sorted(glob.glob(os.path.join(labels_dir, "*post*")))

# print("Pre images: ", len(pre_images))
print("Post images: ", len(post_images))
```

```
# print("Pre labels: ", len(pre_label))
# print("Post labels: ", len(post_label))
```

```
output_pre_dir = f"{base_path}/{folder_name}/clasify/pre"

if not os.path.exists(output_pre_dir):
    os.makedirs(output_pre_dir)

output_post_dir = f"{base_path}/{folder_name}/clasify/post"

if not os.path.exists(output_post_dir):
    os.makedirs(output_post_dir)
```

```
for image_file in tqdm.notebook.tqdm(post_images[5500:]):
    image_post = cv2.imread(image_file)
    image_pre = cv2.imread(image_file.replace('post','pre'))

    label_file = image_file.replace('images','labels').replace('jpg','txt')

    img_file = image_file.split("/")[-1]
    with open(label_file, "r") as f:
        lines = f.readlines()

    for i, line in enumerate(lines):
        class_id, center_x, center_y, width, height = map(float, line.split())

        x = int((center_x - width / 2) * image_post.shape[1])
        y = int((center_y - height / 2) * image_post.shape[0])
        w = int(width * image_post.shape[1])
        h = int(height * image_post.shape[0])

        cropped_image_post = image_post[y:y+h, x:x+w]
        cropped_image_pre = image_pre[y:y+h, x:x+w]

        class_name = damage_dict[int(class_id)]

        class_post_dir = os.path.join(output_post_dir, class_name)
        class_pre_dir = os.path.join(output_pre_dir, class_name)

        if not os.path.exists(class_post_dir):
            os.makedirs(class_post_dir)
        if not os.path.exists(class_pre_dir):
            os.makedirs(class_pre_dir)

        output_path = os.path.join(class_post_dir, f"{img_file[:-4]}_{i}.jpg")
        cv2.imwrite(output_path, cropped_image_post)
        cv2.imwrite(output_path.replace('post','pre'), cropped_image_pre)
```

```
delete=[]
damage=2
images_dir = f"{base_path}/{folder_name}/clasify/post/{damage_dict[damage]}/*"
post_images = glob.glob(images_dir)
print(len(post_images))

images_dir = f"{base_path}/{folder_name}/clasify/pre/{damage_dict[damage]}/*"
pre_images = glob.glob(images_dir)
print(len(pre_images))

for im in post_images:
  if im.replace('post', 'pre') not in pre_images:
    delete.append(im)

for im in pre_images:
  if im.replace('pre','post') not in post_images:
    delete.append(im)
```

```
for d in delete:
  os.remove(d)
```

```
image_cache = {}
```

```
def load_image(image_name):
  image = cv2.imread(image_name).astype(np.float32)
  image = cv2.resize(image, (64, 64))
  image = np.divide(image, 256)
  return image

def generate_image_triples_batch(image_triples, batch_size, shuffle=False):
    while True:
        n = len(image_triples)

        if shuffle:
            indices = np.random.permutation(np.arange(n))
        else:
            indices = np.arange(n)

        shuffled_triples = [image_triples[ix] for ix in indices]
        num_batches = len(shuffled_triples) // batch_size

        for bid in range(num_batches):
            images_left, images_right, labels = [], [], []
            batch = shuffled_triples[bid * batch_size : (bid + 1) * batch_size]

            for i in range(batch_size):
                lhs, rhs, label = batch[i]
                # print(lhs)
```

```
                if lhs in image_cache:
                  l_image, r_image = image_cache[lhs]
                  images_left.append(l_image)
                  images_right.append(r_image)

                else:
                  l_image = load_image(lhs)
                  r_image = load_image(rhs)
                  images_left.append(l_image)
                  images_right.append(r_image)
                  image_cache[lhs] = [l_image, r_image]

                labels.append(label)

            left = np.array(images_left)
            right = np.array(images_right)
            Y = np_utils.to_categorical(np.array(labels), num_classes=4)
            yield ([left, right], Y)

triples = []

for damage in range(4):
  images_dir = f"{base_path}/{folder_name}/clasify/pre/{damage_dict[damage]}/*"
  pre_images = glob.glob(images_dir)
  print(f"{damage_dict[damage]} pre images:  {len(pre_images)}")

  images_dir = f"{base_path}/{folder_name}/clasify/post/{damage_dict[damage]}/*"
  post_images = glob.glob(images_dir)
  print(f"{damage_dict[damage]} post images:  {len(post_images)}\n")

  trip = np.column_stack((pre_images, post_images, [damage]*len(post_images)))

  triples.extend(trip)
```

```
len(triples)
triples[0]

file_cache_name = f"{base_path}/{folder_name}/clasify/image_cache.pkl"

with open(file_cache_name, 'rb') as fp:
    image_cache = pickle.load(fp)
    print('Image cache dictionary load')

triples_train, triples_valid = train_test_split(triples, train_size=0.8)

BATCH_SIZE = 16

train_gen = generate_image_triples_batch(triples_train, BATCH_SIZE,
shuffle=True)
```

```
val_gen = generate_image_triples_batch(triples_valid, BATCH_SIZE, shuffle=False)

num_train_steps = len(triples_train) // BATCH_SIZE
num_val_steps = len(triples_valid) // BATCH_SIZE
```

```
%%timeit
lhs, rhs, label = triples[0]
l_image = load_image(lhs)
r_image = load_image(rhs)
```

```
%%timeit
l_image, r_image = image_cache[triples[0][0]]
```

```
callbacks_list = []

accuracy_name = "categorical_accuracy"
val_acc_name = f"val_accuracy"

timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

models_dir = f"{base_path}/{folder_name}/clasify/model-checkpts/{timestamp}"
Path(models_dir).mkdir(parents=True, exist_ok=True)

model_filename = os.path.join(models_dir, f'model_{timestamp}_'+
f'{{epoch:03d}}_{{{val_acc_name}:.4f}}.h5')


log_dir = f"{base_path}/{folder_name}/clasify/logs/log-{timestamp}"
Path(log_dir).mkdir(parents=True, exist_ok=True)

callbacks_list.append([
                        tf.keras.callbacks.ModelCheckpoint(model_filename,
                                        monitor=val_acc_name,
                                        mode="max",
                                        save_best_only=True
                                        ),
                        tf.keras.callbacks.ReduceLROnPlateau(), # reduce learning
rate when validation loss plateaus
                        tf.keras.callbacks.EarlyStopping(monitor=val_acc_name,
patience=50),
                        tf.keras.callbacks.TensorBoard(log_dir, histogram_freq=1,
update_freq='batch')
])

print(f"A list of callbacks: \n{callbacks_list}")
```

```
def plot_history(history):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 8))

  # Plot training & validation loss values
  ax1.plot(history.history['loss'])
  ax1.plot(history.history['val_loss'])
  ax1.set_title('Model loss')
  ax1.grid()
  ax1.set_ylabel('Loss')
  ax1.set_xlabel('Epoch')
  ax1.legend(['Train', 'Validation'], loc='upper left')

  # Plot training & validation accuracy values
  ax2.plot(history.history['accuracy'])
  ax2.plot(history.history['val_accuracy'])

  ax2.plot(history.history['auc'])
  ax2.plot(history.history['val_auc'])

  ax2.set_title('Model accuracy')
  ax2.grid()
  ax2.set_ylabel('Accuracy')
  ax2.set_xlabel('Epoch')
  ax2.legend(['Train accuracy', 'Validation accuracy', 'Train auc', 'Validation
auc'], loc='lower right')
  plt.show()
```

```
base_model = VGG16(weights='imagenet', input_shape=(46, 46, 3), pooling='max',
include_top = False)

base_model.summary()

input_image_a = Input(shape=(46, 46, 3), name = 'input_image_a')
input_image_b = Input(shape=(46, 46, 3), name = 'input_image_b')

feat_image_a = base_model(input_image_a)
feat_image_b = base_model(input_image_b)

combined_features = concatenate([feat_image_a, feat_image_b], name =
'merge_features')

combined_features = layers.Dense(512, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(256, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(128, activation = 'linear')(combined_features)
```

```
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(64, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(32, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(16, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(8, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(4, activation = 'softmax')(combined_features)
similarity_model = tf.keras.Model(inputs = [input_image_a, input_image_b],
outputs = [combined_features], name = 'Similarity_Model')
similarity_model.summary()

tf.keras.utils.plot_model(similarity_model, show_shapes=True)

similarity_model.compile(loss="categorical_crossentropy",
                         optimizer="adam",
                         metrics=['accuracy',
                                  tfa.metrics.F1Score(num_classes=4,
average='macro', threshold=0.5)
                                  ])

history = similarity_model.fit_generator(train_gen,
                                         steps_per_epoch=num_train_steps,
                                         epochs=30,
                                         validation_data=val_gen,
                                         validation_steps=num_val_steps,
                                         callbacks=callbacks_list)

val_acc = history.history[f"val_accuracy"]
best_epoch = np.argmax(val_acc)
best_value = val_acc[best_epoch]
print(f'The best epoch: {best_epoch + 1}, and its the best accuracy value:
{100*best_value:.2f}%')

val_f1 = history.history[f"val_f1_score"]
best_epoch = np.argmax(val_f1)
best_value = val_f1[best_epoch]
```

```
print(f'The best epoch: {best_epoch + 1}, and its the best auc value:
{100*best_value:.2f}%')

plot_history(history)


base_model = ResNet50(weights='imagenet', input_shape=(64, 64, 3),
pooling='max', include_top = False)

base_model.summary()
```

```
input_image_a = Input(shape=(64, 64, 3), name = 'input_image_a')
input_image_b = Input(shape=(64, 64, 3), name = 'input_image_b')

feat_image_a = base_model(input_image_a)
feat_image_b = base_model(input_image_b)

combined_features = concatenate([feat_image_a, feat_image_b], name =
'merge_features')

combined_features = layers.Dense(512, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(256, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(128, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(64, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(32, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(16, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(8, activation = 'linear')(combined_features)
combined_features = layers.BatchNormalization()(combined_features)
combined_features = layers.Activation('relu')(combined_features)

combined_features = layers.Dense(4, activation = 'softmax')(combined_features)
```

```
similarity_model_resnet = tf.keras.Model(inputs = [input_image_a,
input_image_b], outputs = [combined_features], name = 'Similarity_Model')
similarity_model_resnet.summary()
```

```
similarity_model_resnet.compile(loss="categorical_crossentropy",
                          optimizer="adam",
                          metrics=[tf.keras.metrics.AUC(), "accuracy"])

history_resnet = similarity_model_resnet.fit_generator(train_gen,
                                          steps_per_epoch=num_train_steps,
                                          epochs=30,
                                          validation_data=val_gen,
                                          validation_steps=num_val_steps,
                                          callbacks=callbacks_list)
```

```
val_acc = history_resnet.history[f"val_accuracy"]
best_epoch = np.argmax(val_acc)
best_value = val_acc[best_epoch]
print(f'The best epoch: {best_epoch + 1}, and its the best accuracy value:
{100*best_value:.2f}%')

val_auc = history_resnet.history[f"val_auc_3"]
best_epoch = np.argmax(val_auc)
best_value = val_auc[best_epoch]
print(f'The best epoch: {best_epoch + 1}, and its the best auc value:
{100*best_value:.2f}%')

plot_history(history_resnet)
```

ДОТАТОК В. КОД КЛАСИФІКАТОРА ІЗ ВИЗНАЧЕННЯМ ОБ'ЄКТІВ

```
!pip install ultralytics
```

```
import os
import matplotlib.pyplot as plt
import random
import glob

import cv2
import numpy as np
from ultralytics import YOLO

import torch
import tensorflow as tf

# checking availability of GPU resources
print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else
'CPU'})")

from pathlib import Path

base_path = "/content/drive/MyDrive/Colab Notebooks"

os.chdir(f'{base_path}')

folder_name = "DamageBuilding"
Path(f'{base_path}/{folder_name}').mkdir(parents=True, exist_ok=True)
os.chdir(f'{base_path}/{folder_name}')

!pwd

!ls

detect_model = YOLO("runs/detect/train6/weights/best.pt")
classify_name = "20230608-062743/model_20230608-062743_014_0.7642.h5"
MODEL_PATH = f"clasify/model-checkpts/{classify_name}"
classify_model = tf.keras.models.load_model(MODEL_PATH)

def class_predict(pre_image , post_image):
  pre_image = tf.image.resize(pre_image, (64, 64))[None, :]
  pre_image = tf.image.per_image_standardization(pre_image)

  post_image = tf.image.resize(post_image, (64, 64))[None, :]
  post_image = tf.image.per_image_standardization(post_image)

  pred = classify_model.predict([pre_image, post_image])
  return np.argmax(pred)
```

```
# Color codes for bounding boxs
damage_dict = {
    0: (0, 255, 0),
    1:(255, 255, 102),
    2: (255, 153, 51),
    3: (255, 0, 0)
}

def plot_images_with_boxes(image_file, result):
    f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, sharey=True, figsize=(30, 15))

    annotation_path = image_file.replace('images', 'labels').replace('jpg',
'txt')
    print(annotation_path)
    pre_image = image_file.replace('post', 'pre')

    image = cv2.imread(image_file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    pre_image = cv2.imread(pre_image)
    pre_image = cv2.cvtColor(pre_image, cv2.COLOR_BGR2RGB)

    with open(annotation_path, "r") as f:
        annotations = f.readlines()

    ax1.imshow(pre_image)
    ax1.set_title('Pre image', fontsize=24)

    ax2.imshow(image)
    ax2.set_title('Post image', fontsize=24)

    image_predict = image.copy()

    for annotation in annotations:
        class_id, x, y, width, height = map(float, annotation.split())
        left = int((x - width / 2) * image.shape[1])
        top = int((y - height / 2) * image.shape[0])
        right = int((x + width / 2) * image.shape[1])
        bottom = int((y + height / 2) * image.shape[0])

        cv2.rectangle(image, (left, top), (right, bottom),
damage_dict[int(class_id)], 2)

    ax3.imshow(image)
    ax3.set_title('True label', fontsize=24)

    for box in result[0].boxes:
      left, bottom, right, top = map(int, box.xyxy[0])
      # print(f"{right-left}, {top-bottom} px")
```

```
      class_id = class_predict(pre_image[left:right, bottom:top],
image[left:right, bottom:top])

      cv2.rectangle(image_predict, (left, bottom), (right, top),
damage_dict[int(class_id)], 2)

    ax4.imshow(image_predict)
    ax4.set_title('Predict label', fontsize=24)

    ax1.axis('off')
    ax2.axis('off')
    ax3.axis('off')
    ax4.axis('off')
    plt.show()

image_path = 'PostImages/images'
label_path = 'PostImages/labels'

pre_images = sorted(glob.glob(os.path.join(image_path, "*pre*")))
post_images = sorted(glob.glob(os.path.join(image_path, "*post*")))

# pre_label = sorted(glob.glob(os.path.join(label_path, "*pre*")))
# post_label = sorted(glob.glob(os.path.join(label_path, "*post*")))

num_images = 5

random.shuffle(post_images)
image_files = post_images[:num_images]

for image_file in image_files:
  result = detect_model.predict(source=image_file, show_labels=False, save=True,
retina_masks=True)
  plot_images_with_boxes(image_file, result)
```