МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА РОЗПІЗНАВАННЯ ПОШКОДЖЕНИХ БУДІВЕЛЬ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Спеціальність 122 «Комп'ютерні науки»

122 – КРМ – 601.21810308

Виконав студент 6-го курсу, групи 601
_____ В. В. Димо
« » лютого 2024 р.

Консультант: д-р техн. наук, професор
_____ О. П. Гожий
« » лютого 2024 р.

Миколаїв – 2024

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20__р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Димо Валерію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Інтелектуальна система розпізнавання пошкоджених будівель на основі нейронних мереж».

Керівник роботи Гожий Олександр Петрович, д-р техн. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «20» жовтня 2023 р. № 201

2. Строк представлення кваліфікаційної роботи студентом «20» лютого 2024 р.

3. Вхідні (початкові) дані до роботи: набір даних, що містить зображення поверхні землі з будівлями (супутникові знімки, або фотографії з безпілотних літальних апаратів); сегментаційні маски до них.

Очікуваний результат: система розпізнавання (семантична сегментація) пошкоджених будівель на знімках поверхні землі.

4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

– аналіз сучасного стану попередньої оцінки руйнувань та розпізнавання пошкоджених будівель на знімках;
– огляд існуючих методів та алгоритмів штучного інтелекту;
– технологічні та програмні рішення побудови, навчання та тестування

– алгоритмів машинного навчання;

– реалізація та побудова нейронної мережі для розпізнавання (семантичної сегментації);

– аналіз результатів застосування обраних методів машинного навчання для вирішення поставленої задачі;

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Оптимізація роботи та поведінка в умовах надзвичайних ситуацій воєнного характеру»

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	д-р біол. наук, професор Григор'єва Л. І.	
Методична частина	д-р техн. наук, професор Гожий О. П.	

Керівник роботи _____д-р техн. наук, проф. Гожий О. П._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання _____Димо В. В.____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_31_» __жовтня___ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи магістра

Тема: Інтелектуальна система розпізнавання пошкоджених будівель на основі нейронних мереж

	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника. Подання заяви на затвердження теми КРМ	01.09.2023	10.10.2023	Виконано
2	Отримання завдання на виконання	11.10.2023	31.10.2023	Виконано
3	Складання календарного плану КРМ	02.10.2023	10.11.2023	Виконано
4	Збір матеріалів та огляд літератури за досліджуваною темою	10.11.2023	01.12.2023	Виконано
5	Проходження переддипломної практики: огляд існуючих архітектур згорткових мереж	27.11.2023	23.12.2023	Виконано
6	Опис фахової частини, зокрема визначення архітектури нейронної мережі, збір та аналіз даних. Реалізація мережі U-Net. Створення датасету та навчання нейронної мережі	01.12.2023	01.01.2024	Виконано
7	Розробка спеціальної частини з охорони праці	01.01.2023	18.01.2024	Виконано
8	Розробка методичної частини КРМ	10.01.2024	10.02.2024	Виконано
9	Перший попередній захист КРМ	29.01.2024	29.01.2024	Виконано
10	Корегування та доробка результатів роботи згідно попереднього захисту	30.01.2024	11.02.2024	Виконано
11	Другий попередній захист КРМ	12.02.2024	12.02.2024	Виконано
12	Кінцеве оформлення пояснювальної записки та презентації доповіді, подання КРМ рецензентові	13.02.2024	14.02.2024	Виконано
15	Подання КРМ, її електронної копії та інших документів до захисту	19.02.2024	19.02.2024	Виконано
16	Захист КРМ перед екзаменаційною комісією	26.02.2024	26.02.2024	Виконано

Розробив студент _____Димо В. В._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи __д-р техн. наук, проф. Гожий О.П._____ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«09» листопада 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
студента групи 601 ЧНУ ім. Петра Могили

Димо Валерія Володимировича

на тему: **«ІНТЕЛЕКТУАЛЬНА СИСТЕМА РОЗПІЗНАВАННЯ ПОШКОДЖЕНИХ БУДИНКІВ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ»**

Актуальність дослідження полягає у необхідності автоматизації процесу виявлення пошкоджених будівель для оцінки руйнувань, використовуючи знімки поверхні землі. Використання сучасних нейронних мереж та алгоритмів прискорить даний процес, підвищить точність прогнозів, що в свою чергу допоможе в ході відновлення інфраструктури та житлового сектору країн, що постраждали в наслідок негоди або ведення бойових дій.

Об'єктом дослідження є процес розпізнавання пошкоджених будівель на знімках (супутникових або за допомогою БпЛА).

Предметом дослідження є нейромережеві моделі та технології для розпізнавання пошкоджених будівель.

Мета дослідження – автоматизація процесів розпізнавання будівель, які мають пошкодження, з використанням нейронних мереж.

В результаті виконання роботи було досліджено сучасні моделі нейронних мереж для проведення семантичної сегментації, їх переваги та недоліки, обрано як оптимальну архітектуру U-Net, проаналізовано вплив параметрів моделі на точність класифікації, побудовано відповідну модель, а також розроблено програмне забезпечення, що використовує нейронну мережу у роботі.

Робота складається з 5 розділів, кожен відповідно присвячений: аналізу предметної області, збору даних та будові мережі U-Net, реалізації згорткової мережі та інтелектуальної системи, аналізу отриманих результатів, а також охороні праці та методичній частині кваліфікаційної роботи. Загальний обсяг роботи – 117 сторінок. Магістерська кваліфікаційна робота містить один додаток, 79 рисунків, 4 таблиці і посилання на 58 літературних джерел.

Ключові слова: штучний інтелект, згорткові нейронні мережі, семантична сегментація, виявлення пошкоджених будівель, CNN, U-Net.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Dymo Valerii

«INTELLIGENT SYSTEM FOR RECOGNITION OF DAMAGED BUILDINGS BASED ON NEURAL NETWORKS»

The relevance of the study lies in the need to automate the process of detecting damaged buildings for damage assessment using images of the earth's surface. The use of modern neural networks and algorithms will speed up this process, increase the accuracy of forecasts, which in turn will help in the course of restoring the infrastructure and housing sector of countries affected by natural disasters or hostilities.

The object of the study is the process of recognizing damaged buildings on images (satellite or UAV).

The subject of research is neural network models and technologies for recognizing damaged buildings.

The purpose of the research is to automate the processes of recognizing damaged buildings using neural networks.

As a result of the work, modern models of neural networks for semantic segmentation were investigated, their advantages and disadvantages were chosen as the optimal U-Net architecture, the influence of model parameters on classification accuracy was analyzed, an appropriate model was built, and software using a neural network was developed.

The work consists of 5 sections, each of which is respectively dedicated to: analysis of the subject area, data collection and building of the U-Net network, implementation of the convolutional network and intelligent system, analysis of the obtained results, as well as labor protection and the methodological part of the qualification work.

The overall scope of the work is 117 pages. The thesis contains one application, 79 figures, 4 tables and 58 sources in it.

Keywords: artificial intelligence, convolutional neural networks, semantic segmentation, detection of damaged buildings, U-Net.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БпЛА – безпілотний літальний апарат

ООН – Організація Об’єднаних Націй

ЮНЕСКО – Організація Об’єднаний Націй з питань освіти, науки і культури

API – application programming interface

BIOSS – Biological Signalling Studies

CCE – categorical cross-entropy

DaLA – Damage and Loss Assessment

FCN – fully convolutional networks

GLCM – gray level co-occurrence matrix

GUI – graphical user interface

IoU – intersection over union

R-CNN – Region-based Convolutional Neural Network

ReLU – rectified linear unit

RGB – red green blue

SAR – synthetic-aperture radar

SSD – single-shot multibox-detector

SVG – scalable vector graphics

SVM – support vector machine

UI – user interface

VGG – visual geometry group

XML – extensible markup language

ВСТУП

Кожен рік на планеті відбувається багато катастроф: як природніх, так і спричинених людиною. Урагани, землетруси, цунамі – це лише невелика частина з них, від яких страждають мільйони людей на різних континентах, і з наслідками від яких необхідно рахуватися. Інша сторона катастроф – бойові дії та війни, які відбуваються з різною інтенсивністю. Непрямі збитки економікам країн від різних подій складно підрахувати з необхідною точністю, а прямі збитки достатньо часто бувають в небачених масштабах: руйнування житлового сектору, логістичних шляхів тощо.

Зазвичай експерти працюють із відкритими даними та оцінюють масштаби руйнувань вручну або за допомогою алгоритмів. Команда учених з Київської школи економіки оцінила збитки інфраструктурі України після повномасштабного вторгнення Росії 24 лютого 2022 року приблизно у 138 мільярдів доларів (за вартістю заміни) станом на кінець 2022 року [1]. Через півроку були оцінені збитки житловому сектору, що можуть складати більше 54 мільярдів доларів, найбільш постраждалими регіонами є Донецька, Харківська та Луганська області [2].

Актуальність дослідження полягає у необхідності автоматизації процесу оцінки руйнувань, наприклад, використовуючи фотографії території за допомогою супутників, або отримання зображень з безпілотних літальних апаратів. Розпізнавання пошкоджених будівель за допомогою сучасних архітектур нейронних мереж прискорить процес оцінки можливих руйнувань, підвищить точність прогнозів для різних задач, та надасть потужний інструмент експертам, що допоможе в ході відновлення інфраструктури та житлового сектору України та інших країн, які постраждали від бойових дій.

Об’єкт дослідження – процес розпізнавання пошкоджених будівель на знімках (супутникових або за допомогою БпЛА).

Предмет дослідження – нейромережеві моделі та технології для розпізнавання пошкоджених будівель.

Мета дослідження – автоматизація процесів розпізнавання будівель, які мають пошкодження, з використанням нейронних мереж. **Завдання**, які необхідно виконати для досягнення поставленої мети:

– проаналізувати наявні дослідження та публікації, які відображують можливі шляхи вирішення проблеми;

– розглянути наявні моделі нейронних мереж;

– обрати оптимальну архітектуру нейронної мережі для розпізнавання будівель на знімках;

– реалізувати обрану архітектуру за допомогою програмних засобів;

– провести тестування мережі та проаналізувати результати, за можливості покращити ефективність моделі;

– розробити інтелектуальну систему, яка буде використовувати розроблену модель для розпізнавання пошкоджених будівель.

Апробація результатів дослідження. Основні ідеї та положення, які розглядаються у дипломній роботі, викладені у статті «Застосування нейронних мереж для визначення пошкоджених будівель» та представлені в рамках XXVI Всеукраїнської щорічної науково-практичної конференції «Могилянські читання – 2023: Досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти».

1 РОЗПІЗНАВАННЯ ПОШКОДЖЕНИХ БУДІВЕЛЬ НА ЗНІМКАХ

1.1 Опис задачі, сучасний стан та шляхи вирішення проблеми

Подолання наслідків будь-якої катастрофи, не залежно від походження та масштабів – складна та багатоетапна справа, яка потребує велику кількість даних, експертиз та попередніх розрахунків, які у більшості випадків виконуються людьми самостійно. Автоматизація – крок до прискорення та покращення оцінки шкоди та збитків, що є необхідною складовою для успішного відновлення регіонів, які постраждали в наслідок природніх катастроф чи бойових дій.

Загалом існують багато міжнародних та державних агенцій, комісій та інших установ, які займаються в описаній сфері, серед них можна зазначити Економічну комісію ООН, ЮНЕСКО, Світовий банк тощо. Кожна з цих організацій мають власні комісії та методології, які націлені на оцінку руйнувань та прорахунок можливих наслідків. Так з 1972 року була розроблена Методологія оцінки шкоди та збитків (англ. DaLA, The Damage and Loss Assessment Methodology), яка була реалізована за ініціативною наведених вище організацій [3]. Даний документ націлений на пояснення різним експертам та виконавчим групам оптимального шляху для оцінки шкоди загальній економіці постраждалої країни. Автори намагаються створити гнучкий інструмент, який можна адаптувати до катастроф різних типів та безпосередньо вимог державної власності, враховуючи особливості постраждалих регіонів.

Досить часто документи, які націлені на опис збитків, які були завдані катастрофою націлені не тільки на звичайний прорахунок прямих пошкоджень, але й прогнозування подальшої зміни ситуації, вплив пошкодженою інфраструктури на економічну ситуації регіону на місяці та роки наперед. То ж в методології DaLA автори намагаються враховувати вплив катастроф на засоби існування окремих категорій осіб для того, щоб повністю визначити потреби у відновленні та реконструкції [3].

Як було сказано вище, подолання наслідків надзвичайних ситуацій – багатоетапний процес, який не може повністю виключати роботу людини. Це значно ускладнює ситуацію, оскільки експертним групам необхідно отримати інформацію від структур місцевих адміністрацій про стан регіону до катастрофи, після чого отримати інформацію про наслідки після. Дана інформація допомагає експертам зрозуміти, яка частина країни постраждала більше, де існує перша необхідність в проробці оцінки нанесеної шкоди, де постраждала критична інфраструктура чи логістичні шляхи тощо.

Але багато інформації можна отримати за допомогою сучасних технологічних засобів та інструментів, наприклад, з супутників, аерофотозйомки або звичайних дронів. За останні роки БпЛА стали широко використовуватися у громадському секторі, що спростило та здешевило багато послуг, серед яких отримання високоякісних фотографій земельних ділянок.

За допомогою фотографій можна скласти карту постраждалих регіонів, отримати загальну картину того, що відбувалося, відбувається і може відбутися у майбутньому. Все це допомагає у вирішенні проблеми, оскільки немає необхідності самостійно виїздити на місце та проводити оглядове дослідження на початкових етапах виробництва плану відновлення. Відповідно, такі карти та окремі фотографії можуть стати корисними при попередній оцінці вартості відновлення окремих об’єктів або регіону в цілому. Звісно, це не замінить повноцінної експертизи у майбутньому, але дозволить зекономити дорогоцінний час на попереднє уявлення про масштаби катастрофи та виділення коштів, необхідних для поступового подолання наслідків.

Багато компаній та надавачів послуг у сфері аеро- та космічної фотозйомки співпрацюють з міжнародними організаціями, надаючи безкоштовні фото та ресурси, а також взаємодіють із місцевою владою. Також існує можливість знайти фото наслідків катастроф, які такі компанії викладають на власних ресурсах або сайтах. Так, наприклад, робить компанія Maxar (рис. 1.1), яка викладає відкриті набори даних фактично кожен рік [4].

Рисунок 1.1 – Пошкодження завдані ураганом Доріан, фотозйомка з космосу від Maxar Technologies

Таким чином можна казати про достатню кількість інструментів для досліджень, які доступні як на безкоштовній основі, так і можуть надаватися комерційно різними компаніями. Виняток складає той факт, що здебільшого доступні ресурси стосуються природніх катастроф та катаклізмів, з якими стикається людство кожного року, це урагани, повені, землетруси тощо. У відкритих даних може бути складно знайти велику кількість даних, наприклад, які б дозволити повноцінно визначити вплив бойових дій чи війн у країнах світу.

Цей фактор необхідно враховувати, оскільки військові дії та природні катастрофи мають різні патерни руйнувань, що в подальшому ускладнюють роботу автоматизованих систем. Так, наприклад, цілком ймовірно, що після повені більшість будинків можуть залишитися цілими, але зазнати деяких пошкоджень, або зруйнуватися таким чином, що пошкодження будуть слабо помітні під прямим кутом. Інший варіант, що більшість руйнувань від землетрусів відбувається так, що будинок перетворюється на суцільну руїну. В той час, під час бойових дій характерні як руйнування всієї будівлі, так і лише її частини, або завдані збитки від великого масштабу пожежі всередині, що відповідно може не бути помітно при використанні моделі, яка пройшла тренування на наборах даних, які не мають достатню кількість таких пошкоджень.

Наприклад, на рис. 1.2 наведено приклад фотографій, які порівнюють характерні наслідки після повені, землетрусу та бойових дій.

Рисунок 1.2 – Пошкодження, завдані різними надзвичайними подіями, фото Maxar Technologies

Порівнявши деяку кількість фотографій після різних подій, можна дійти висновку, що один з етапів, на яку необхідно звернути увагу – це збір даних для подальшого використання у навчанні моделі, яка буде використовуватися в інтелектуальній системи для розпізнавання. Таким чином, необхідно або використовувати наявні дані, наприклад, за допомогою сервісів Google Maps чи Google Earth, або використовувати власні технічні засоби – зйомку з БпЛА, аерофотозйомку за допомогою зондів тощо.

Кожен із способів має власні переваги та недоліки, які необхідно враховувати, так у випадку комерційного використання фотографій сервісу Google необхідно зв’язатися із представниками компанії та обговорити ціну та сферу використання, або іншим чином провести оплату використанням інших сервісів компанії, яка надає аналогічні послуги [5]. Відповідно, власна зйомка потребує наявність технічних засобів, їх обслуговування тощо.

1.2 Аналіз наявних досліджень та публікацій

В процесі виконання кваліфікаційної роботи було виконано аналіз наявних досліджень та публікацій, які пов’язані з поставленою задачею або використовують інструменти, які можуть стати в нагоді для вирішення існуючих проблем в даній області.

У статті «Виявлення зруйнованих будівель після природньої небезпеки за допомогою супутникових зображень RGB високої роздільної здатності та модифікованих згорткових нейронних мереж U-Net», написаній вченими з Бостонського університету та університету Тафтса, розглядається використання згорткових нейронних мереж для вирішення проблеми виявлення зруйнованих будинків після природних катастроф [6]. Автори використовують набір даних із супутниковими зображеннями xView, який містить більше 2 мільйонів відмічених об’єктів розділених на 60 категорій, розділивши його на потрібні фрагменти та обравши лише обрану зону для дослідження, таким чином значно скоротивши необхідну кількість зображень для навчання.

В статті було запропоновано наступну архітектуру U-Net (рис. 1.3).

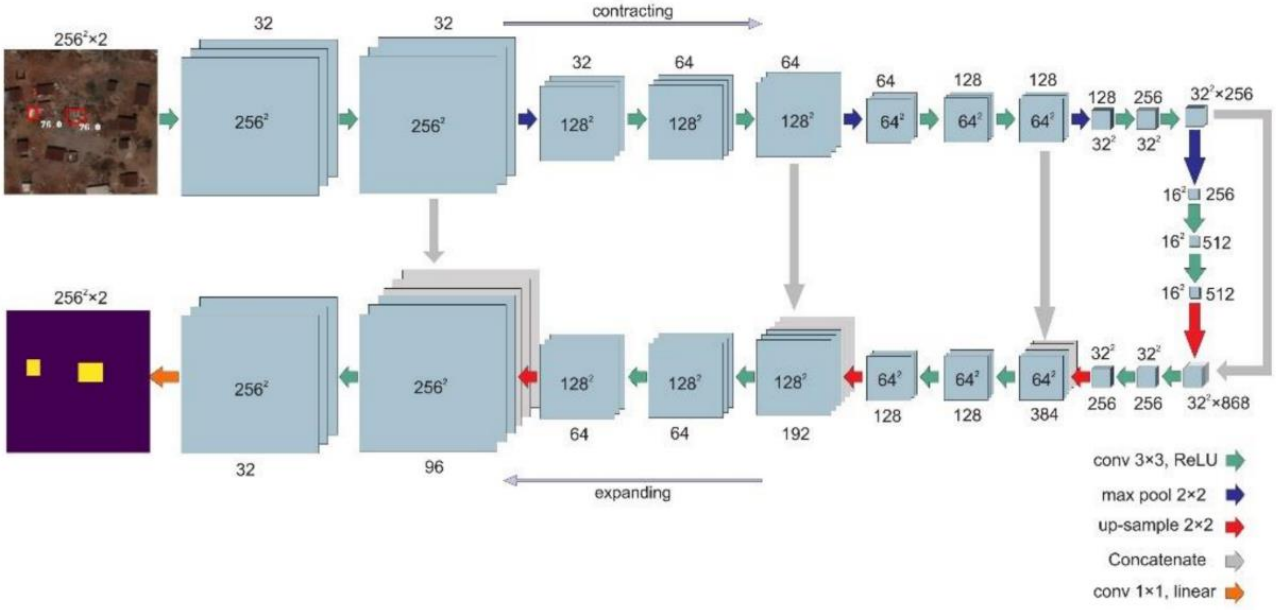

Рисунок 1.3 – Використана в дослідженні нейронна мережа

При дослідженні було використано ті зображення, які містили як тільки об’єкти з одного класу (в даному випадку: зруйновані та не зруйновані будівлі), а також обох класів, що повинно покращити процес навчання. В результаті дослідження мережа показала результат 60.3% розпізнавання зруйнованих будівель на рівні пікселів, на що вплинули особливості використаного набору даних, який містив захоплення зображень будинків разом із прибудинковою територіє. Автори також пропонують різні методи покращення результатів, наприклад, використання різних значень для вихідної функції як на рівні пікселів, так і загалом для зображення. Таким чином вдалося добитися результатів на рівні 95.9% розпізнавання цілих будівель, та 76% для зруйнованих [6].

В іншому дослідженні «Швидкий самоконтрольований метод на основі глибокого навчання для виявлення пошкоджень після землетрусу за допомогою даних БпЛА (приклад: Сарпол-і-Захаб, Іран)» вчені з Ірану та Америки пропонують більш складну архітектуру з використанням різних компонентів (рис. 1.4). Для досягнення бажаного результату, в статті пропонується новий метод, заснований на глибокому навчання, для швидкого виявлення будівель після землетрусу [7].

Система складається з багатьох компонентів, таких як обробка зображень та виділення ознак (неглибокі, глибокі та їх суміш), застосування одноепохового згорткового автокодеру для їх виділення. Після деяких процедур застосовується процедура, яка заснована на правилах, для автоматичного вибору належних навчальних зразків, які пізніше будуть подаватися алгоритмам класифікації на наступному кроці. Автори також досліджують вплив вибору відношення для навчального та тестувального набору на точність оцінки, вплив різних параметрів моделювання. Система, побудована в рамках дослідження, має загальну точність на рівні 93%, каппа коефіцієнт деяких застосованих алгоритмів склав 74% [7].

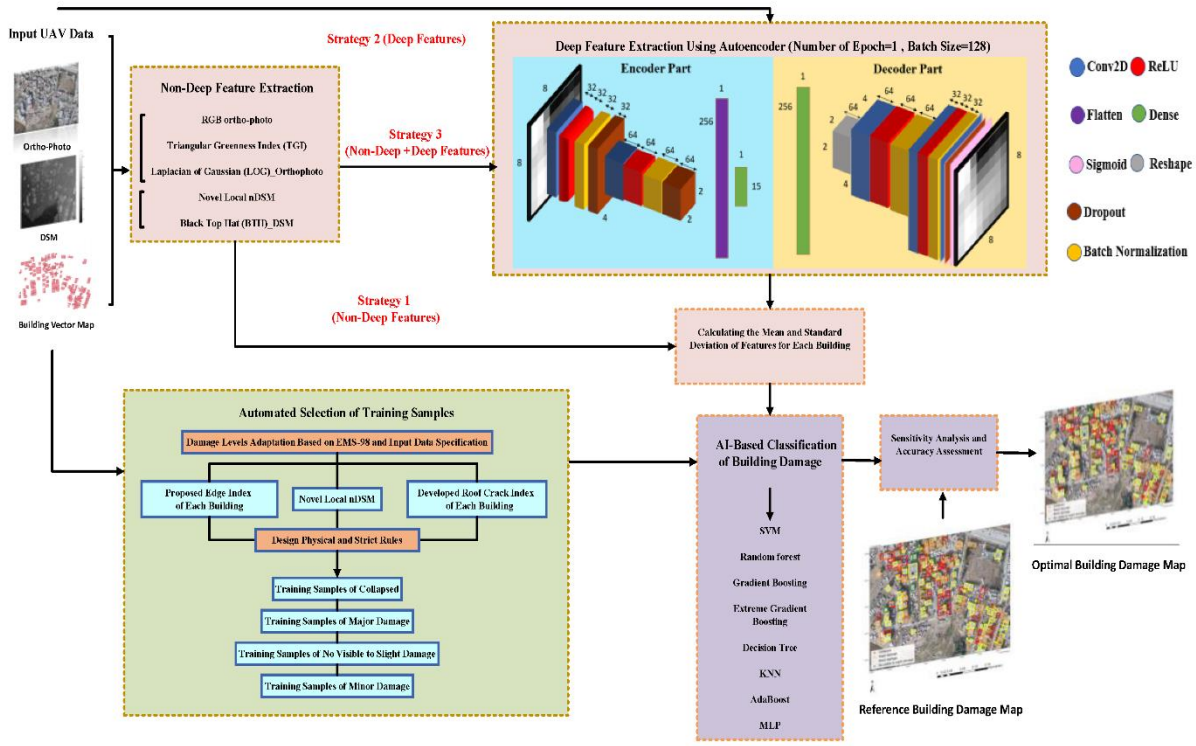

Рисунок 1.4 – Запропонована структура системи

В наступній статті «Виявлення пошкоджень будівлі з аерофотознімків після подій за допомогою SSD» йде мова про використання алгоритму SSD (англ. Single-shot Multibox Detector) для виявлення об’єктів на зображеннях та моделі автокодеру, який містить модель VGG16 [8]. Вчені з Північнокитайського технологічного університету зазначають, що використання додаткових нейронних мереж покращує загальну точність на 10% в порівнянні з простим використанням алгоритму SSD.

В дослідженні використовується набір даних, який містить високоякісні зображення з роздільною здатністю 1920 на 1080, фотографії зроблені після урагану Сенді – потужного циклону 2012 року. Основний набір містить 20000 зображень, з яких було відібрано невелику кількість необхідних, що склало приблизно 500 зображень в порівнянні з початковим, в той час 70% було виділено для навчання і 30% на тренування [8]. Побудована модель мала наступну структуру (рис. 1.5).

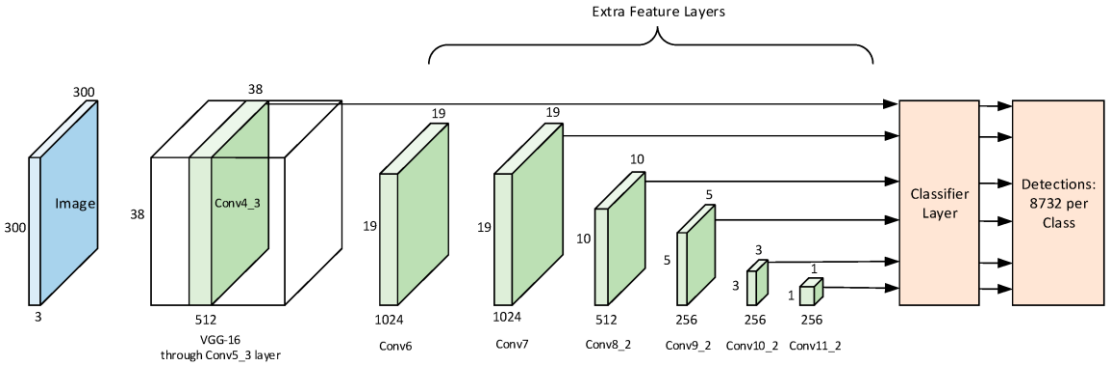

Рисунок 1.5 – Архітектура мережі SSD для дослідження

Автори наводять, що в результаті вдалося побудувати модель, яка досягла точності 79.4% класифікації зруйнованих будівель, і 70% не пошкоджених, в той час відгук моделі на тестовому наборі даних склав близько 62%.

Варто відмітити невелику кількість наявних досліджень, присвячених подоланню наслідків саме після військових дій. Наприклад, в статті авторів Національної ради з наукових досліджень та Ісламського університету Лівану «Післявоєнне виявлення пошкоджених будівель» досліджується набір даних 2015 року з Сирії, в якій відбувалася громадянська війна [9]. Зображення зняті з регіонів Забадані та близько Дамаску супутником GeoEye-1 (рис. 1.6).

Рисунок 1.6 – Приклад зображень, використаних для дослідження

Розроблена авторами математична модель використовує декілька основних ознак, таких як тінь, дисперсія та кореляція. Оскільки тінь будівлі відображає її геометричне зображення, наприклад, після вибуху або руйнувань тінь буде

деформована, що і використовується для класифікації зруйнованих та вцілілих будівель. Матриця сумісності рівнів сірого (англ. Gray Level Co-occurrence Matrix, GLCM) представляє статистичні характеристики текстури другого порядку зображення. GLCM – це матриця, де кількість рядків і стовпців дорівнює кількості рівнів сірого, на зображенні. В результаті тестування на одному із зображень, точність класифікації склала 95.65% на розпізнавання вцілілих будівель і 81.25% на пошкоджених, що є беззаперечно гарним результатом.

Ще одна робота, яка розглядається в рамках даного дослідження присвячена оцінці пошкоджених будівель у Києві внаслідок повномасштабного вторгнення Росії «Оцінка пошкоджень будівель, пов’язаних із війною, у Києві, Україна, за допомогою радіолокаційних зображень Sentinel-1 та оптичних зображень Sentinel-2» [10]. Робота від авторів, які вже були зазначені на початку цього розділу, розглядають інший можливий спосіб розпізнавання зруйнованих будівель, а саме – використання SAR-зображень разом з аналізом текстур супутникових знімків та обрахуванням коефіцієнту інтенсивності пікселів.

Автори нагадують, що зображення не завжди можна отримати із-за того, що над досліджуваним регіоном можуть бути хмари або погана видимість через інших причині, що унеможливлює використання зображень із звичайних супутників. В той час, обробка SAR-зображення в багатьох випадках потребують іншого підходу, тому вчені досліджують розроблений метод для поставленої задачі, без використання нейронних мереж. Дана робота цікава дослідженням впливу розміру будівлі на відповідному зображенні на якість та точність оцінки. Так, в ході дослідження було отримано модель з загальною точністю 58% з урахуванням всіх об’єктів у наборі даних, в той час як більшість будівель мала невелику площину. При зменшені площі будівлі збільшувалася точність розпізнавання, враховуючи площу об’єктів більших за 300 метрів квадратних, точність розпізнавання збільшувалася до 76%, що є позитивним результатом [10].

1.3 Дослідження нейронних мереж для поставленої задачі

Виходячи з аналізу наявних публікацій, можна зробити висновки, що кожен обраний алгоритм та підхід можуть мати свої особливості, позитивні та негативні сторони, які необхідно буде вирішувати. Класичні алгоритми є досить простим та дієвим рішенням у більшості ситуацій, але на складних задачах застосування тільки їх може не надавати бажаних результатів. Якщо раніше класичні алгоритми дозволяли зекономити потужності машини, а також наявність у тогочасних комп'ютерів невеликої пам'яті, зараз цієї проблеми здебільшого не існує.

Використання нейромережевих технологій – сучасний та потужний підхід для вирішення поставленої задачі. За останні десятиліття штучні нейронні мережі та алгоритми глибокого навчання зробили суттєвий прорив, створивши багато корисних та ефективних моделей, які застосовуються як у промисловості, медицині, науці, так і в звичайному житті людини – вони впроваджуються в застосунки для комп'ютерів та телефонів, використовуються у соціальних мережах та супермаркетах.

Розробляючи нейронні мережі, вчені надихалися будовою людського мозку: основна одиниця, клітина-нейрон, в якому через дендрити в ядро поступають деякі сигнали (тобто інформація), що з часом має властивість накопичуватися, в свою чергу це призводить до збудження та відповіді – аналогічним сигналом до аксону [11]. Клітини створюють групи (шари), яких в людському мозку незліченна кількість, і постійно працюють, передаючи та оброблюючи інформацію. Сучасні нейронні мережі набагато складніші за перші алгоритми, які були представлені. Одні з таких мереж – клас згорткових нейронних мереж, в основі яких лежить поняття згортки: дані мережі складаються з багатьох шарів, кожен з яких має властивість «згортатися» в менші за розміром, або «розгортатися» відповідно, використовуючи різноманітні операції та фільтри [12]. Дані нейронні мережі стали дуже популярними для обробки зображень та

текстової інформації, оскільки її набагато легше представити у вигляді матриць або подібних.

Виходячи із проаналізованої інформації, варто звернути увагу на декілька архітектур згорткових мереж, які використовуються для вирішення задач розпізнавання об’єктів на зображеннях, що були представлені як в статтях, наведених вище, так і в інших виданнях та літературі, яка освітлює сучасні нейромережеві моделі.

VGG16. Дана модель розроблена науковцями Кареном Симоняном та Ендрю Зіссерманом з Департаменту інженерних наук Оксфордського університету. Була представлена на конкурсі ImageNet 2014 році та отримала перші місця у виявлені та класифікації об’єктів (рис. 1.7), назва VGG походить від назви дослідницької групи (англ. Visual Geometry Group), в яку входять автори [13].

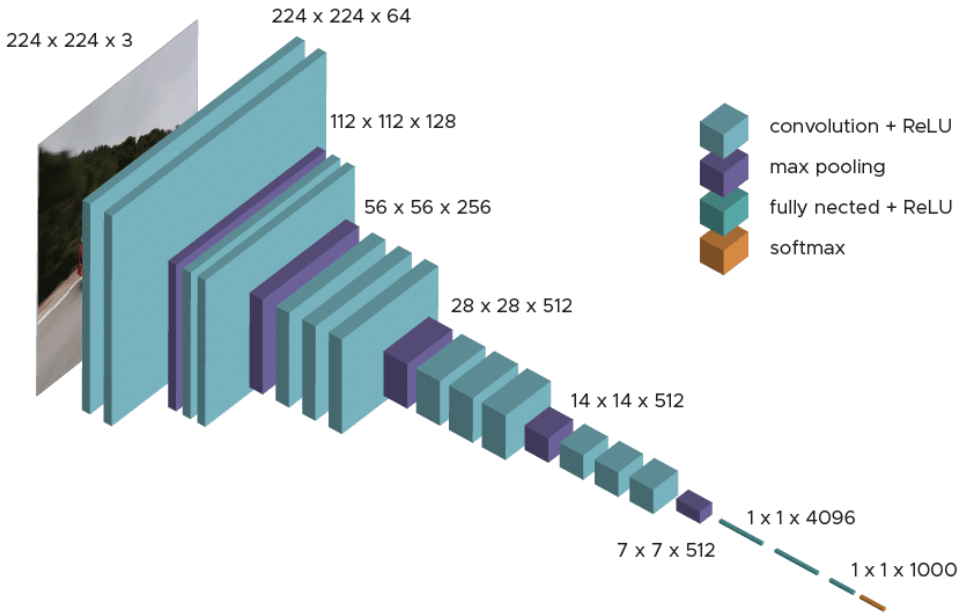

Рисунок 1.7 – Архітектура VGG16

VGG16 – це згорткова нейронна мережа, що є різновидом штучних нейронних мереж, має вхідні, приховані та вихідні шари. Автори модифікували згорткову мережу, збільшивши глибину, зменшили фільтри згортко до розміру 3

на 3, а також глибину до 16-19 шарів. Таким чином модель збільшила глибину та кількість параметрів, що коливається від 133 до 144 мільйонів, в залежності від конфігурації. Автори вирішили сконцентруватися на фільтрах, використовуючи фільтр шару згортки 3 на 3 з кроком 1, а також фільтру 2 на 2 з кроком 2 для повнозв'язного шару, а не на кількості гіперпараметрів нейронної мережі [13].

Таким чином розробникам вдалося розробити модель, яка змогла досягти дуже гарних результатів на наборі даних ImageNet, який містить 14 мільйонів зображень та 1000 класів, а саме – близько 92.7%. В той час як модель має високу точність, в неї є відповідні особливості та потреби, а саме [13]:

– низька швидкість навчання (оригінальна модель була навчена на наборі ImageNet протягом 2-3 тижнів);

– складність архітектури, відповідно – більше зв'язків для налаштування;

– великий розмір навченої моделі, близько 500 МБ, що може не підійти для деяких задач.

Модель VGG має різні конфігурації, в залежності від глибини та кількості шарів нейронної мережі, що може давати різні результати. Дана згорткова мережа показала себе з позитивної сторони в задачах класифікації, знаходження об'єкту на зображені та семантичної сегментації, і відповідно може бути застосована для вирішення завдання розпізнавання пошкоджених будівель.

U-Net. Мережа була представлена групою науковців з Відділу комп'ютерних наук і Центру досліджень біологічних сигналів BIOSS, як рішення класифікації (а саме сегментації) біомедичних зображень, наприклад, знаходження пошкоджених клітин на знімку, виявлення хворих ділянок легень тощо [14]. Загалом в розробленій мережі лежить ідея повнозв'язної згорткової нейронної мережі, яка була б ефективна для захоплення контексту, просторової інформації, в основному для біомедичних зображень (рис. 1.8).

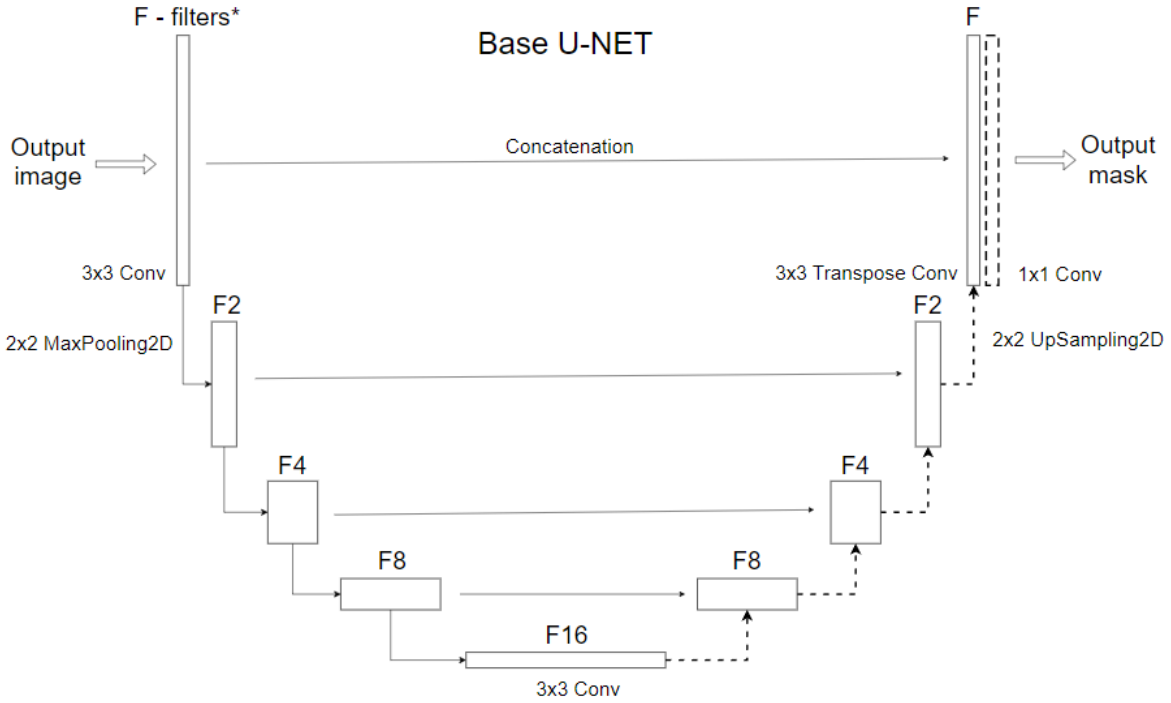

Рисунок 1.8 – Архітектура мережі U-Net

Підхід, що використовується в даній мережі, використовує скорочувальний шлях для захоплення контексту, та відповідно розширювальний – для точної локалізації, тобто працює за логікою encoder-decoder. В той час, мережа уникає повнозв'язних шарів, і використовує згортку для підтримки контекстної інформації. Шари підвищення дискретизації (англ. upsampling layers) використовуються для збільшення вихідної роздільної здатності, а також функції високої роздільної здатності зі шляху підвищення дискретизації, який звужується. Модель використовує фільтри 2 на 2, а також 3 на 3 для операцій згортки із кроком 2 для зменшення дискредитації, а також згортку 1 на 1 для 64-компонентного вектору ознак у потрібну кількість класів. Таким чином мережа має всього 23 шарів згортки [14].

В дослідженнях авторів зазначаються результати, які були отримані для спеціалізованих наборів даних, що містять різноманітні біомедичні зображення, точність на яких модель показала у 77.56% та 92.03% відповідно, що є значним покращенням у наявних моделей на той час [14]. В свою чергу, виходячи з

результатів дослідження та аналізу інших дослідників на цю тему [15], мережа має наступні переваги:

– покращення даних за допомогою аугментації;

– велика кількість ознак, які відбираються;

– висока ефективність при роботі з зображеннями високої роздільної здатності;

Відповідно, мережа має деякі особливості, в тому числі [14, 15]:

– велика кількість параметрів через відсутність зв’язків і додавання додаткових шарів для шляху розширення;

– можлива висока обчислювальна вартість, враховуючи будову нейронної мережі.

Враховуючи особливості роботи нейронної мережі та результати, надані в ході дослідження, згорткова модель U-Net може цілком застосовуватися в обраній сфері.

R-CNN. Модель розроблена командою науковців на чолі з Росом Гіршиком та представлена у статті «Багата ієрархія функцій для точного виявлення об’єктів і семантичної сегментації» у 2014 році. Як зазначають автори, система виявлення складається з трьох модулів: перший – генерує незалежні від категорії пропозиції регіону, що визначають набір потенційних виявлень, друга – велика згорткова мережа, яка «витягує» вектор ознак фіксованої довжини з кожної області, і останній модуль – набір лінійних SVMs (англ. Support vector machines). Будова представлена на рис. 1.9 [16].

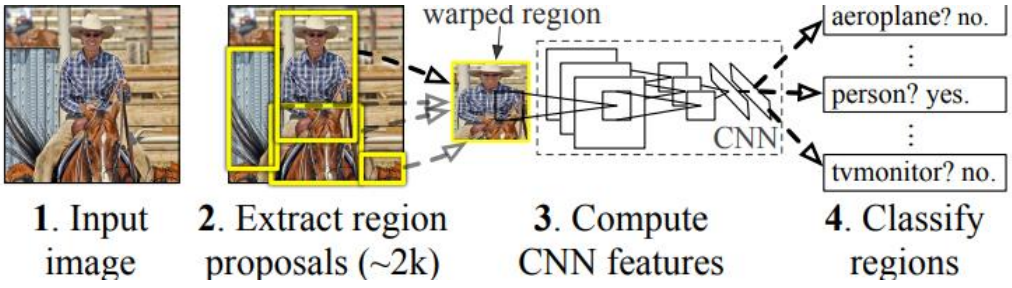

Рисунок 1.9 – Представлена схема роботи мережі R-CNN

Основні особливості роботи R-CNN, які були представлені [16].

1. Відбір регіонів: вибірковий пошук використовувався для створення набору регіональних пропозицій, які ймовірно містили об’єкти. Ці області були виділені із зображення та розглядалися як потенційні об’єкти.

2. Вилучення ознак: кожна пропозиція регіону була змінена до фіксованого розміру та передана через попередньо навчену CNN для вилучення вектора ознак фіксованої довжини для цього регіону.

3. Класифікація об’єктів: витягнуті ознаки було подано в окремий класифікатор для виявлення об’єктів. У випадку оригінального R-CNN це, як правило, була опорна векторна машина (SVM).

4. Точне налаштування: CNN, який використовувався для виділення функцій, зазвичай був попередньо навченою моделлю, як-от AlexNet або VGG, яка була налаштована на завдання виявлення об’єктів для підвищення продуктивності.

Таким чином, авторам вдалося досягти оптимальності в деяких випадках, і покращити результати поточних моделей, які існували на той час. Автори наводять дизайнерські рішення в моделях, опис роботи на тестування в своєму дослідженні, та відповідно результати моделей на наборах даних PASCAL VOC 2010 12 і ILSVRC2013. Початкова модель, яка була запропонована авторами змогла в середньому з точністю 47.9% класифікувати та сегментувати зображення на PASCAL VOC 2011 (рис. 1.10), що є відносно непоганим результатом для такого набору та задачі семантичної сегментації.

VOC 2011 val	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
O_2P [4]	**84.0**	**69.0**	21.7	47.7	42.2	42.4	**64.7**	**65.8**	57.4	**12.9**	37.4	20.5	43.7	35.7	52.7	51.0	**35.8**	**51.0**	28.4	59.8	49.7	46.4
full R-CNN fc_6	81.3	56.2	23.9	42.9	40.7	38.8	59.2	56.5	53.2	11.4	34.6	16.7	48.1	37.0	51.4	46.0	31.5	44.0	24.3	53.7	51.1	43.0
full R-CNN fc_7	81.0	52.8	**25.1**	43.8	40.5	42.7	55.4	57.7	51.3	8.7	32.5	11.5	48.1	37.0	50.5	46.4	30.2	42.1	21.2	57.7	**56.0**	42.5
fg R-CNN fc_6	81.4	54.1	21.1	40.6	38.7	**53.6**	59.9	57.2	52.5	9.1	36.5	**23.6**	46.4	38.1	53.2	51.3	32.2	38.7	**29.0**	53.0	47.5	43.7
fg R-CNN fc_7	80.9	50.1	20.0	40.2	34.1	40.9	59.7	59.8	52.7	7.3	32.1	14.3	48.8	42.9	54.0	48.6	28.9	42.6	24.9	52.2	48.8	42.1
full+fg R-CNN fc_6	83.1	60.4	23.2	48.4	**47.3**	52.6	61.6	60.6	**59.1**	10.8	**45.8**	20.9	**57.7**	43.3	**57.4**	**52.9**	34.7	48.7	28.1	60.0	48.6	**47.9**
full+fg R-CNN fc_7	82.3	56.7	20.6	**49.9**	44.2	43.6	59.3	61.3	57.8	7.7	38.4	15.1	53.4	**43.7**	50.8	52.0	34.1	47.8	24.7	**60.1**	55.2	45.7

Рисунок 1.10 – Результати сегментації зображень, наведені в ході тестування мережі R-CNN

Перевагами моделі являються підвищена точність, гарна локалізації об’єктів та можливість вдосконалення. В той час, з деякими особливостями, з якими прийдеться мати справу є велика складність обчислення, повільна робота із зображеннями в режимі реального часу та можливість перекриття одного регіону іншим, що ускладнює роботу з класифікацією зображень. З іншого боку, за останні роки було розроблено багато нових архітектур та модифікацій, серед яких, наприклад, Fast R-CNN, Faster R-CNN та Mask R-CNN, які змогли покращити деякі моменти в базовій згортковій мережі та відповідно отримати кращі результати в тих задачах, для яких вони було розроблені [16-18].

Висновки до розділу 1

В першому розділі було проаналізовано поточну ситуацію в області оцінки пошкоджень та збитків у наслідок різних катастроф, наведено приклади вирішення даного завдання, виявлено місце використання технологій, які б допомогли в автоматизованому виявлені пошкоджених будівель на картах (зображеннях), що б значно спростило етап виявлення в зоні стихійного лиха бо під час бойових дій.

Відповідно, було проаналізовано наявні дослідження та публікації, які були націлені на вирішення даної або суміжних проблем, визначено необхідні кроки для подолання, а також наявні та можливі інструменти, які використовуються для розпізнавання пошкоджених об’єктів на зображеннях. Автори використовують як класичні алгоритми, так більш складніші нейронні мережі, які націлені на виявлення будівель або сегментацію зображень.

Найчастіше для обробки та аналізу зображень використовують згорткові нейронні мережі, оскільки їх архітектура та принцип роботи найкраще розпізнають їх особливості, із можливих для використання можна виділити архітектури U-Net, R-CNN та VGG.

2 АРХІТЕКТУРА U-NET ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗНІМКАХ

2.1 Згорткові нейронні мережі

Формування сучасного виду та функціонування нейронних мереж було започатковано ще в минулому столітті, в деяких випадках дякуючи дослідникам в таких областях як медицина та біологія – вони надали інформацію, як працюють людські та тваринні сенсори, що спонукало на розробку відповідних алгоритмів. Якщо перші нейронні мережі були засновані на принципах роботи людського мозку, то згорткові нейронні мережі наслідують функціонування зорової кори тварин [19, 20].

Роботи Г'юбела та Візела надихнули розробників на створення нових алгоритмів, наприклад, неокогнітрону, представленого у 80-х роках минулого століття, який використовувався для розпізнавання патернів на зображеннях. Робота Фукусіми – розробника нового алгоритму, запропонувало використання «простих» та «складних клітин» [20]. Оскільки зі збільшенням вхідного шару та додатковими шарами відповідно збільшується кількість параметрів всієї моделі, нейронну мережу стає важче та довше навчати, в той час запропонований метод дозволяв скоротити просторову інформацію, якою б оперував алгоритм, тим самим спростивши модель та підвищивши рівень варіативності, який компенсувався більшою кількістю карт ознак [19].

Як результат, в 90-х роках власну згорткову мережу розробив науковець Ян Лекун, в 1998 році в своїй статті він описав згорткову мережу LeNet з наступною архітектурою (рис. 2.1). Вона поєднувала різні напрацювання та дослідження, Лекун розробляв мережу для розпізнавання символів у документах, і зміг досягти гарних результатів – тестування показало помилку на тестовому наборі менше 1% [19], але через брак потужностей, мережа LeNet не стала популярною у свій час.

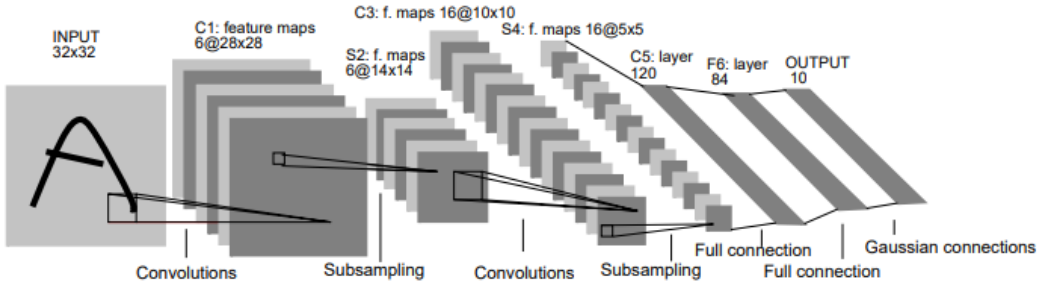

Рисунок 2.1 – Запропонована архітектура мережі для розпізнавання рукописних символів

Після стрімкого розвитку техніки, програмних та апаратних можливостей, згорткові нейронні мережі набули більшої популярності. Так у 2012 році було представлено концепцію нейронної мережі AlexNet на конкурсі ImageNet, яка на різних тестах показала відмінні результати: топ-1 та топ-5 рівень помилок були відповідно 37.5% та 17.0%, що було краще минулих найкращих моделей [21]. На рис. 2.2 наводиться початкова архітектура нейронної мережі.

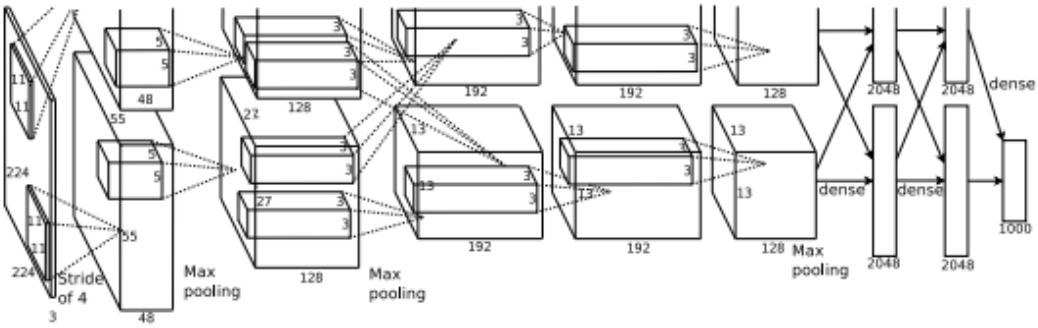

Рисунок 2.2 – Згорткова мережа AlexNet

Мережа AlexNet має 60 мільйонів параметрів та 650 тисяч нейронів, які розміщені у 5 згорткових шарах з шарами максимального поєднання (англ. max-pooling), трьох повнозв'язних шарів, вихідних шар має 1000 каналів з функцією softmax, що використовується для класифікації відповідно 1000 класів. Автори зазначають, що використання графічного процесору (відеокарти) дозволило пришвидшити навчання та підвищити ефективність операції згортки.

Використання методу «випадіння» (англ. dropout), який був розроблений в тому ж році, в повнозв'язних шарах дозволило ефективно боротися з перенавчанням моделі [21].

Враховуючи всі напрацювання, які були отримані до цього часу, багато фахівців почали розробку власних згорткових мереж та представляти нові архітектури, багато з яких ставали все більш ефективними, в порівнянні з минулими. Якщо перші моделі були простими, відносно сучасних варіантів, новітні архітектури можуть мати десятки шарів в різних комбінаціях та розмірностей, але загалом ідеї згорткових мереж залишилися, багато в чому завдяки роботі Яна Лекуна.

Таким чином, на сьогодні більшість згорткових нейронних мереж використовують наступні структурні особливості.

Вхідний шар. Приймає дані для тренування моделі, наприклад, у випадку розпізнавання пошкоджених будівель, це відповідні зображення поверхні деякого розміру. Варто зауважити, що від розміру зображення залежить складність нейронної мережі. Так, для невеликого кольорового зображення 128 на 128 пікселів, кількість параметрів для навчання буде складати менше 50000, в той час, для RGB-зображень 512 на 512 це вже буде 786 тисяч, що більше в 10 разів.

Під час навчання на вхід нейронної мережі поступово передаються зображення, зазвичай відбувається передача одразу декількох зображень – партії (англ. batch). Розмір партії – це налаштовуваний гіперпараметр, який визначає кількість зразків (зображень), які будуть передані на опрацювання моделі перед оновленням параметрів. При подачі зображень невеликими партіями зменшується навантаження на систему, оскільки комп'ютер отримує за раз менше об'єктів для обробки, в той час надто малий розмір партії може вплинути на кінцеву точність.

З вхідного шару зображення передаються на шар згортки, таким чином передаючи інформацію в глиб мережі.

Шар згортки. Являється основним шаром згорткових мереж, оскільки виконує важливу функцію – збирання ознак із зображення. Відбувається це за

допомогою ядра (англ. kernel), його ще називають фільтром. Це деяка область, яка проходить по всьому зображенню та за допомогою простих розрахунків обраховує кінцеве значення в заданій області (рис. 2.3) [22]. Зазвичай це відбувається шляхом перемноження параметрів кожного нейрона та підсумовування.

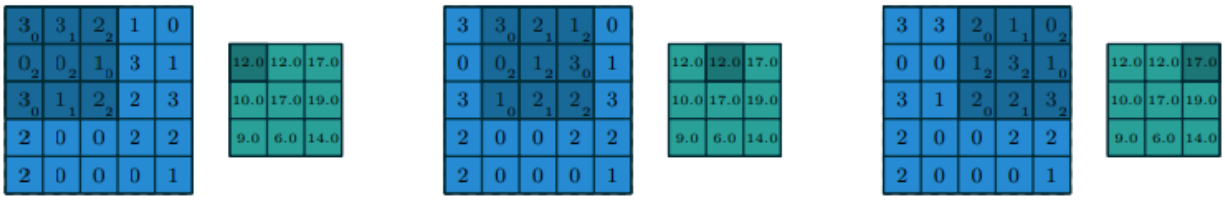

Рисунок 2.3 – Перші три кроки згортки

Згортка є лінійною операцією, але завдяки більш складним технікам дозволяє виявляти різні патерни та об’єкти, що значно вплинуло на використання даних мереж в задачах комп’ютерного зору. Від розміру ядра залежить те, яким саме особливостям буде надаватися перевага: так менший розмір ядра буде збирати локальні патерни, більші фільтри будуть збирати загальну інформацію, але втрачати локальну.

Шар згортки використовує такі параметри як розширення (англ. padding) та крок (англ. stride), що значно розширює можливості у збиранні та виявлені особливостей. Так крок відповідає за те, скільки буде проведено ітерацій ядром на одному зображенні, розширення дозволяє збільшити чи зменшити область збирання інформації (якщо це необхідно). На рис. 2.4 наведено приклад застосування різних налаштувань ядра для операції згортки [22].

У випадках застосування padding, зону без інформації ініціалізують нулями. Як можна побачити, в більшості випадках згортка використовується для зменшення вихідного розміру, в той час як іноді потребується обернена операція, яка дозволяє обрати більше ознак з області.

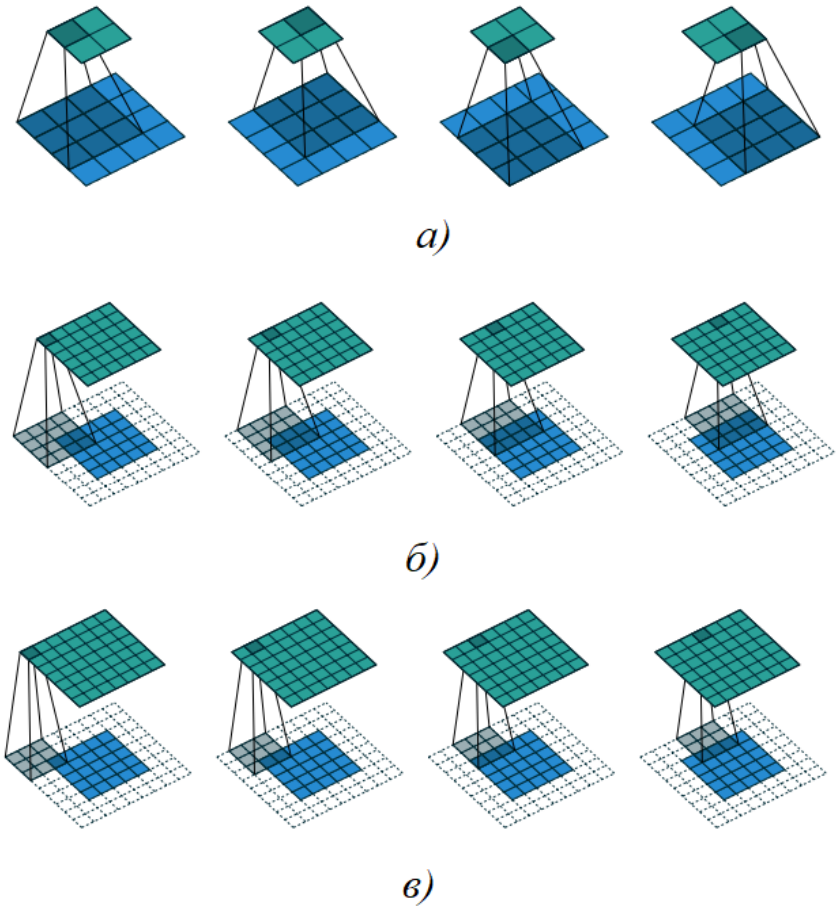

Рисунок 2.4 – Використання різних налаштувань ядра згортки: а) ядро 3 на 3 із кроком 1 без розширення; б) ядро 4 на 4 з напіврозширенням 2 на 2 та кроком 1; в) ядро 3 на 3 з повним розширенням 2 та кроком 1.

Таким чином в міру поглиблення у мережу, особливості зображень стають більш ексклюзивними та інформативними, в той час надмірність зменшується. Це відбувається через повторюваність згорток на кожному етапі та стиснення інформації у підвибірках.

Шар об'єднання. Об'єднання (англ. pooling) є наступною важливою операцією в згорткових мережах та створює окремий блок (який іноді можуть називати підвибірками, англ. subsampling), який використовується для зменшення розміру матриць особливостей за допомогою деяких математичних функцій. Зазвичай використовується максимізація значень ядра (англ. max-pooling), але існують й інші, наприклад, середнє арифметичне (рис. 2.5) [22].

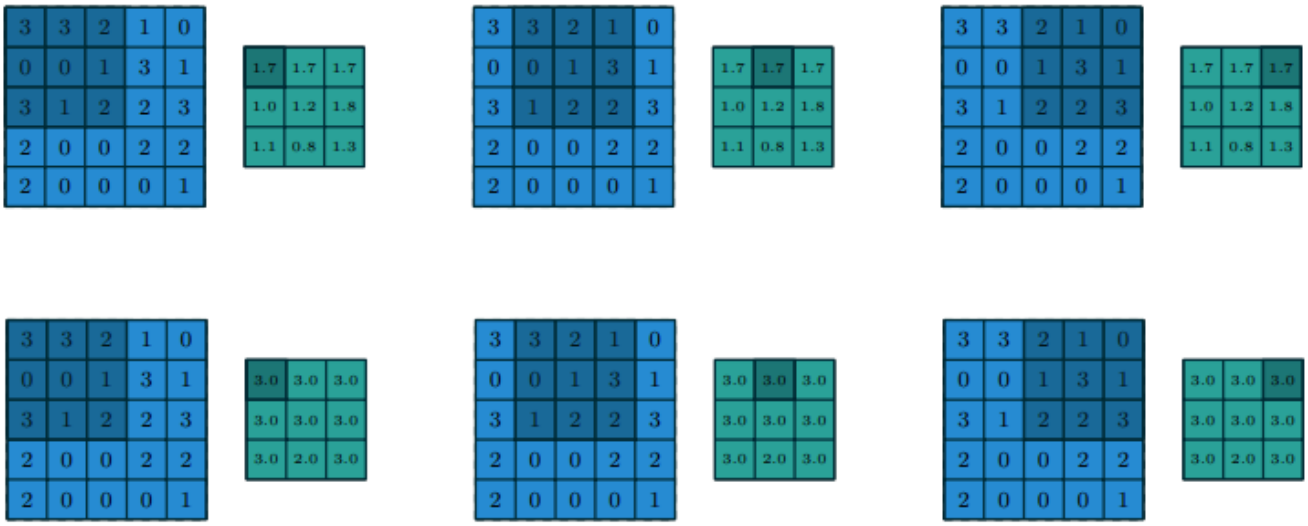

Рисунок 2.5 – Використання функцій об’єднання, які використовують середнє та максимальне значення ядер відповідно

Як можна побачити, операція об’єднання дуже схожа на згортку, але передбачає використання деякої функції замість лінійних операцій. Шар об’єднання допомагає зменшити обчислювальну складність моделі шляхом спрощення представлення та виявлення найбільш важливих ознак, наприклад, при застосуванні операції max-pooling. Недоліком операції може бути втрата деякої інформації, якщо обраний розмір ядра надто великий відносно вхідних даних тощо.

Шар з функцією активації. Оскільки в більшості випадків у мережі застосовуються лінійні операції, даний шар створений для внесення нелінійності в розрахунки. Шар може містити будь-яку функцію активації, загалом вибір грає важливу роль в залежності від поставленої задачі та набору даних, тому цьому кроку потрібно приділити значну увагу.

На рисунку 2.6 наведено вигляд функцій сигмоїди та ReLU (англ. Rectified Linear Unit) – скоригованих лінійних блоків [23]. Дані функції часто застосовуються в машинному навчанні, але можуть використовуватися й інші.

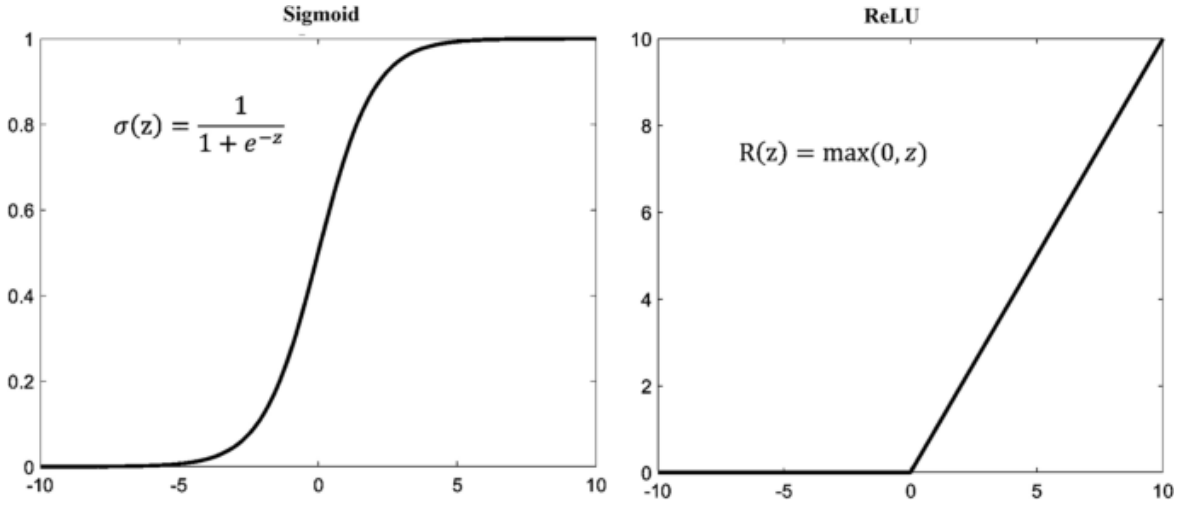

Рисунок 2.6 – ReLU та сигмоїдальна функція активації

Виходячи із поставленого завдання та набору даних, можуть використовуватися різні функції, наприклад, ReLU має меншу обчислювальну складність і часто використовується в прихованих шарах нейронної мережі, сигмоїдальна функція підходить для бінарної класифікації, коли існують 2 класи, в той час як soft-max функція застосовується при визначенні ймовірностей для декількох класів.

Повнозв'язний шар. В класичних згорткових мережах використовуються повнозв'язні шари, кожен нейрон яких поєднується з кожним нейроном минулого шару, таким чином відтворюючи всі комбінації між ними. В багатьох задачах це допомагає «засвоїти» нейромережі різні варіації патернів та адаптуватися до виконуваної задачі [12, 18].

Зазвичай даний шар йде після згортки та об'єднання, перетворюючи 2D-матрицю (якою представлене зображення) у одномірний масив із всіма ознаками, який в процесі використовується для класифікації. Так, наприклад, в задачі класифікації чисел повнозв'язний шар містив би 10 нейронів, кожен з яких відповідав за відповідний клас. Якщо мова йде про задачу сегментації, в такому випадку даний шар може представляти з себе декілька окремих структур, кількість яких відповідає кількості класів, які можуть бути на зображені.

Варто зазначити, що не в кожній згортковій мережі може бути наявні повнозв'язні шари, наприклад, автори архітектури U-Net зазначають, що в мережі вони відсутні [14].

Вихідний шар. Останній шар, що міститься в нейронній мережі і відповідає за кінцевий результат. В задачах класифікації, як зазначалося вище, це може бути один шар нейронів, кількість яких відповідає наявним класам, і відповідно містить значення ймовірності відношення до них. В задачах сегментації – це декілька шарів нейронів відповідною розмірністю, зазвичай яка відповідає вхідному зображенню (але може бути й модифікована за необхідністю).

2.2 Техніки комп'ютерного зору

Комп'ютерний зір, як дисципліна, належить до теорії та технології створення машин (комп'ютерів), які були б здатні розпізнавати зображення та об'єкти на них, як робить це людина. Коли йде мова про штучний інтелект, основною задачею комп'ютерного зору є навчання нейронної мережі таким чином, щоб алгоритм зміг виконати поставлену задачу, це може бути класифікація об'єкту на зображенні, виділення зони, на якому наявний об'єкт (або декілька водночас), створення сегментаційної карти тощо [24, 25]. Таким чином можна виділити декілька технік, які можна застосувати у вирішенні поставленої задачі – розпізнавання пошкоджених будівель. На рис. 2.7 наведено приклад таких технік.

Одна із звичайних задач, з якою мають справу вчені – класифікація, можна окремо виділити бінарну класифікацію та багатокласову, відповідно в таких задачах існує або два класи, або більше, наприклад, розпізнавання різних тварин на зображенні можна віднести до багатокласової класифікації, в той час як розпізнавання людини і тварин – бінарна. В цілому, вирішення задачі класифікації не є складною справою для нейронних мереж на даний час.

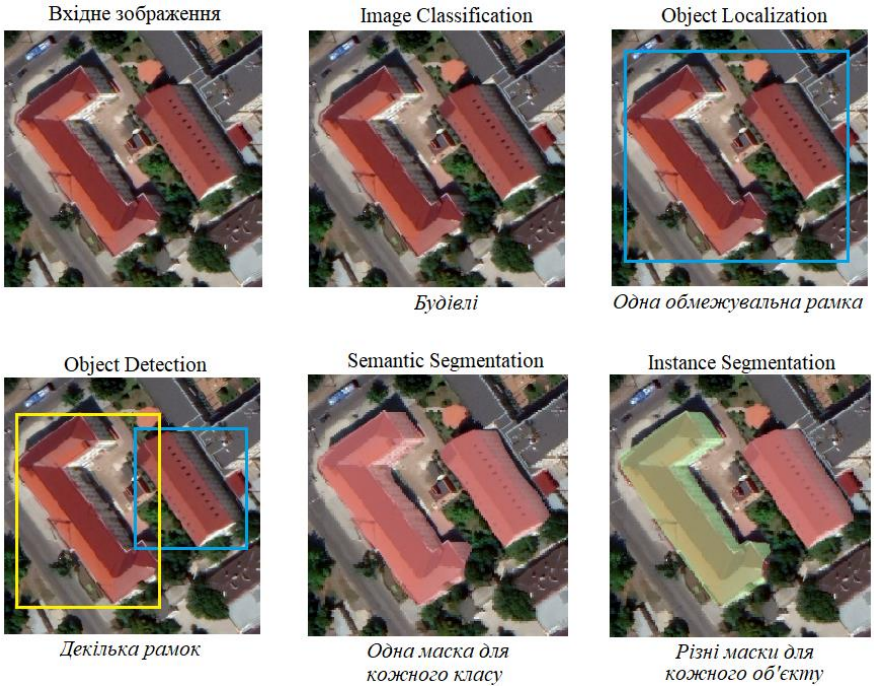

Рисунок 2.7 – Техніки комп’ютерного зору

Більш складними техніками є визначення зони, в якій може знаходитися об’єкт. В такому випадку йде мова про дві техніки – це локалізація (англ. object localization) або виявлення об’єкту (англ. object detection). Дві техніки в цілому схожі між собою, і основною сутністю є визначення координат, між якими може знаходитися об’єкт або декілька об’єктів [25].

Якщо йде мова про локалізацію, зазвичай мається на увазі виявлення зони одного головного об’єкту на зображенні. При визначенні координат, їх можна поєднати рамкою, таким чином отримавши уявну зону (третій приклад на рис. 2.7). Така техніка може застосовуватися у задачах, де на фотографії необхідно виділити один об’єкт, наприклад, маску на обличчі людини. В задачі розпізнавання пошкоджених будинків дана техніка може застосовуватися, але в такому випадку необхідно буде визначати кожен будинок окремо, нормалізовувати зображення та подавати його на вхід, що в цілому є складним технічним процесом.

В такому випадку може застосовуватися задача виявлення об’єктів на зображенні, техніка object detection набула великої популярності з появою

згорткових мереж, оскільки дозволяє достатньо точно визначати розташування багатьох об’єктів на зображенні. Даний підхід можна порівняти з локалізацією, коли вона застосовується одразу до всіх можливих об’єктів на зображенні, на яких навчається нейронна мережа. На рис. 2.8 наведено приклад виявлення об’єктів на зображенні з великою кількістю будівель [7].

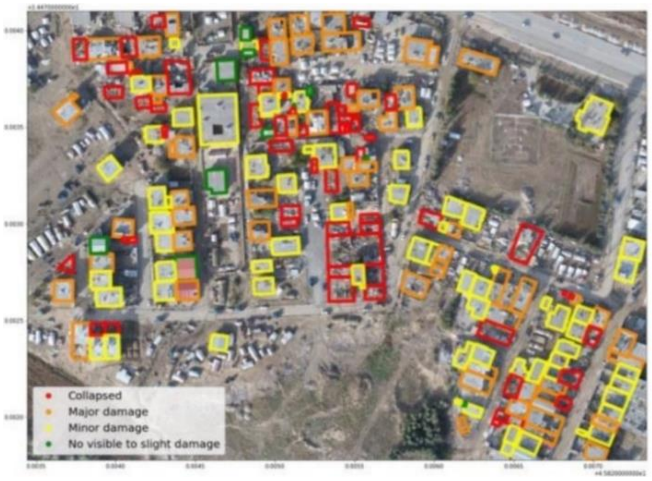

Рисунок 2.8 – Приклад використання техніки object detection для виявлення будинків з різним ступенем пошкодження

Іншою технікою комп’ютерного зору є сегментація – розбиття зображення на деякі зони (сегменти), на відміну від object detection, сегментація відносить кожен піксель зображення до деякого класу. Як і виявлення об’єктів, може поділятися на семантичну сегментацію (англ. semantic segmentation) та сегментацію кожного екземпляру (англ. instance segmentation). В першому випадку йде мова про віднесення всіх об’єктів одного класу до одної групи, фактично – виділення одним кольором всіх об’єктів класу. В той час сегментація кожного екземпляра виділяє кожен об’єкт класу окремо, що може знадобитися в деяких задач, коли існує необхідність відслідковувати кожен об’єкт як унікальний [25].

Сегментація набула поширення в таких сферах, як медицина, де існує необхідність виділення точного силуету досліджуваного об’єкта, тому

класифікація на рівні кожного пікселю є великою перевагою, а також інших – автоматичному керуванні машиною, визначенні доріг на супутникових знімках тощо.

В даній роботі розглядається застосування саме семантичної сегментації, оскільки дана техніка надає широкі можливості при класифікації зон пошкоджених будинків, і може відобразити як ту частину, яка може бути зруйнованою, так і цілу частину будівлі. Створення сегментаційної карти надасть можливість корегувати класифікацію на рівні пікселів, що може покращити результати розпізнавання пошкоджень, а відображення за допомогою теплових карт (англ. heatmap) спростить візуалізацію та інтерпретацію.

2.3 Обробка та аугментація зображень

Згорткові нейронні мережі – потужний інструмент для вирішення задач розпізнавання, але в той час потребує відповідних ресурсів та підходу у застосуванні. Будь-які нейронні мережі, в першу чергу, потребують набору даних, який буде використано для навчання: він повинен бути збалансованим, мати достатню кількість зображень для кожного класу та відповідати іншим статистичним критеріям. Наприклад, набір даних, який буде містити 30 зображень одного класу та 170 іншого, може не підійти для навчання мережі, або потребувати додаткових втручань з боку дослідника.

Таким чином, обробка та підготовка вхідних даних є ключовим моментом в завданнях розпізнавання об’єктів. Одними із найпростіших етапів є вирівнювання зображення до бажаного розміру, в такому випадку можна застосувати методи масштабування до більшого (англ. upscaling) або меншого (англ. downscaling) розміру. Недоліком даної процедури є погіршення якості зображенні, і відповідно, втрата деякої частини даних на рівні пікселів, що в цілому може негативно вплинути на подальше розпізнавання. Іншим способом розв’язання даної проблеми може бути обрання деякого розміру зображення ще на етапі створення набору даних. Дане рішення є оптимальним, якщо на поточний час наявна велика

вибірка даних з однаковими параметрами, або є можливість створити матеріал самостійно.

Розмірність та якість зображень впливає як на результати розпізнавання, так і на швидкість навчання нейронної мережі. Але окрім цих параметрів варто враховувати один із найголовніших – кількість наявних зразків у наборі даних. Глибокі та згорткові нейронні мережі потребують великої кількості матеріалу в датасеті, щоб мати змогу навчатися виявляти патерни та об’єкти на зображеннях. Таким чином, для задач багатокласової класифікації може потребуватися сотні й тисячі зображень для кожного класу, який необхідно розпізнавати.

У випадку, коли розмір датасету не задовільний – виникає проблема недонавчання або перенавчання. Проблема може вирішуватися декількома шляхами: зменшенням або збільшенням складності (глибини) моделі, додавання «вилучення» (dropout) в деякі шари, так і збільшенням кількості зразків у навчальному і тестовому наборах. Якщо в моделі застосовуються зображення високої роздільної здатності та якості, зменшення гіперпараметрів для навчання в більшості випадків буде хибним рішенням, оскільки в такій ситуації мережа не зможе виявляти достатню кількість особливостей, або матиме велику похибку на тестовому наборі [26-27]. В той час, збільшення кількості можна реалізувати за допомогою деяких алгоритмів та операцій над кожним або деякими наявними зразками, тобто провести їх аугментацію.

Існує багато технік, за допомогою яких можна провести змінення зображення для збільшення набору даних, серед них: повертання зображення (англ. flipping), збільшення деякої зони зображення (англ. cropping), зміна властивостей кольору або представлення – робота з насиченням, яскравістю, контрастом тощо. Також до цих технік можна додати трансформацію зображень різними шумами, наприклад за гаусом. Дані аугментації націлені на зміну зображення в просторі або збільшення варіативності шляхом маніпуляцій із кольоровою гамою, що в більшості випадків являється дієвим та достатнім засобом.

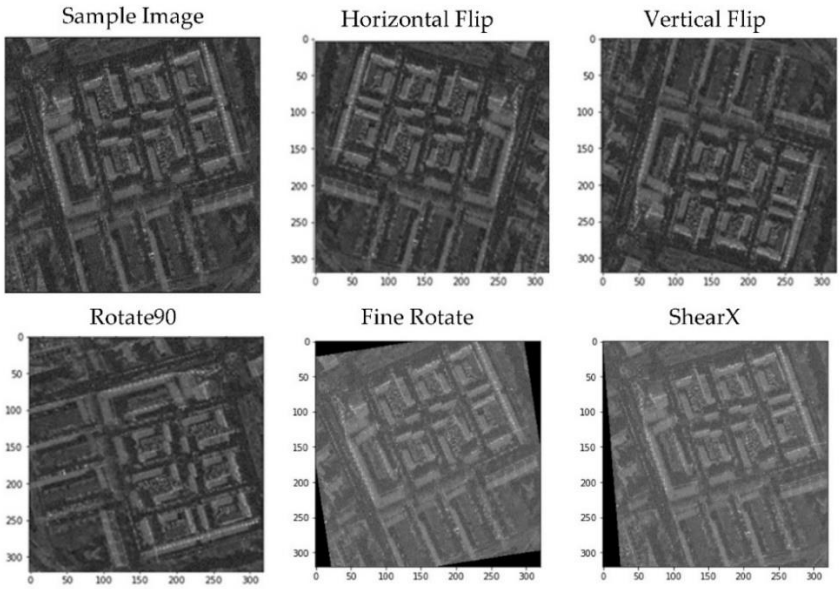

Рисунок 2.9 – Приклад застосування аугментацій

На рис. 2.9 наведені приклади трансформацій із дослідження авторів Варшавського університету технологій, які працювали над дослідженням впливу аугментацій на якість семантичної сегментації на SAR зображеннях [27]. Варто відмітити, що геометричні трансформації можуть бути корисними у випадках, коли вони не змінюють основні властивості та особливості об’єктів на зображеннях, інакше існує ймовірність негативного впливу на подальшу сегментацію.

Окрім геометричних трансформацій та зміни гами зображень, існують таких інші практики, наприклад, використання ефекту перенесення текстури (англ. texture transfer), коли в якості базової текстури використовується обране зображення, а всі інші зображення наслідують його; або внесення змін в значення на рівні кожного пікселю, що можна порівняти із додаванням шуму. Але в такому випадку не відбувається візуальної зміни, тобто внесені коригування важко розпізнати людині, в той час на рівні значень матриці нейромережа може сприймати одне зображення за інше – найчастіше даний спосіб використовується при атаках за допомогою зображень. Використання даної техніки може бути використане для визначення слабких місць навченої мережі [26].

В деяких випадках, коли створення набору даних неможливо через критичну малу кількість зразків, або може бути надто довгим чи дорогим процесом для дослідження, використовуються генеративні мережі або звичайні алгоритми за принципом скопіювати вставити (англ. copy-paste або cut-and-paste) [28-29]. Дані техніки являють собою унікальну можливість створити «з нуля» необхідні зразки для навчання мережі, здебільшого маючи лише елементи для формування зображень: фон та відповідно об’єкти для подальшої класифікації чи сегментації.

На рис. 2.10 наведено приклад процесу генерації зображень за допомогою декількох фотографій та елементів об’єктів, які використовували автори у своїй роботі [29]. Варто відмітити, що даний ефект може бути досягнуто простим застосуванням традиційних технік, як повороти та віддзеркалення, та відповідно різними фоновими зображеннями, використовуючи базові точки для вставки, або випадкові на деякій обраній осі.

Рисунок 2.10 – Використання технік для генерації зображень

Таким чином можна відмітити, що процес обробки та аугментації даних являється одним із ключових для створення необхідного набору для навчання глибоких та згорткових нейронних мереж. Дані техніки дозволяють швидко розширити зразки при їх недостатній кількості, або сформувати власноруч, якщо зображення неможливо отримати самостійно чи процес є дороговартісним.

Недоліком аугментацій можуть бути уповільнення процесу навчання, пошкодження або втрата частки даних, тому потрібно приділяти увагу тим трансформаціям, які не є «природніми» для набору даних, що використовується для вирішення поставленої задачі.

2.4 Архітектура мережі U-Net

U-Net – одна з моделей згорткових мереж, яка була створена для сегментації зображень для завдань біомедицини (наприклад, сегментація клітин за комп'ютерним знімком або легень на рентгенівському знімку). Дана мережа відноситься до повністю згорткових (англ. fully convolutional network, FCN), оскільки, як зазначають автори, вона не містить повнозв'язних шарів. Також варто відмітити іншу особливість архітектури – в залежності від звичайної FCN, U-Net має структуру, схожу на encoder-decoder, про що буде детальніше описано далі.

Архітектура. Як вже зазначалося, автори дослідження Роннебергер та Фішер вирішили взяти за основну повністю згорткову мережу (рис. 2.11), яка в свій час була запропонована Лонгом та Шелхамером [18]. Дана модель не має повнозв'язних шарів, натомість використовує згортання даних в менші шари згортки, після чого застосовується підвищення дискредитації (англ. up-sample), що створює мережу, яка дозволяє прогнозувати значення на рівні пікселів, що в свою чергу має велику перевагу в таких задачах, як сегментація.

Відмінність звичайної FCN від U-Net полягає в тому, що автори останньої внесли деякі зміни, що позитивно вплинули на використання моделі для завдання сегментації: додали додаткові шари на останньому кроці, що підвищило можливість обирати більше особливостей із зображень, а також створювати вихідне зображення однакового розміру; додали використання «пропуску з'єднань» (англ. skip connections) з об'єднанням, що в свою чергу збільшило точність модель для локальних змін та на рівні кожного пікселі, що грає важливу роль в задачах біомедицини, для яких вона створювалася [14].

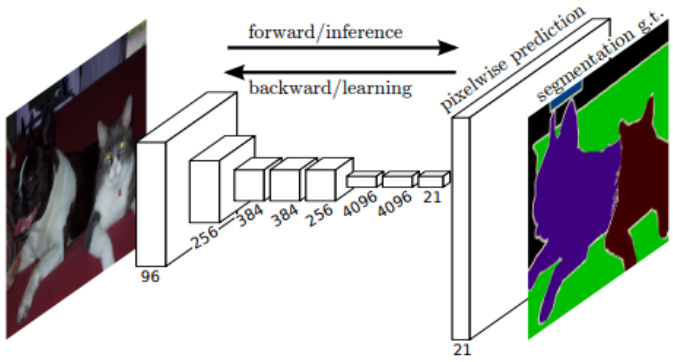

Рисунок 2.11 – Повністю згорткова мережа

Таким чином, виділивши основні частини, можна сформувати наступну архітектуру (рис. 2.12). Як можна побачити, модель складається з двох основних частин: шлях згортки або згодження (англ. contracting path), який іноді називають кодувальним (англ. encoder) та, відповідно, шлях розширення (англ. expanding path) або декодер – коли набір ознак перетворюється на сегментаційну карту за допомогою підвищення дискредитації.

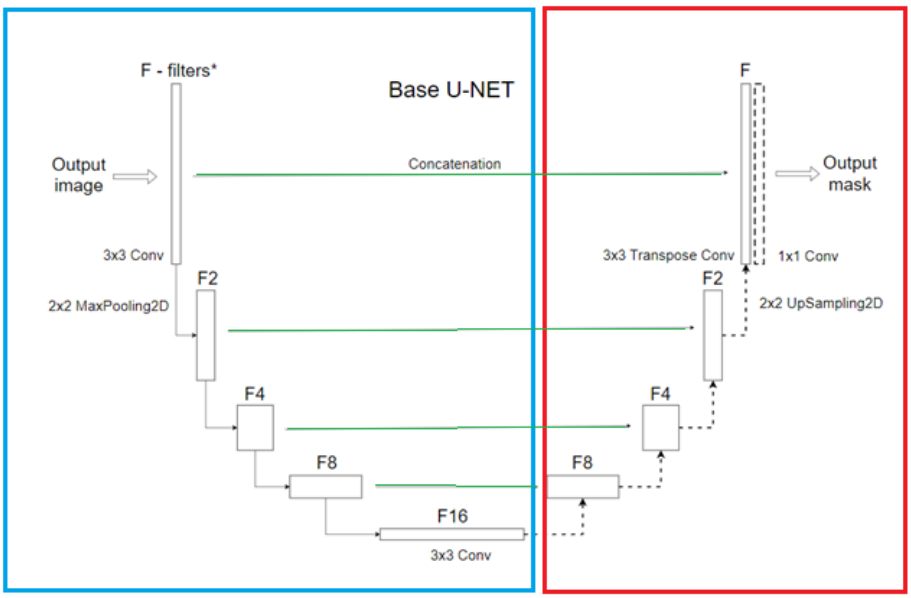

Рисунок 2.12 – Архітектура U-Net, основні елементи виділені кольорами

Застосування «skip connections». На рисунку вище синім виділена частина шляху згортки, червоним – підвищення дискредитації, а також зеленим –

використання конкатенації (об’єднання) разом з пропуском з’єднань. Останній крок є не менш важливим за інші, оскільки він вирішує декілька проблем.

По-перше, при збільшенні кількості шарів в обох частинах – збільшується глибина моделі, при подібній архітектурі модель буде «надавати перевагу» більш загальним особливостям, погіршуючи результати на локальних. Додавання пропуску зв’язків дасть можливість використовувати вже існуючу інформацію з іншого шару, що підвищує можливість моделі розпізнавати патерни та форми об’єктів [14]. По-друге, використання даної техніки дозволяє коригувати значення на рівнів пікселів, що підвищує як загальну точність моделі, так і класифікацію локальних ознак, що в свою чергу позитивно впливає на формування кінцевої сегментаційної маски.

Функції. Стандартна модель використовує функцію softmax у вихідному шарі. Softmax, або нормалізована експоненціальна функція, перетворює вектор дійсних чисел у розподіл ймовірностей можливих результатів, таким чином, вона підходить для задач класифікації, в тому числі сегментації зображень, перетворюючи деякі значення пікселів вихідного масиву на масив ймовірностей класів, загальна сума яких дорівнює 1. Нижче наведена формула функції активації softmax з описом для конкретної задачі:

$$SoftMax(output_c) = \frac{\exp(output_c)}{\sum_{j=1}^{k} \exp(output_j)}, \qquad (2.1)$$

де c – визначений клас у межах від 1 до k;

j – ітерація можливих класів від 1 до k;

output – вихідні дані мережі (зображення);

exp – експонента від деякого набору значень;

Як можна зрозуміти, вихідні значення, без функції активації – являються звичайними числами, які відображають кінцевий результат операцій в шарах

нейронної мережі. Саме для цього застосовуються функції активації, яка у випадку архітектури U-Net та задачі семантичної сегментації, є функцією softmax, що перетворюють набір значень у набір ймовірностей (сума дорівнює одиниці). В свою чергу, ймовірності використовуються для обрахування точності та втрат моделі після кожної епохи навчання.

Однією із поширених функцій витрат, яка використовується у задачах сегментації, є категоріальна (або бінарна у випадку класифікації одного зразка одного класу на зображені) перехресна ентропія (англ. categorical cross-entropy), яка виміряє «відстань» між прогнозованим розподілом та дійсним розподілом класів. Таким чином, чим нижче значення ентропії, тим краще узгодження між прогнозом та дійсністю, в той час як збільшення призводить до погіршення моделі стосовно реального прогнозування результатів. Формула 2.2 відображає застосування функції у випадку багатозразкової класифікації:

$$CCE\_loss = -\sum_{c=1}^{k} ground\_truth_c \bullet \log(SoftMax(output_c)), \qquad (2.2)$$

де c – визначений клас у межах від 1 до k;

output – вихідні дані мережі (зображення);

ground_truth – сегментаційна карта (маска) до зображення;

Використання логарифмів у функції дозволяє «штрафувати» помилкові прогнозування з високим рівнем довіри, що значно покращує кінцеву класифікацію. Варто відмітити, що в оригінальній статті U-Net автори використовують додаткову вагу до функції, що дозволяє частково компенсувати дизбаланс класів та надавати більшого значення для границь сегментації, що грає свою роль в задачах біомедицини [14]. Але оскільки в задачах розпізнавання пошкоджених будівель форма має не таке важливе значення, а дизбаланс можна вирішити іншими шляхами – застосування додаткової ваги використовувати не обов’язково.

Висновки до розділу 2

В другому розділі було розглянуто згорткові нейронні мережі та архітектуру U-Net, наведено приклади та визначено для використання у роботі техніки комп’ютерного зору, а саме семантичну сегментацію, що має переваги в розпізнаванні будівель на зображеннях.

Згорткові нейронні мережі – потужний інструмент комп’ютерного зору, який в свою чергу потребує як вмілого налаштування, так і необхідного розміру набору даних, на якому повинна «навчитися» розпізнавати ознаки об’єктів на зображеннях. Складніша мережа може показати кращий результат, але й потребує більшого об’єму даних та часу для навчання. Таким чином існує необхідність збалансувати архітектурні рішення та вхідні дані, для чого можуть використовуватися аугментації.

Мережа U-Net використовує повністю згорткову нейронну мережу з додатковими шарами на кінцевому етапі (шар розширення), що дозволяє визначати більше локальних ознак, а отже зростає точність класифікації форми об’єкта. В свою чергу, застосування пропуску з’єднань підвищує точність класифікації на рівні пікселів, що покращує кінцевий результат у вигляді сегментаційної маски.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ПОШКОДЖЕНИХ БУДІВЕЛЬ

3.1 Програмні засоби реалізації інтелектуальної системи

Python – інтерпретована об’єктно-орієнтовна мова програмування високого рівня, з наявною динамічною типізацією. Таким чином, дана мова програмування є корисною для швидкої та простої розробки, дозволяє не загострювати увагу на таких деталях, як виділення пам’яті та її очистка, явно вказувати типи даних тощо. Також Python підтримує модулі та пакети, що в свою чергу розмежовують необхідний код, дозволяють підключати необхідні класи та функції, в залежності від їх фактичного використання [30].

Мова програмування має активне товариство розробників, які створюють різноманітні бібліотеки для загального користування, це значно спрощує розробку програмних продуктів, для яких необхідно застосовувати алгоритми та моделі, наприклад, машинного навчання або штучного інтелекту, оскільки можна скористатися вже існуючою реалізацією. Окрім бібліотек користувачів, для цієї мови програмування активно ведеться розробка офіційних програмних рішень та фреймворків, таких як PyQt, Tensorflow, PyTorch, OpenCV тощо. В свою чергу, маючи підтримку різних платформ – Windows, Unix-систем, Mac OS та інших, Python може застосовуватися майже в будь-якому середовищі, що є ідеальним кандидатом для створення інтелектуальних систем та розробки штучного інтелекту.

За необхідності використання локальних або інших баз даних, можна скористатися пакетами таких СКБД (системами керування баз даних) як PostgreSQL, Firebird, Oracle, Microsoft SQL та інші [30]. Для веб-серверів існує інтерфейс шлюзу, використовуючи відповідні бібліотеки, які підтримують це програмне рішення.

В даній роботі розглядається реалізація інтелектуальної системи розпізнавання пошкоджених будівель, з використанням мови програмування

Python 3, а також таких бібліотек та фреймворків як PyQt 5, Tensorflow, Sklearn та Albumentations, про які більш детально наводиться нижче.

Tensorflow. Tensorflow – безкоштовне програмне забезпечення з відкритим кодом, розроблене командою Google Brain [31]. Бібліотека може застосовуватися для будь-яких завдань, наприклад, для побудови моделей штучного інтелекту, їх навчання, застосування алгоритмів глибокого навчання тощо.

Бібліотека надає багато різних рівнів абстракцій та API (англ. Application Programming Interface), кожен з яких використовується для обраних задач та облегшує реалізацію алгоритмів і моделей. Так за допомогою інтерфейсу високого рівня Keras, можна створювати моделі машинного навчання, використовуючи спеціальні класи та методи, які представляють ті чи інші реалізовані алгоритми, методи машинного навчання, шари нейронних мереж, і взаємодіють між собою як будівельні блоки [31]. Tensorflow надає різні налаштування та можливість створювати «комунікаційні лінії» (англ. pipelines) між різними частинами інтерфейсу, що дозволяє швидко будувати потрібні моделі та впроваджувати їх в існуючі проекти, не зважаючи на мову програмування або платформу, яка буде використовуватися при розробці або застосуванні.

Таким чином, Tensorflow та Keras є потужними інструментами для реалізації як нейронної мережі, так і інтелектуальної системи загалом – надаючи готові реалізації алгоритмів, а також дозволяючи швидко та просто створювати, навчати та тестувати власні моделі.

Sklearn. Це інша бібліотека, яка надає інструменти для обчислення та математичних розрахунків, які застосовуються в таких галузях як видобуток даних, аналіз даних, машинне навчання та штучний інтелект тощо [32]. Sklearn має менше можливостей, ніж минула бібліотека, але тим не менш надає корисні засоби для застосування в обраній задачі. В свою чергу, дану бібліотеку можна використовувати для оцінки моделей, застосування додаткових метрик да побудови графічних відображень результатів.

Хоча Tensorflow має власні методи для завантаження та обробки даних, деякі із методів Sklearn також можна використовувати для цих цілей, оскільки бібліотека має вбудовані набори даних, класи-завантажувачі тощо. Це може стати корисним на етапі завантаження даних та їх обробки, а також тестуванні моделі, якщо буде необхідність змінити наявний набір даних, або провести тестування на іншому наборі даних.

Albumentations. Як зазначалося у минулих розділах, аугментації – один із важливих інструментів на етапі обробки та підготовки даних для навчання. Бібліотека Albumentations надає швидкий та потужний інструмент для застосування аугментацій до даних, і використовується для таких задач машинного навчання, як класифікація, сегментація, розпізнавання об’єктів тощо [33].

Дана бібліотека є одним з найкращих інструментів, оскільки в реалізації застосовуються оптимізовані функції з бібліотек OpenCV та NumPy, а розробники проводять регулярні тестування та порівняння з існуючими програмними рішеннями для подальшого вдосконалення. Також Albumentations містить понад 60 видів аугментацій з різними налаштуваннями, дозволяє створювати набори з аугментацій, які будуть застосовуватися згідно обраного правила або вказаною ймовірністю. На рис. 3.1 наведено два види трансформацій для семантичної сегментації.

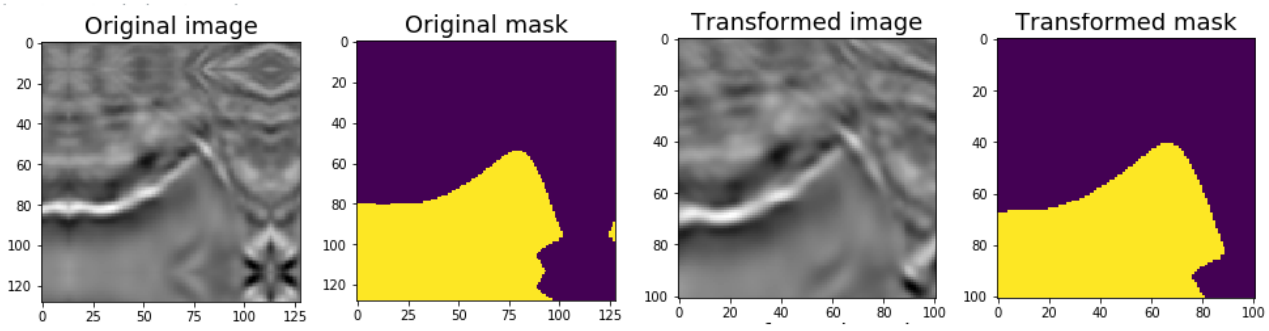

Рисунок 3.1 – Приклад застосованих аугментацій

Бібліотека поєднує гнучкість програмних рішень та масштабування, що дозволяє просто створювати потужні реалізації з використанням багатьох аугментацій. Застосування даної бібліотеки в інтелектуальній системі прискорить «налаштоване спотворення» зображень для етапу навчання та тестування, що дасть змогу навчити нейронну мережу, використовуючи при цьому менше реальних даних. Оскільки бібліотека дозволяє застосовувати одні й ті самі аугментації одразу на вхідне зображення та маску – не потрібно застосовувати декілька методів два рази, а також задумуватися над зберіганням стану аугментацій, оскільки бібліотека застосовує випадкові поєднання.

PyQt. Qt – один із найпотужніших фреймворків для реалізації графічних інтерфейсів, що підтримує різні реалізації та мови програмування, включаючи Python. PyQt 5 містить в собі крос-платформний фреймворк Qt, написаний на мові програмування C++ та безпосередньо вбудований інтерпретатор Python, таким чином реалізуючи набір для реалізації GUI (англ. Graphical User Interface), що підтримує вже наявні методи та класи, що існують в базовому фреймворку [34].

PyQt підтримує всі наявні оригінальні абстракції, роботу із сокетами та пакетами, отже дозволяє реалізовувати багатопотокові програми, які будуть взаємодіяти із графічним інтерфейсом паралельно, не блокуючи його. Також наявна підтримка регулярних виразів, Unicode, підключення до різних баз даних та роботи з файлами SVG, OpenGL, XML тощо, так і фреймворку мультимедіа та вбудованого браузеру [34].

Оскільки Python проста та водночас потужна об’єктно-орієнтовна мова програмування, застосування такого фреймворку як Qt дозволяє швидко та без додаткових зусиль створювати інтерфейси програмних продуктів. Великою перевагою є також, що як і Python, Qt написаний та використовує реалізації на мовах C та C++, що дозволяє безпосередньо створювати пакети, бібліотеки та модулі.

Іншою перевагою є наявність проміжкового програмного забезпечення для побудови графічних інтерфейсів, таких як PyQt Tools та PyQt Designer. PyQt

використовує файли розмітки для поєднання реалізованих абстракцій у коді програми, що дозволяє редагувати GUI як вручну, так і за допомогою окремого інструмента, яким являється PyQt Designer. Це додатковий модуль, який дозволяє розробнику швидко та на інтуїтивному рівні створювати програмний інтерфейс, на рис. 3.2 наведено робоче вікно застосунку.

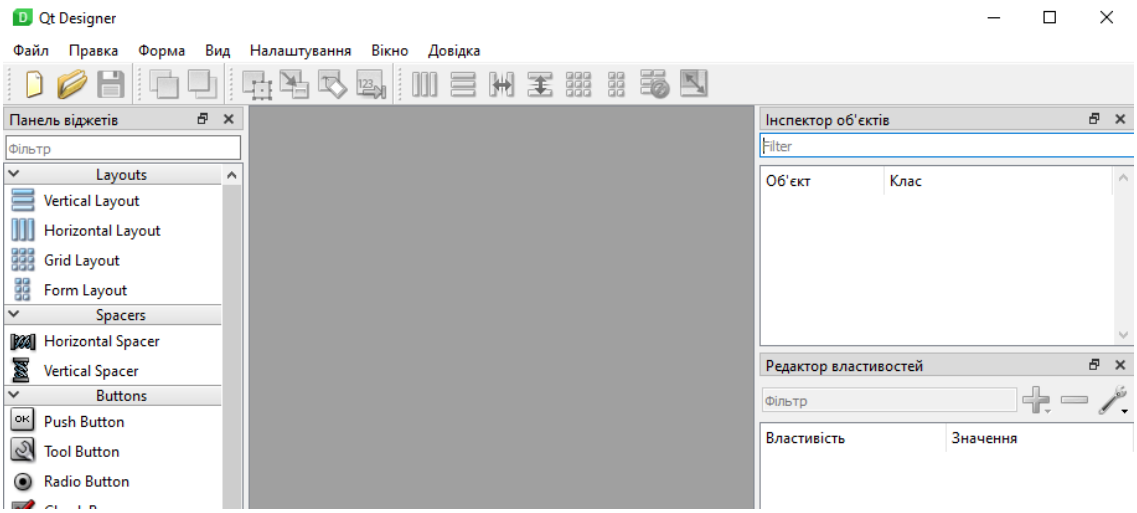

Рисунок 3.2 – Програмне вікно Designer

Таким чином, використання мови програмування Python, доступних бібліотек для машинного навчання та штучного інтелекту, а також інструментів для побудови графічного інтерфейсу, дозволить створити необхідну інтелектуальну систему для виконання поставлених завдань, що буде використовувати реалізовану архітектуру U-Net.

3.2 Збір та підготовка набору даних

Збір даних – один з найголовніших етапів, оскільки від нього залежить навчання та робота побудованої моделі. Щоб сформувати набір даних, використовувалися доступні безкоштовні сервіси, один з яких – Google Earth, що дозволяє отримати зображення майже в будь-якому куточку планети. Політика Google не дозволяє використовувати фотографії на безкоштовній основі для

комерції, але можна для особистого використання, в наукових роботах або статтях за умови вказання джерела даних [35].

На рисунку 3.3 наведено приклад перших зображень, які були зроблені у застосунку Google Earth. Фотографії мають розмір 512 на 512 пікселів, а також були наближені, щоб отримати більш точну та якісну картинку (висота зйомки відповідає приблизно 400 метрів над землею). Всього було обрано 50 зображень із різних районів міста Маріуполь, зроблених супутником в межах травня 2022 року.

Рисунок 3.3 – Перші п’ять фотографій

Задля економії часу та ресурсів, оскільки вони є обмеженими, набір зображень було сформовано здебільшого із будинків, які знаходяться в приватному секторі. Це дасть змогу моделі зосередитися на одному типі будинків, оскільки будівлі промислової зони або багатоповерхівки мають інші параметри та значно відрізняються від приватних будинків – вони більші, створені з інших матеріалів та мають різні сигнатури на зображені. В свою чергу, навчання тільки на приватних будинках дасть змогу створити якісну модель, після чого можна буде проводити навчання на інших будівлях, або на всіх – в тому випадку, коли будуть доступно більше потужностей, так як тоді набір даних збільшиться в десятки, а можливо і сотні разів.

Оскільки якість зображень в цілому не є високою, було реалізовано метод, для часткового покращення візуального відображення. Було відкинуто використання «важких» сторонніх бібліотек або сервісів з використанням

штучних нейронних мереж, замість цього застосовано звичайне фільтрування та розмиття за гаусом за допомогою бібліотеки OpenCV (рис. 3.4).

Даний метод застосовує невелике ядро із заданими значеннями у якості фільтру, після чого відбувається розмиття за гаусом невеликої області. Таким чином зображення стає «гострішим», але в той час не втрачає своєї реалістичності, оскільки це важливо для реального застосування.

```
def enhance_image(self, image_name, results):
    origin_path = self.origin_folder + image_name
    origin_image = cv2.imread(origin_path)

    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    sharpened_image = cv2.filter2D(origin_image, -1, kernel)

    result_image = cv2.GaussianBlur(sharpened_image, ksize: (3, 3), sigmaX: 0.4)
    result_image_path = self.enhanced_folder + image_name
    cv2.imwrite(result_image_path, sharpened_image)
```

Рисунок 3.4 – Метод покращення якості зображення

Наступним кроком було використання додаткового інструменту для створення сегментаційних карт, або як їх часто називають – масок зображень. Маска – це аналогічне зображення, або масив (матриця) з даними, де кожен піксель має значення, яке в свою чергу відповідає деякому класу. Для створення подібних зображень було застосовано бібліотеку LabelMe, яка має реалізацію на Python [36].

Для створення повноцінного датасету було спочатку зібрано всі зображення в одну папку, і створено додаткові – для покращених зображень та вихідних даних. Вручну було виділено будинки та збережено в конфігураційний файл, після чого реалізовано метод, який автоматизує процес формування сегментаційних карт (рис. 3.5).

```python
def label_dataset(self):
    images_list = os.listdir(self.enhanced_folder)
    for index, img in enumerate(images_list):
        if img.endswith(".png"):
            name = img[:-4]
            file = f'{self.enhanced_folder}{img[:-4]}.json'
            print(name, file)
            # labelme_export_json apc2016_obj3.json -o apc2016_obj3_json
            subprocess.run(["labelme_export_json", file, "-o", self.labeled_folder + name])
```

Рисунок 3.5 – Реалізація методу створення маски із наявних зображень

Наступним кроком було створено метод, який переміщує необхідні зображення з однієї папки в іншу, що дозволяє автоматизовано сформувати датасет із зображень, що розміщуються в різних папках (рис. 3.6). Цей процес не є обов'язковим, але значно скорочує час внесення змін в набір, якщо це буде необхідно.

```python
def relocate_images(self):
    images_folder = os.listdir(self.labeled_folder)
    for index, folder in enumerate(images_folder):
        print(index, folder)
        old_name = 'label.png'
        old_folder = f"{self.labeled_folder}{folder}\\"
        print(f"{old_folder}{old_name}")
        print(f"{self.segmented_folder}{folder}.png")

        subprocess.run( args: ["copy", f"{old_folder}{old_name}",
                               f"{self.segmented_folder}{folder}.png"],
                        shell=True)
```

Рисунок 3.6 – Метод зберігання зображень в іншу папку

Таким чином, було створено декілька класів із методами, що частково автоматизують створення набору даних із супутниковими зображеннями, що містять будівлі. Набір даних містить 100 зображень, на кожному з яких зображено як мінімум один об'єкт, що належить до кожного класу. Всього існує три класи: normal (візуально неушкоджена будівля) – зеленого кольору, damaged (будівля, що має ушкодження) – червоного, та background – фонове зображення, що має

чорний колір. Рисунок 3.7 містить приклад початкового, обробленого та сегментаційного зображення.

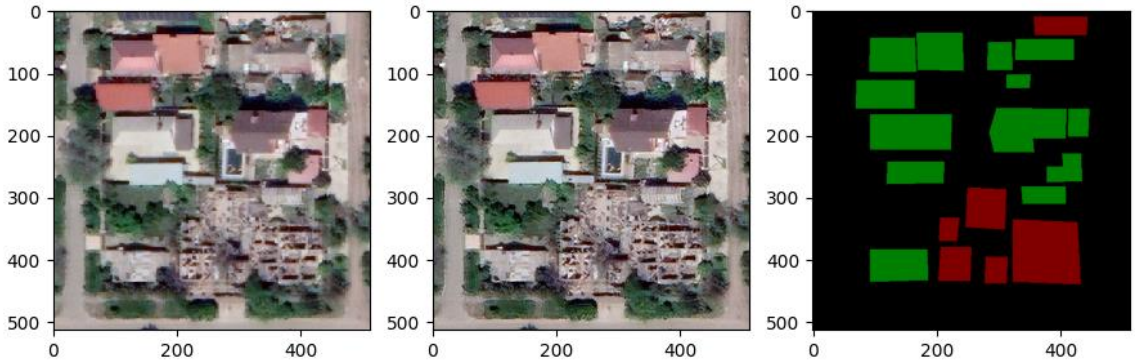

Рисунок 3.7 – Приклад вхідного зображення, обробленого та його маски

Сформований набір даних містить початкові зображення, зображення з обробкою, сегментаційні маски, а також конфігураційні файли бібліотеки LabelMe у форматі json, що дозволяє за необхідності змінити розташування будівель, або інші параметри сегментаційної маски. Даний датасет можна використовувати при навчанні моделі в задачах сегментації, наступним кроком є створення нейронної мережі з архітектурою U-Net та навчання на створеному наборі даних.

3.3 Побудова та навчання нейронної мережі

Оскільки навчання моделей штучних нейронних мереж потребує великої обчислювальної потужності, було вирішено перенести створення та навчання моделі в хмарне рішення, таке як Google Colaboratory (або просто Google Colab). Це проект від компанії Google, який дозволяє використовувати так звані блокноти для написання та запуску скриптів Python, використовуючи надані серверні потужності. На безкоштовній основі можна використовувати потужності процесору без обмеження за часом. Для звичайних користувачів (безоплатно) можна використовувати відеокарту Nvidia Tesla T4 приблизно 12 годин, або коли буде пройдено ліміт по ресурсам [37].

Для початку роботи було зроблено вхід у акаунт Google, створено пустий проект з розширенням .ipynb (рис. 3.8), який використовує Colab. В свою чергу сервіс має багато налаштувань і можливостей, але при розробці можна зосередитися лише на основних – це можливість додавати частини тексту – для опису та заголовків проекту, та коду, який можна запустити одразу в браузері. Плюс використання даного рішення також в тому, що немає необхідності встановлювати спеціальні IDE, а також завантажувати бібліотеки – після запуску необхідного коду з імпортуванням, в проект буде завантажено вказані бібліотеки на сервер Google, що в свою чергу економить як час, так і місце.

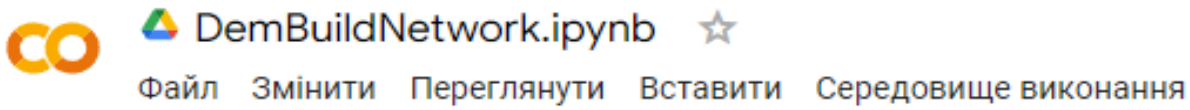

Рисунок 3.8 – Заголовок та меню створеного проекту

Наступний крок для виконання поставленого завдання – завантажити набір даних, який був розроблений в минулому підрозділі. Це можна зробити двома способами: завантаження «вручну» в папку проекту, або використати інше хмарне рішення від Google – Google Drive, тобто Google-диск, що дозволяє користувачеві зберігати власні файли та дані. Диск від компанії є більш зручним та надає більше можливостей, тому було обрано саме його. Після підключення диску, використавши той самий акаунт, було виконано підключення диску до проекту наступним чином (рис. 3.9)

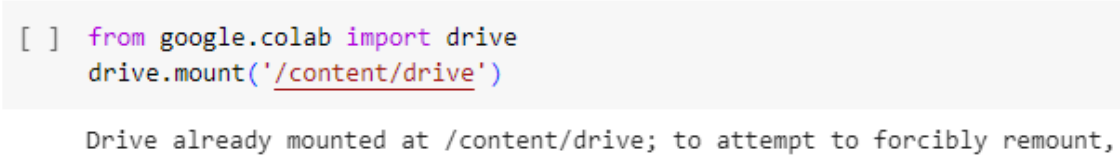

Рисунок 3.9 – Монтування диску

Для того, щоб швидко використовувати необхідний шлях до даних, було винесено основні змінні (рис. 3.10).

```
import os

# images_v2_new
path = '/content/drive/MyDrive/Datasets/images_v2_new'
path1 = f'{path}/enhanced/'
path2 = f'{path}/segmented/'

print(path1, path2)
```

```
/content/drive/MyDrive/Datasets/images_v2_new/enhanced/ /content/drive/I
```

Рисунок 3.10 – Збереження шляхів до даних

Таким самим чином було створено блок коду для імпортування необхідних бібліотек, наприклад, tensorflow, numpy, matplotlib тощо. Оскільки існує необхідність працювати із набором даних напряму з папок проекту, було реалізовано відповідну функцію load_data (рис. 3.11). Функція завантажує список зображень з відповідних змінних, де збережено шляхи до набору даних оригінальних зображень та їх масок. Це було реалізовано, оскільки немає необхідності наперед завантажувати всі зображення.

```
[ ]  def load_data(path1, path2):
         image_dataset = os.listdir(path1)
         mask_dataset = os.listdir(path2)

         orig_img = []
         mask_img = []
         for file in image_dataset:
             orig_img.append(file)
         for file in mask_dataset:
             mask_img.append(file)

         orig_img.sort()
         mask_img.sort()

         return orig_img, mask_img
```

Рисунок 3.11 – Функція завантаження зображень набору даних

Оскільки в роботі будуть використовуватися аугментації, було реалізовано трансформації з бібліотеки Albumentations за допомогою класу Compose. Він дозволяє створювати потужні комбінації з різних доступних трансформацій; оскільки багато з них сильно деформують об’єкти, було обрано як основі – ShiftScaleRotate, що реалізовує поворот, зсув та збільшення зображення, а також в одному з наборів трансформацій було додано MotionBlur – розмиття (рис. 3.12).

```
transforms_v2 = Compose([
            A.OneOf([
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.3,
                                       scale_limit=0.1,
                                       rotate_limit=270,
                                       p=0.7),
                    # A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1, hue=0.1, p=0.5),
                ]),
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.2,
                                       scale_limit=0.05,
                                       rotate_limit=90,
                                       p=0.5),
                    A.MotionBlur(blur_limit=3, p=0.3),
                    # A.GaussNoise(var_limit=(1, 2), p=0.4)
                ]),
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.2,
                                       scale_limit=0.15,
                                       rotate_limit=180,
                                       p=0.8),
```

Рисунок 3.12 – Використані набори трансформацій

Таким чином, при використанні трансформації буде обрано, та використано з заданою ймовірністю) один з трьох наведених наборів, та застосовано аугментацію з випадковими параметрами в межах заданих.

На рис. 3.13 наведено реалізований метод, який створений для застосовування аугментації для кожного зображення в наборі задану кількість ітерацій. Таким чином можна розширити існуючий набір в декілька разів, що часто може покращити точність моделі, оскільки збільшується варіантивність даних.

```python
def do_augmentation(img, mask, target_shape_img, target_shape_mask, iterations):
    length = len(img)
    iterations += 1
    m = length * iterations
    i_h, i_w, i_c = target_shape_img
    m_h, m_w, m_c = target_shape_mask

    X = np.zeros((m, i_h, i_w, i_c), dtype=np.float32)
    y = np.zeros((m, m_h, m_w, m_c), dtype=np.int32)

    counter = 0
    for i in range(0, iterations):
        for j in range(0, length):
            if i == 0:
                X[counter] = img[j]
                y[counter] = mask[j]
            else:
                augmented = transforms_v2(image=img[j], mask=mask[j])
                X[counter] = augmented["image"]
                y[counter] = augmented["mask"]
            counter += 1
```

Рисунок 3.13 – Метод для розширення набору даних

Щоб завершити частину із підготовкою та попередньою обробкою даних, було створено метод, який відповідатиме за підготовку зображень набору, що будуть використовуватися при навчанні (рис. 3.14).

Оскільки модель працює як з основним зображенням, так і з маскою – необхідно, по-першу, завантажити пару зображень в пам’ять, після чого змінити розмір зображень на той, що буде використовуватися при навчанні. Для зручності було обрано динамічний розмір, який можна буде змінювати в ході дослідження. По-друге, для «нормалізації» даних було поділено значення кожного пікселя в RGB-спектрі на 256, отримавши значення від 0 до 1, що в свою чергу прискорить та спростить навчання.

```python
for file in img:
    index = img.index(file)
    path = os.path.join(path1, file)
    single_img = Image.open(path).convert('RGB')
    single_img = single_img.resize((i_h, i_w))
    single_img = np.reshape(single_img, (i_h, i_w, i_c))
```

Рисунок 3.14 – Головна частина реалізованого методу для попередньої обробки зображень набору

Тепер необхідно побудувати модель згорткової нейронної мережі U-Net. Загалом архітектура моделі складається з наступних елементів: енкодеру та декодеру, також можна виділити як окремий елемент проміжне місце між ними, що є ніби найвужчим в архітектурі – «шийку» (англ. bottleneck). На рис. 3.15 наведено базові елементи – блок згортки та блок енкодеру, який складається відповідно з цього блоку, а також зниження дискретизацію та «скиду».

```
def conv2d_block(input_tensor, n_filters, kernel_size=3):
    x = input_tensor
    for i in range(2):
        x = tf.keras.layers.Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), \
                                   kernel_initializer='he_normal', padding='same')(x)
        x = tf.keras.layers.Activation('relu')(x)

    return x
def encoder_block(inputs, n_filters=64, pool_size=(2, 2), dropout=0.3):
    f = conv2d_block(inputs, n_filters=n_filters)
    p = tf.keras.layers.MaxPooling2D(pool_size=pool_size)(f)
    p = tf.keras.layers.Dropout(dropout)(p)

    return f, p
```

Рисунок 3.15 – Блок згортки та енкодеру

Як можна помітити, блок згортки складається з двох згорткових шарів та двох шарів активації, які реалізовані в надбудові Keras бібліотеки Tensorflow. Таким чином, використовуючи блок енкодеру можна побудувати загальний шар (рис. 3.16), який містить 4 відповідних блоків з різними параметрами: кожен блок збільшується кількість фільтрів вдвічі, розмір фільтру 2 на 2, значення скиду можна задавати при визначені моделі. Частина bottleneck містить окремий блок згортки з максимальним розміром фільтрів.

```
def encoder(inputs, n_filters, dropout):
    f1, p1 = encoder_block(inputs, n_filters=n_filters, pool_size=(2, 2), dropout=dropout)
    f2, p2 = encoder_block(p1, n_filters=n_filters * 2, pool_size=(2, 2), dropout=dropout)
    f3, p3 = encoder_block(p2, n_filters=n_filters * 4, pool_size=(2, 2), dropout=dropout)
    f4, p4 = encoder_block(p3, n_filters=n_filters * 8, pool_size=(2, 2), dropout=dropout)

    return p4, (f1, f2, f3, f4)

def bottleneck(inputs, n_filters):
    bottle_neck = conv2d_block(inputs, n_filters=n_filters * 16)

    return bottle_neck
```

Рисунок 3.16 – Елементи енкодер та «шийка»

За схожою логікою було побудовано елементи декодеру, а саме блок декодеру, що складається з шару підвищення дискретизації, поєднання (конкатенації) та «скиду» (рис. 3.17), та відповідно головний блок, що складається з 4 підблоків (рис. 3.18).

```
def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
    u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides=strides, padding='same')(inputs)
    c = tf.keras.layers.concatenate([u, conv_output])
    c = tf.keras.layers.Dropout(dropout)(c)
    c = conv2d_block(c, n_filters, kernel_size=3)

    return c
```

Рисунок 3.17 – Блок декодеру

Як можна побачити, головний блок декодеру також містить вихідний шар, що містить функцію активації Softmax, таким чином перетворюючи значення кожного пікселю на ймовірності для існуючих класів.

```
def decoder(inputs, n_filters, convs, output_channels, dropout):
    f1, f2, f3, f4 = convs

    c6 = decoder_block(inputs, f4, n_filters=n_filters * 8, kernel_size=(3, 3), strides=(2, 2), dropout=dropout)
    c7 = decoder_block(c6, f3, n_filters=n_filters * 4, kernel_size=(3, 3), strides=(2, 2), dropout=dropout)
    c8 = decoder_block(c7, f2, n_filters=n_filters * 2, kernel_size=(3, 3), strides=(2, 2), dropout=dropout)
    c9 = decoder_block(c8, f1, n_filters=n_filters, kernel_size=(3, 3), strides=(2, 2), dropout=dropout)

    outputs = tf.keras.layers.Conv2D(output_channels,
                                     (1, 1),
                                     activation='softmax'
                                     )(c9)

    return outputs
```

Рисунок 3.18 – Основний декодер

Після створення всіх необхідних блоків, можна сформувати модель мережі U-Net, результат наведено на рис. 3.19. Таким чином існує вхідний шар, шари енкодеру та декодеру, які складаються з відповідних підблоків, та «шийка», яка поєднує ці два блоки між собою.

```
def unet(n_filters, shape, dropout):
    inputs = tf.keras.layers.Input(shape=shape)

    encoder_output, convs = encoder(inputs, n_filters, dropout)

    bottle_neck = bottleneck(encoder_output, n_filters)

    outputs = decoder(bottle_neck, n_filters, convs, output_channels=OUTPUT_CHANNELS, dropout=dropout)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```

Рисунок 3.19 – Побудована кінцева модель

Модель приймає декілька параметрів, а саме кількість фільтрів – що напряму впливає на глибину та складність моделі, збільшуючи чи зменшуючи загальну кількість параметрів для навчання у блоках, форму – тобто вхідний розмір зображення (для навчання та розпізнавання відповідно), а також значення «скиду», що загалом оптимальне в межах 0.2 та 0.4.

Перед навчанням було завантажено набір за допомогою реалізованих функцій, проведено покращення набору (див. рис. 3.4) та відповідно переглянуто декілька зображень для пересвідчення правильної роботи функцій. На рис. 3.20 наведено приклад основного зображення та його маски, що представлені за допомогою бібліотеки matplotlib.

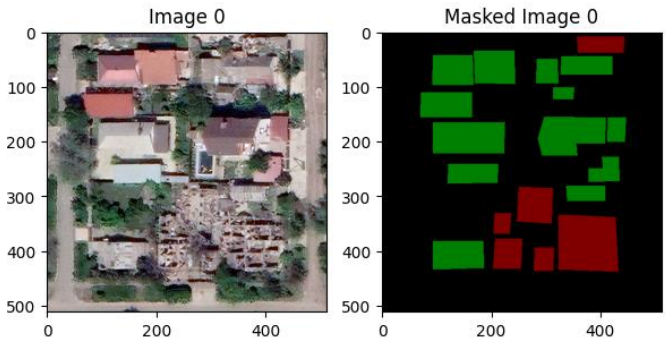

Рисунок 3.20 – Оригінальне зображення та маска

Відповідно, було проведено попередню обробку за іншою розробленою функцією, і теж виведено для ознайомлення (рис. 3.21). Для попереднього тестування побудованої моделі було використано зображення з розміром 256 на

256 пікселів, справа від оригінального зображення наведено маску, яку вже перетворено у матрицю, яку можна подавати на вхід моделі.

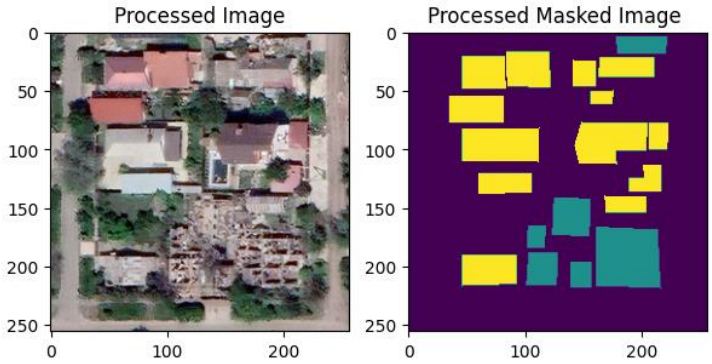

Рисунок 3.21 – Зображення та маска вхідного набору для подавання на вхід побудованої згорткової мережі

Було вирішено використати декілька метрик, які найближче пояснюють якість моделі на рівні пікселів, а також розпізнавання для кожного класу, що складно досягнути звичайними, такими як перехресна ентропія чи простою точністю.

На рис. 3.22 наведено власну реалізацію метрики MeanIoU (англ. Intersection over Union, IoU), з використанням реалізованої функції надбудови Keras, оскільки в поточній реалізації виникає помилка в обрахуванні, що унеможливлює використання. На рис. 3.23 наводиться візуальна інтерпретація метрики IoU та формула для її розрахунку.

```
class UpdatedMeanIoU(tf.keras.metrics.MeanIoU):
  def __init__(self,
               y_true=None,
               y_pred=None,
               num_classes=None,
               name=None,
               dtype=None):
    super(UpdatedMeanIoU, self).__init__(num_classes=num_classes, name=name, dtype=dtype)

  def update_state(self, y_true, y_pred, sample_weight=None):
    y_pred = tf.math.argmax(y_pred, axis=-1)
    return super().update_state(y_true, y_pred, sample_weight)
```

Рисунок 3.22 – Клас метрики UpdatedMeanIoU

Рисунок 3.23 – Формула та візуальна інтерпретація метрики

Як можна побачити, дана метрика являє собою співвідношення частини перетину та загальної площі двох (або більше) класів. При навчанні моделі буде використовувати декілька таких метрик.

Відповідно, перед початком навчання було розділено набір даних на дві частини – тренувальну та для проведення тестування (рис. 3.24). Було вирішено використати співвідношення 30 на 70, в рамках тестування використано параметр random_state, який кожен раз зберігає один і той самий стан для розподілу даних.

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3, random_state=1)
print(f"Train size {len(X_train)} Test size {len(X_valid)}")

X_train, y_train = do_augmentation(X_train, y_train,
                                   target_shape_img,
                                   target_shape_mask,
                                   10)
X_valid, y_valid = do_augmentation(X_valid, y_valid,
                                   target_shape_img,
                                   target_shape_mask,
                                   10)

N_FILTERS = 32
INPUT_SHAPE = (SIZE, SIZE, 3)
```

Рисунок 3.24 – Частина коду для підготовки до тренування моделі

Також застосовується реалізований метод для проведення аугментацій, було вирішено попереднє тестування моделі проводити з використанням десятикратного розміру набору даних, як для набору тренування так і тестування. Таким чином кількість зображень в тренувальному наборі буде сягати 350, і для проведення тестів – 150.

Останнім кроком необхідно використати побудовану мережу з обраними параметрами та скомпілювати, обравши необхідні метрики та функцію втрат. На рис. 3.25 наведено код для тренування моделі.

```
unet = unet(n_filters=N_FILTERS, shape=INPUT_SHAPE, dropout=0.3)
unet.summary()

unet.compile(optimizer=tf.keras.optimizers.Adam(),
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy',
                       UpdatedMeanIoU(num_classes=3),
                       UpdatedIoU(num_classes=3, target_class_ids=[1]),
                       UpdatedIoU(num_classes=3, target_class_ids=[2])])

weights_v1 = {0: 1, 1: 3, 2: 2}

results = unet.fit(X_train, y_train,
                   class_weight=weights_v1,
                   batch_size=15, epochs=25,
                   validation_data=(X_valid, y_valid))
```

Рисунок 3.25 – Код для побудови та початку тренування моделі

Побудована для попереднього тестування модель складається з 8630563 загальних та параметрів для тренування, що в свою чергу є досить «легкою» версією. Це було досягнуто меншою кількістю фільтрів (множник N_FILTERS на рис. 3.25, який дорівнює 32 в даному випадку).

Для проведення тренування було використано функцію втрат SparseCategoricalCrossentropy, яка є реалізацією перехресної ентропії для декількох класів, в свою чергу використано параметр from_logits=False, оскільки при побудові моделі використовується функція Softmax. Параметри для тренування обрані наступні: ваги для класів – збільшено важливість класу пошкоджених будинків на 3 та не пошкоджених на 2, оскільки більшу частину зображення займає фон; розмір пачки для навчання – 15, кількість епох до закінчення – 25, що є малим значенням, але було враховано також невеликий набір даних та складність моделі. На рис. 3.26 наведено результати.

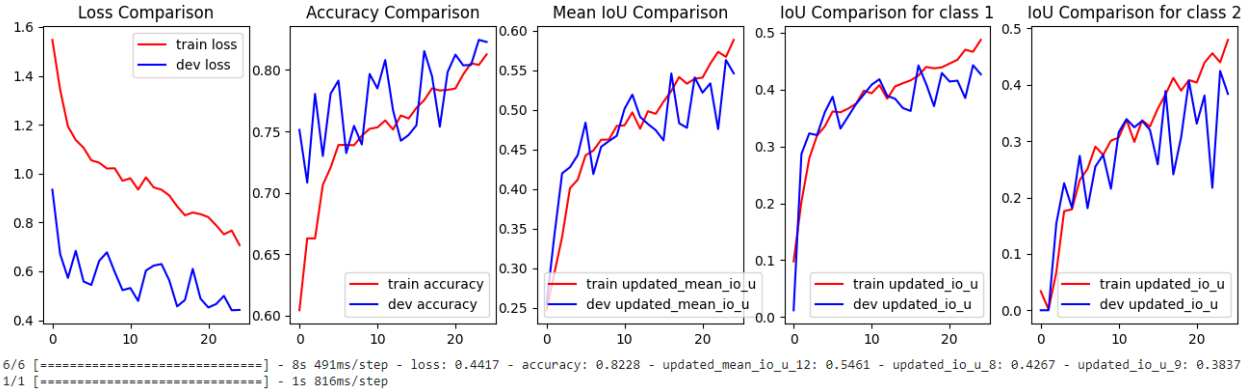

Рисунок 3.26 – Результати попереднього тестування моделі U-Net

Згідно з результатів навчання та проведеного тестування, модель має загальну точність 82.28%, що є гарним результатом, але ця метрика враховує фон в розрахунках, що не відображає реальні результати точності для класів будинків. Використана метрика MeanIoU має значення 0.5884 для трьох класів на навчальному наборі та 0.5461 на тестовому, в свою чергу звичайне IoU для класу зруйнованих будівель 0.4872 і 0.4267, для будинків без пошкоджень 0.4417 та 0.3837 відповідно.

На рис. 3.27 наведено деякі результати на вибірці для тестування, на зображеннях наведено оригінальне зображення та маску, та відповідну згенеровану маску мережею, яка містить ймовірнісне значення класів, до яких належить кожен піксель.

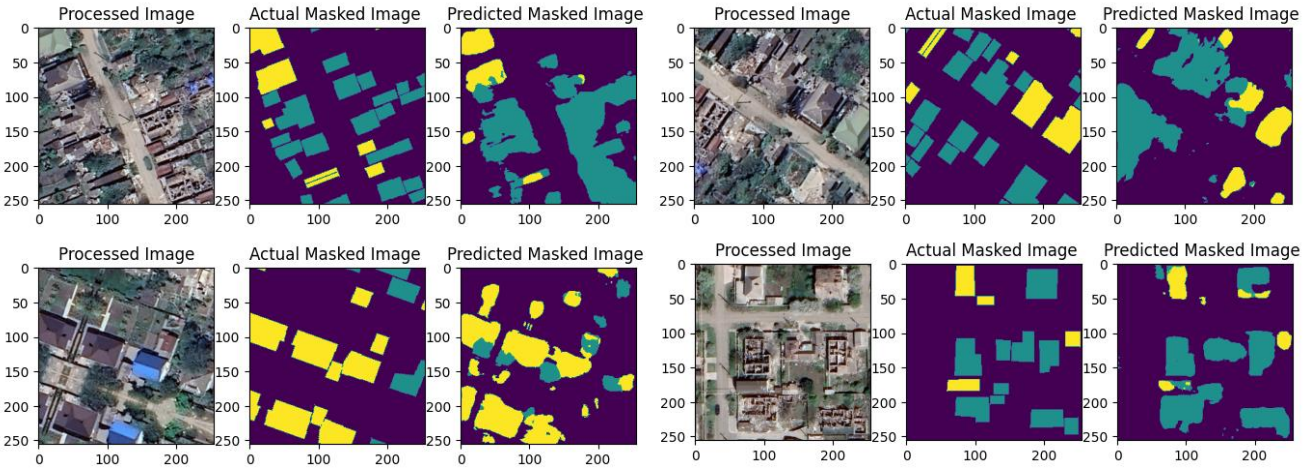

Рисунок 3.27 – Візуальне порівняння результатів

Враховуючи різні показники, які були отримані після навчання, можна зробити висновок, що модель в цілому справляється із задачею розпізнавання, хоча містить відповідні похибки. Це можна пояснити простотою використаної моделі, невеликим розміром набору даних та маркуванням – будівля, яка маркується як та, що має руйнування, може мати незруйновану частину.

3.4 Реалізація інтелектуальної системи та графічного інтерфейсу

Для початку розробки графічного інтерфейсу та системи в цілому, було створено порожній проект PyQt у встановленому окремому застосунку Designer. Designer – середовище побудови графічних інтерфейсів, містить більшість готових елементів, таких як кнопки, поля для вводу даних або для відображення графічних матеріалів.

На рис. 3.28 можна побачити приклад файлу menu.ui, що містить xml-розмітку з даними елементів, які містяться у графічному інтерфейсів. Елементи розміщуються поступово у вигляді «дерева», яке надає доступ бібліотеці для подальшої взаємодії. Для того, щоб почати використовувати інтерфейс та його елементи, необхідно створити окремий клас, що буде відповідати за окремі дії користувача (рис. 3.39).

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1061</width>
    <height>606</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
```

Рисунок 3.28 – Розмітка графічного інтерфейсу

Клас UI містить метод ініціалізації, де вказуються різні дані, в тому числі використання файлу із розміткою графічного інтерфейсу за допомогою методу uic.loadUi(). На рис. 3.29 вказано частину реалізації даного класу.

```
class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()
        uic.loadUi( uifile: "gui/menu.ui", self)

        self.model = None
        self.size = (512, 512)

        title = "Damaged Buildings Detector"
        self.setWindowTitle(title)
```

Рисунок 3.29 – Клас графічного інтерфейсу

Для взаємодії з елементами інтерфейсу було реалізовано декілька методів. На рис. 3.30 наведено метод для обирання користувачем файлу. Оскільки модель має чіткі параметри та розмір зображення для використання, при обранні фотографії її розмір припасовується згідно параметрів обраної моделі.

```
def choose_file_event(self):
    path = os.path.abspath('test_images/')
    file_name, text = QFileDialog.getOpenFileName(self, 'Open file',
                                        path, "Image files (*.jpg *.png)")
    self.path_to_file_edit.setText(file_name)

    image = QPixmap(file_name)
    image = image.scaled(self.original_image_label.width(), self.original_image_label.height())

    self.original_image_label.setPixmap(image)
```

Рисунок 3.30 – Метод обрання фотографії в застосунку

Також було реалізовано схожий метод, який визначає папку для зберігання зображень (рис. 3.31). Це необхідно, якщо користувач має бажання завантажувати дані в обране місце, а не корінь проекту чи загальну папку завантажень.

```
def choose_folder_event(self):
    folder_name = QFileDialog.getExistingDirectory(self, 'Select Folder')
    self.path_to_folder_edit.setText(folder_name)
```

Рисунок 3.31 – Реалізація збереження шляху до обраної папки

Наступним кроком було реалізовано метод для обрання моделі, яка буде використовуватися (рис. 3.32). Користувач може обирати будь-яку папку, де зберігаються файли моделі з розширенням .h5. Також в даному методі ініціалізуються необхідні змінні, що пов’язані з характеристиками моделі.

```
def choose_model_event(self):
    path = os.path.abspath('models/')
    model_name, text = QFileDialog.getOpenFileName(self, 'Open file',
                                        path, "Model files (*.h5)")
    self.path_to_model_edit.setText(model_name)

    self.model = keras.models.load_model(self.path_to_model_edit.text(),
                                         custom_objects={'UpdatedMeanIoU': UpdatedMeanIoU,
                                                         'UpdatedIoU': UpdatedIoU})
```

Рисунок 3.32 – Метод вибору моделі

Останній метод, що є найважливішим – відповідає за сегментацію обраного зображення з використанням обраної моделі (рис. 3.33). При завантаженні у пам’ять зображення попередньо оброблюється та подається у метод predict().

```
def start_segmentation_event(self):
    img = Image.open(self.path_to_file_edit.text()).convert('RGB')

    img = img.resize((self.size[0], self.size[1]))
    img = np.reshape(img,  newshape: (self.size[0], self.size[1], 3))
    img = img / 256.

    test = img[np.newaxis, ...]
    pred_y = self.model.predict(test)
    pred_mask = tf.argmax(pred_y[0], axis=-1)
    pred_mask = pred_mask[..., tf.newaxis]
```

Рисунок 3.33 – Метод сегментації зображення

Після того, як модель опрацювала зображення та вихід було подано матрицю з даними, серед ймовірностей обирається максимальне значення для обрання ймовірного класу, до якого належить кожен піксель. На рис. 3.34 наведено продовження реалізації методу.

```
img_name = 'result {:%Y-%m-%d %H-%M-%S}.png'.format(datetime.now())
segmented_path = self.path_to_folder_edit.text() + '/' + img_name

tf.keras.utils.save_img(segmented_path, pred_mask[:, :, :], scale=True)

image = QPixmap(segmented_path)
image = image.scaled(self.original_image_label.width(), self.original_image_label.height())

self.seg_image_label.setPixmap(image)
```

Рисунок 3.34 – Збереження та відображення згенерованої сегментаційної карти

Таким чином було реалізовано основний функціонал інтелектуальної системи, яка використовує побудовано модель згорткової мережі U-Net. На рис. 3.35 наведено побудований інтерфейс програми. Для початку роботи необхідно обрати модель для подальшого використання, папку (шлях) для зберігання зображень та, відповідно, обрати вхідне зображення.

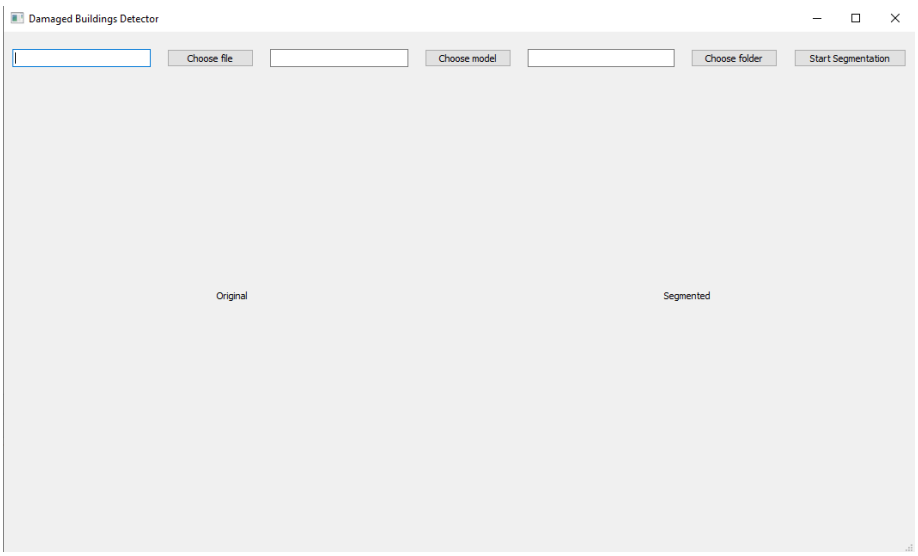

Рисунок 3.35 – Початковий стан програми

Програма має простий та зрозумілий інтерфейс. Після виконання всіх дій, необхідно натиснути на кнопку «Start Segmentation», після чого застосунок одразу виведе зображення, на якому виділені сегменти, які нейронна мережа зарахувала до зруйнованих будинків, та тих, що не отримали ушкоджень (рис. 3.36).

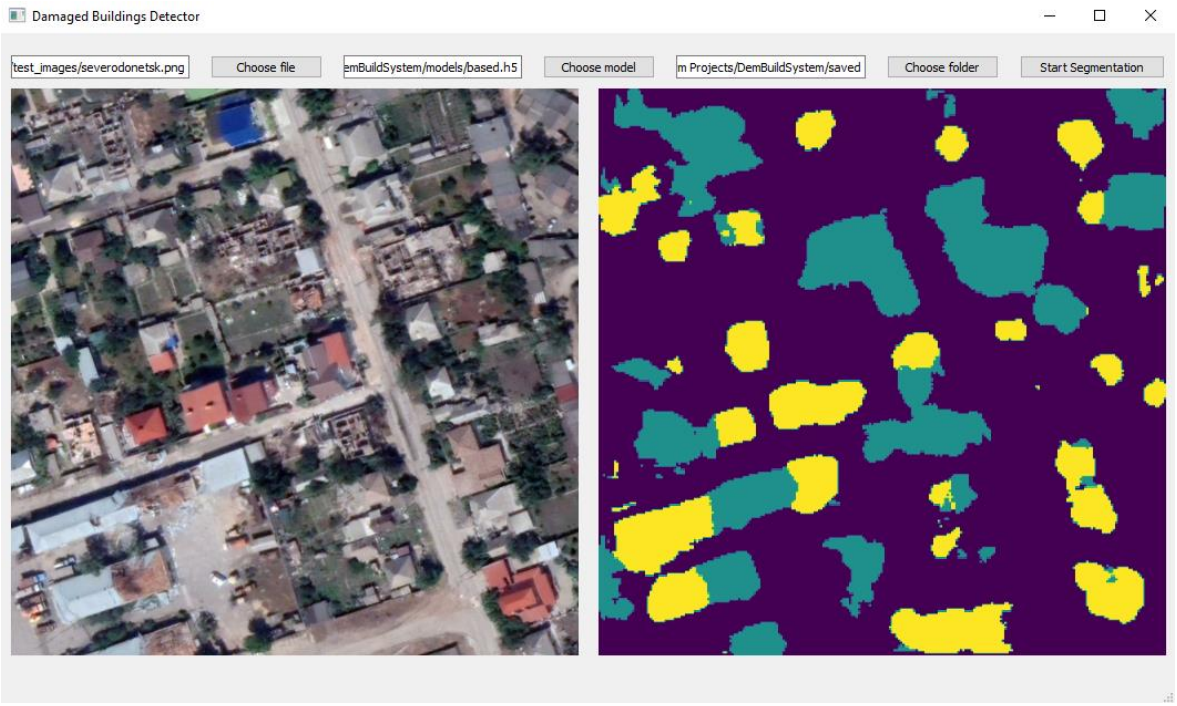

Рисунок 3.36 – Результати роботи програми на тестовому зображенні

Оскільки в реалізації застосовуються елементи, що контролюють обране розширення файлу користувача, а також обробку помилок у разі випадкового завершення чи закриття підменю, програма працює коректно протягом всього періоду використання.

3.5 Дослідження впливу параметрів на точність моделі

Щоб дослідити вплив параметрів на якість та точність моделі, було побудовано та проведено тестування для декількох моделей, що мають різні характеристики, відмінні від базової. Під час дослідження було звернено увагу на такі показники як втрати (англ. loss), точність моделі (англ. accuracy), та відповідно показники IoU окремо для кожного класу.

У таблиці 3.1 наведено результати тестування побудованих моделей, також вказано час навчання та основні характеристики, які були змінені протягом часу проведення дослідження.

Таблиця 3.1 – Результати дослідження точності моделей на тренувальному наборі

№	Модель	Loss	Accuracy	IoU Damaged	IoU Normal	Time (m)
1	32 Filters, 256x256, 25 epochs, 10x augmentations	0.4417	0.8228	0.4267	0.3837	6m 7s
2	32 Filters, 256x256, 50 epochs, 10x augmentations	0.5706	0.7961	0.3908	0.3387	11m 57s
3	32 Filters, 256x256, 25 epochs, 5x augmentations	0.5165	0.7888	0.3968	0.3267	2m 41s
4	64 Filters, 256x256, 25 epochs, 10x augmentations	**0.4023**	**0.8421**	**0.4583**	**0.4914**	13m 16s
5	64 Filters, 256x256, 40 epochs, 10x augmentations	0.4928	0.8404	0.4498	0.4787	19 37s

Згідно з результатів, декілька моделей виявилися кращими в ряді показників. Здебільшого на якість моделі вплинуло збільшення кількості фільтрів, що поглибило модель, але підвищило змогу оброблювати більше деталей та робити точніші прогнози. Найкраща модель №4 покращила прогноз для зруйнованих будівель майже на 3% та вцілілих будівель більше ніж на 10%, що можна вважати гарним результатом. Враховуючи складність моделей та кількість зображень в наборі даних, загальний результат дослідження є задовільним.

Висновки до розділу 3

Python та Google Colab – потужні та корисні інструменти для розробки як простих моделей штучних нейронних мереж, так і складних згорткових нейронних мереж. Наявність реалізованих класів для побудови різних шарів дозволяє швидко та просто створювати необхідну модель для поставленої задачі, а можливість використовувати потужності відеокарт у сервісі Colaboratory значно пришвидшує процес навчання та тестування.

В ході роботи було побудовано базову згорткову нейронну мережу U-Net, проведено тестування на розробленому наборі даних. Для реалізації інтелектуальної системи було використано фреймворк PyQt5, який має всі необхідні інструменти для побудови графічного інтерфейсу користувача. В інтелектуальній системі є можливість обрання зображення для сегментації, моделі нейронної мережі, яка буде використовуватися, та шлях зберігання опрацьованого зображення.

В кінці розділу описано дослідження параметрів моделі на її якість, для чого використовувалися такі параметри як рівень втрат, точність, а також метрики IoU та MeanIoU. Згідно результатів, вдалося побудувати більш ефективну модель, яка мала глибшу архітектуру та навчалася на більшій кількості епох, що позначилося на якості: вдалося підвищити рівень IoU для зруйнованих на 3%, та 10% для вцілілих будівель.

ВИСНОВКИ

В роботі було розглянуто процес розпізнавання пошкоджених будівель на знімках поверхні землі – це можуть бути як супутникові знімки, так і зображення, зроблені за допомогою безпілотних летальних апаратів. Визначення територій, які найбільше постраждали внаслідок різних катастроф, є важливою частиною для надання допомоги населенню, та пріоритетним завданням для подальшого відновлення постраждалих регіонів.

В ході виконання кваліфікаційної роботи було розглянуто сучасний стан проблеми, проаналізовано наявні дослідження та публікації в сфері розпізнавання об’єктів та патернів на зображеннях, зокрема супутникових знімках, за допомогою штучних нейронних мереж та інших алгоритмів; можливі інструменти для вирішення поставленої задачі. Як результат, було обрано у якості рішення процес семантичної сегментації за допомогою згорткових нейронних мереж, а саме архітектуру U-Net, що є оптимальною при використанні невеликого набору даних, та застосовується для вирішення задач сегментації зображень. Набір даних, що використовувався для навчання моделі, було створено самостійно за допомогою сервісу Google Earth. Набір містить 50 зображень розмірністю 512 на 512 пікселів, які містять зображення поверхні землі у місті Маріуполь.

Метою кваліфікаційної роботи магістра була автоматизація процесів розпізнавання будівель, які мають пошкодження, з використанням нейронних мереж. Виходячи з мети роботи, були виконані поставлені завдання, а саме:

1) проаналізовано наявні дослідження та публікації, які відображують можливі шляхи вирішення проблеми;

2) розглянуто наявні моделі нейронних мереж;

3) обрано оптимальну архітектуру нейронної мережі для розпізнавання будівель на знімках;

4) реалізовано обрану архітектуру за допомогою програмних засобів;

5) проведено навчання та тестування моделі;

6) розроблено інтелектуальну систему, що використовує навчену модель.

Результатом роботи є побудована та навчена згорткова мережа U-Net, а також інтелектуальна система, реалізована на мові Python з використанням фреймворку PyQt, яка використовує розроблену мережу для розпізнавання пошкоджених будинків на зображеннях. Користувач може обирати зображення для сегментації, модель та шлях для зберігання результату, що в свою чергу також є зображенням. Реалізована інтелектуальна система має простий та функціональний інтерфейс, чітко працює згідно визначеного технічного завдання.

Для використання в застосунку була побудована базова модель мережі, та досліджено вплив параметрів, таких як, наприклад, розмір та кількість фільтрів, кількість аугментованих зображень, епох навчання тощо, на якість та точність моделі. Як результат, було вдосконалено поточну мережу: вдалося підвищити рівень IoU для зруйнованих на 3%, та 10% для вцілілих будівель, шляхом збільшення кількості аугментацій та епох навчання. Таким чином, покращена модель має загальну точність близько 84.21%, та безпосередньо 49.14% для вцілілих будівель і 45.83% для зруйнованих.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The total amount of damage caused to Ukraine's infrastructure due to the war has increased to almost $138 billion. *KSE* : вебсайт. URL: https://kse.ua/about-the-school/news/the-total-amount-of-damage-caused-to-ukraine-s-infrastructure-due-to-the-war-has-increased-to-almost-138-billion/ (дата звернення: 05.10.2023).

2. Over $54 billion in damage to Ukraine's housing stock as a result of a full-scale war as of the end of May 2023. *KSE* : вебсайт. URL: https://kse.ua/about-the-school/news/over-54-billion-in-damage-to-ukraine-s-housing-stock-as-a-result-of-a-full-scale-war-as-of-the-end-of-may-2023/ (дата звернення: 06.10.2023).

3. Damage, loss and needs assessment - tools and methodology. *GFDRR* : вебсайт. URL: https://www.gfdrr.org/en/damage-loss-and-needs-assessment-tools-and-methodology (дата звернення: 10.11.2023).

4. OPEN DATA PROGRAM. *Maxar* : вебсайт. URL: https://www.maxar.com/open-data (дата звернення: 10.11.2023).

5. Google Maps, Google Earth, and Street View. *Google* : вебсайт. URL: https://about.google/brand-resource-center/products-and-services/geo-guidelines/ (дата звернення: 10.11.2023).

6. V Rashidian V, Baise LG, Koch M, Moaveni B. Detecting Demolished Buildings after a Natural Hazard Using High Resolution RGB Satellite Imagery and Modified U-Net Convolutional Neural Networks. *Remote Sensing*. 2021. Vol. 13(11):2176. DOI: 10.3390/rs13112176.

7. Takhtkeshha, Narges, Ali Mohammadzadeh, and Bahram Salehi. A Rapid Self-Supervised Deep-Learning-Based Method for Post-Earthquake Damage Detection Using UAV Data (Case Study: Sarpol-e Zahab, Iran). *Remote Sensing*. 2023. Vol. 15, 1: 123. DOI: 10.3390/rs15010123.

8. Li, Yundong, Wei Hu, Han Dong, and Xueyan Zhang. Building Damage Detection from Post-Event Aerial Imagery Using Single Shot Multibox Detector. *Applied Sciences.* 2019. Vol. 9, 6: 1128. DOI: 10.3390/app9061128.

9. Ghandour, Ali J., and Abedelkarim A. Jezzini. Post-War Building Damage Detection. *Proceedings*. 2018. Vol. 2, 7: 359. DOI: 10.3390/ecrs-2-05172.

10. Aimaiti, Yusupujiang, Christina Sanon, Magaly Koch, Laurie G. Baise, and Babak Moaveni. War Related Building Damage Assessment in Kyiv, Ukraine, Using Sentinel-1 Radar and Sentinel-2 Optical Images. *Remote Sensing*. 2022. Vol. 14, 24: 6239. DOI: 10.3390/rs14246239.

11. Nils J. Nilsson. Introduction to machine learning. Stanford University, 1998. 188 p.

12. Tommaso Teofili. Deep Learning for Search 1st Edition. Manning, 2019. 328 p.

13. Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large-scale image recognition : стаття. Oxford : University of Oxford, 2015. 14 с. URL: https://arxiv.org/pdf/1409.1556.pdf (дата звернення: 20.11.2023).

14. Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation : стаття. Freiburg : University of Freiburg, 2004. 9 с. URL: https://arxiv.org/pdf/1505.04597.pdf (дата звернення: 20.11.2023).

15. Advantages and disadvantages of UNet-related networks. ResearchGate : вебсайт. URL: https://www.researchgate.net/figure/Advantages-and-disadvantages-of-UNet-related-networks_tbl1_371527497 (дата звернення: 20.11.2023).

16. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Web Spam Taxonomy : стаття. California : Stanford University, 2004. 9 с. URL: https://arxiv.org/pdf/1311.2524.pdf (дата звернення: 25.11.2023).

17. Understanding R-CNN: A Revolution in Object Detection. *Medium* : вебсайт. URL: https://medium.com/@evertongomede/understanding-r-cnn-a-revolution-in-object-detection-a02c2754218c (дата звернення: 25.11.2023).

18. Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *CVPR2015*. DOI: 10.48550/arXiv.1411.4038

19. Gradient-Based Learning Applied to Document Recognition / Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner. *Proc. Of the IEEE*. 1998. URL: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf (дата звернення: 05.12.2023).

20. Kunihiko Fukushima, Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*. 1982. Vol 15, № 6, pp 455-469. DOI: 10.1016/0031-3203(82)90024-3.

21. Alex Krizhevskym, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Depp Convolutional Neural Networks: стаття. Toronto: University of Toronto, 2012. 9 с. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (дата звернення: 05.12.2023).

22. Vincent Dumoulin, Francesco Visin. A guide to convolution arithmetic for deep learning. 2018. DOI: arXiv:1603.07285v2.

23. Feldmann, S., Schmiedt, M., Schlosser, J.M. et al. Recursive quality optimization of a smart forming tool under the use of perception based hybrid datasets for training of a Deep Neural Network. *Discov Artif Intell*. 2022. Vol. 2, № 17.

24. Eros Innocenti, Alessandro Vizzarri. Machine Learning Methods for Computer Vision: стаття. Italy: Guglielmo Marconi University, 2021. 5 с. URL: https://ceur-ws.org/Vol-3118/p12.pdf (дата звернення: 12.12.2023).

25. Richard Szeliski. Computer Vision: Algorithms and Applications: електронна книга. Springer, 2010. 979 с. URL: https://www.cs.ccu.edu.tw/~damon/tmp/SzeliskiBook_20100903_draft.pdf (дата звернення: 14.12.2023).

26. Khaled Alomar, Halil Ibrahim Aysel, Xiaohao Cai. Data Augmentation in Classification and Segmentation: A Survey and New Strategies. *Imaging*. 2023. Vol. 9, № 2, p. 46. DOI: 10.3390/jimaging9020046.

27. Sandhi Wangiyana, Piotr Samczynski, Artur Gromek. Data Augmentation for *Building Footprint Segmentation in SAR Images: An Empirical Study*. Remote Sens. 2022. Vol. 14, №9. DOI: 10.3390/rs14092012.

28. Sengdeok Bang, Francis Baek, Somin Park, Wontae Kim, Hyoungkwan Kim. Image augmentation to improve construction resource detection using generative adversarial networks, cut-and-paste, and image transformation techniques. *Automation in Construction*. 2020. Vol. 115. DOI: 10.1016/j.autcon.2020.103198.

29. Golnaz Ghiasi, Yin Cui, Aravind Srinivas. Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation. *CVPR 2021*. 2021. DOI: 10.48550/arXiv.2012.07177.

30. Python Documentation. *Python*: вебсайт. URL: https://www.python.org/doc/ (дата звернення: 10.01.2024).

31. TensorFlow Api Versions. *Tensorflow*: вебсайт. URL: https://www.tensorflow.org/versions (дата звернення: 10.01.2024).

32. API Reference. *ScikitLearn*: вебсайт. URL: https://scikit-learn.org/stable/modules/classes.html (дата звернення: 15.01.2024).

33. Albumentations documentation. *Albumentations*: вебсайт. URL: https://albumentations.ai/docs/ (дата звернення: 15.01.2024).

34. What is PyQt? *Riverbank Computing*: вебсайт. URL: https://www.riverbankcomputing.com/software/pyqt/ (дата звернення: 17.01.2024).

35. Google Earth Online. *Google Earth*: вебсайт. URL: https://earth.google.com/ (дата звернення: 18.01.2024).

36. LabelMe. *Github*: вебсайт. URL: https://github.com/labelmeai/labelme (дата звернення: 26.01.2024).

37. Colaboratory. *Google*: вебсайт. URL: https://research.google.com/colaboratory/faq.html (дата звернення: 26.01.2024).

ДОДАТОК А
Приклад коду

main.py

```
import sys
from PyQt5.QtWidgets import QApplication
from gui.menu import UI

if __name__ == '__main__':
    app = QApplication(sys.argv)
    UIWindow = UI()
    app.exec_()
```

menu.py

```
import os
from datetime import datetime
import keras
import numpy as np
import tensorflow as tf
from PIL import Image
from PyQt5 import uic
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QMainWindow, QPushButton, QLabel, \
    QLineEdit, QFileDialog
from matplotlib import pyplot as plt
from metrics.fixed_metrics import UpdatedIoU, UpdatedMeanIoU

class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()
        uic.loadUi("gui/menu.ui", self)

        self.model = None
        self.size = (512, 512)

        title = "Damaged Buildings Detector"
        self.setWindowTitle(title)

        self.path_to_file_edit = self.findChild(QLineEdit, "path_to_file_edit")
        self.choose_file_btn = self.findChild(QPushButton, "choose_file_btn")
        self.choose_file_btn.clicked.connect(self.choose_file_event)

        self.path_to_folder_edit = self.findChild(QLineEdit, "path_to_folder_edit")
        self.choose_folder_btn = self.findChild(QPushButton, "choose_folder_btn")
        self.choose_folder_btn.clicked.connect(self.choose_folder_event)

        self.path_to_model_edit = self.findChild(QLineEdit, "path_to_model_edit")
        self.choose_model_btn = self.findChild(QPushButton, "choose_model_btn")
        self.choose_model_btn.clicked.connect(self.choose_model_event)

        self.original_image_label = self.findChild(QLabel, "original_image_label")
```

```
        self.seg_image_label = self.findChild(QLabel, "seg_image_label")

        self.start_seg_btn = self.findChild(QPushButton, "start_seg_btn")
        self.start_seg_btn.clicked.connect(self.start_segmentation_event)

        self.show()

    def choose_file_event(self):
        path = os.path.abspath('test_images/')
        file_name, text = QFileDialog.getOpenFileName(self, 'Open file',
                                            path, "Image files (*.jpg *.png)")
        self.path_to_file_edit.setText(file_name)

        image = QPixmap(file_name)
        image = image.scaled(self.original_image_label.width(),
self.original_image_label.height())

        self.original_image_label.setPixmap(image)

    def choose_folder_event(self):
        folder_name = QFileDialog.getExistingDirectory(self, 'Select Folder')
        self.path_to_folder_edit.setText(folder_name)

    def choose_model_event(self):
        path = os.path.abspath('models/')
        model_name, text = QFileDialog.getOpenFileName(self, 'Open file',
                                            path, "Model files (*.h5)")
        self.path_to_model_edit.setText(model_name)

        self.model = keras.models.load_model(self.path_to_model_edit.text(),
                                             custom_objects={'UpdatedMeanIoU':
UpdatedMeanIoU,
                                                             'UpdatedIoU': UpdatedIoU})
        config = self.model.get_config()
        batch, height, weight, channel =
config["layers"][0]["config"]["batch_input_shape"]
        self.size = height, weight

    def start_segmentation_event(self):
        img = Image.open(self.path_to_file_edit.text()).convert('RGB')

        img = img.resize((self.size[0], self.size[1]))
        img = np.reshape(img, (self.size[0], self.size[1], 3))
        img = img / 256.

        test = img[np.newaxis, ...]
        pred_y = self.model.predict(test)
        pred_mask = tf.argmax(pred_y[0], axis=-1)
        pred_mask = pred_mask[..., tf.newaxis]

        fig = plt.figure(frameon=False)
        ax = plt.Axes(fig, [0., 0., 1., 1.])
```

```
        ax.set_axis_off()
        fig.add_axes(ax)
        plt.imshow(pred_mask, aspect='auto')

        img_name = 'result {:%Y-%m-%d %H-%M-%S}.png'.format(datetime.now())
        segmented_path = self.path_to_folder_edit.text() + '/' + img_name

        fig.savefig(segmented_path)

        image = QPixmap(segmented_path)
        image = image.scaled(self.original_image_label.width(),
self.original_image_label.height())

        self.seg_image_label.setPixmap(image)
```

unet.py

```
import os

# images_v2_new
path = '/content/drive/MyDrive/Datasets/images_v2_new'
path1 = f'{path}/enhanced/'
path2 = f'{path}/segmented/'

import os
import imageio.v2 as imageio
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import albumentations as A
from albumentations import Compose, Rotate, HorizontalFlip, VerticalFlip, RandomRotate90,
Transpose, GridDistortion
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split
import keras.backend as K

def load_data(path1, path2):
    image_dataset = os.listdir(path1)
    mask_dataset = os.listdir(path2)

    orig_img = []
    mask_img = []
    for file in image_dataset:
        orig_img.append(file)
    for file in mask_dataset:
        mask_img.append(file)

    orig_img.sort()
    mask_img.sort()

    return orig_img, mask_img
```

```
transforms_v2 = Compose([
            A.OneOf([
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.3, scale_limit=0.1, rotate_limit=270,
p=0.7),
                    # A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1, hue=0.1,
p=0.5),
                ]),
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.05,
                                       rotate_limit=90, p=0.5),
                    A.MotionBlur(blur_limit=3, p=0.3),
                    # A.GaussNoise(var_limit=(1, 2), p=0.4)
                ]),
                A.Sequential([
                    A.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.15,
                                       rotate_limit=180, p=0.8),

                ]),
            ])
        ])

def do_augmentation(img, mask, target_shape_img, target_shape_mask, iterations):
    length = len(img)
    iterations += 1
    m = length * iterations
    i_h, i_w, i_c = target_shape_img
    m_h, m_w, m_c = target_shape_mask

    X = np.zeros((m, i_h, i_w, i_c), dtype=np.float32)
    y = np.zeros((m, m_h, m_w, m_c), dtype=np.int32)

    counter = 0
    for i in range(0, iterations):
        for j in range(0, length):
            if i == 0:
                X[counter] = img[j]
                y[counter] = mask[j]
            else:
                augmented = transforms_v2(image=img[j], mask=mask[j])
                X[counter] = augmented["image"]
                y[counter] = augmented["mask"]
            counter += 1

    seed = np.random.randint(0, 10000)

    np.random.seed(seed)
    np.random.shuffle(X)
    np.random.seed(seed)
    np.random.shuffle(y)

    return X, y
```

```
def preprocess_data(img, mask, target_shape_img, target_shape_mask, path1, path2):
    m = len(img)
    i_h, i_w, i_c = target_shape_img
    m_h, m_w, m_c = target_shape_mask

    X = np.zeros((m, i_h, i_w, i_c), dtype=np.float32)
    y = np.zeros((m, m_h, m_w, m_c), dtype=np.int32)

    for file in img:
        index = img.index(file)
        path = os.path.join(path1, file)
        single_img = Image.open(path).convert('RGB')
        single_img = single_img.resize((i_h, i_w))
        single_img = np.reshape(single_img, (i_h, i_w, i_c))

        single_mask_ind = mask[index]
        path = os.path.join(path2, single_mask_ind)
        single_mask = Image.open(path)
        single_mask = single_mask.resize((m_h, m_w))
        single_mask = np.reshape(single_mask, (m_h, m_w, m_c))

        single_img = single_img / 256.
        X[index] = single_img

        single_mask = single_mask + 0
        y[index] = single_mask

    return X, y

def conv2d_block(input_tensor, n_filters, kernel_size=3):
    x = input_tensor
    for i in range(2):
        x = tf.keras.layers.Conv2D(filters=n_filters, kernel_size=(kernel_size,
kernel_size), \
                                   kernel_initializer='he_normal', padding='same')(x)
        x = tf.keras.layers.Activation('relu')(x)

    return x

def encoder_block(inputs, n_filters=64, pool_size=(2, 2), dropout=0.3):
    f = conv2d_block(inputs, n_filters=n_filters)
    p = tf.keras.layers.MaxPooling2D(pool_size=pool_size)(f)
    p = tf.keras.layers.Dropout(dropout)(p)

    return f, p
def encoder(inputs, n_filters, dropout):
    f1, p1 = encoder_block(inputs, n_filters=n_filters, pool_size=(2, 2), dropout=dropout)
    f2, p2 = encoder_block(p1, n_filters=n_filters * 2, pool_size=(2, 2), dropout=dropout)
    f3, p3 = encoder_block(p2, n_filters=n_filters * 4, pool_size=(2, 2), dropout=dropout)
    f4, p4 = encoder_block(p3, n_filters=n_filters * 8, pool_size=(2, 2), dropout=dropout)
```

```
    return p4, (f1, f2, f3, f4)

def bottleneck(inputs, n_filters):
    bottle_neck = conv2d_block(inputs, n_filters=n_filters * 16)

    return bottle_neck

def decoder_block(inputs, conv_output, n_filters, kernel_size, strides, dropout):
    u = tf.keras.layers.Conv2DTranspose(n_filters, kernel_size, strides=strides,
padding='same')(inputs)
    c = tf.keras.layers.concatenate([u, conv_output])
    c = tf.keras.layers.Dropout(dropout)(c)
    c = conv2d_block(c, n_filters, kernel_size=3)

    return c

def decoder(inputs, n_filters, convs, output_channels, dropout):
    f1, f2, f3, f4 = convs

    c6 = decoder_block(inputs, f4, n_filters=n_filters * 8, kernel_size=(3, 3),
strides=(2, 2), dropout=dropout)
    c7 = decoder_block(c6, f3, n_filters=n_filters * 4, kernel_size=(3, 3), strides=(2,
2), dropout=dropout)
    c8 = decoder_block(c7, f2, n_filters=n_filters * 2, kernel_size=(3, 3), strides=(2,
2), dropout=dropout)
    c9 = decoder_block(c8, f1, n_filters=n_filters, kernel_size=(3, 3), strides=(2, 2),
dropout=dropout)

    outputs = tf.keras.layers.Conv2D(output_channels, (1, 1), activation='softmax' )(c9)
    return outputs

img, mask = load_data(path1, path2)

# View an example of image and corresponding mask
show_images = 3
for i in range(show_images):
    img_view  = imageio.imread(path1 + img[i])
    mask_view = imageio.imread(path2 + mask[i])
    print(img_view.shape)
    print(mask_view.shape)
    fig, arr = plt.subplots(1, 2, figsize=(7, 7))
    arr[0].imshow(img_view)
    arr[0].set_title('Image '+ str(i))
    arr[1].imshow(mask_view)
    arr[1].set_title('Masked Image '+ str(i))
    plt.show()

# Define the desired shape
SIZE = 256
target_shape_img = [SIZE, SIZE, 3]
target_shape_mask = [SIZE, SIZE, 1]
```

```
X, y = preprocess_data(img, mask, target_shape_img, target_shape_mask, path1, path2)

print("X Shape:", X.shape)
print("Y shape:", y.shape)
print(np.unique(y))

# Visualize the output
image_index = 0
fig, arr = plt.subplots(1, 2, figsize=(7, 7))
arr[0].imshow(X[image_index])
arr[0].set_title('Processed Image')
arr[1].imshow(y[image_index,:,:,0])
arr[1].set_title('Processed Masked Image ')

plt.show()

OUTPUT_CHANNELS = 3

def unet(n_filters, shape, dropout):
    inputs = tf.keras.layers.Input(shape=shape)

    encoder_output, convs = encoder(inputs, n_filters, dropout)

    bottle_neck = bottleneck(encoder_output, n_filters)

    outputs = decoder(bottle_neck, n_filters, convs, output_channels=OUTPUT_CHANNELS,
dropout=dropout)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model

class UpdatedMeanIoU(tf.keras.metrics.MeanIoU):
  def __init__(self, y_true=None, y_pred=None, num_classes=None, name=None, dtype=None):
    super(UpdatedMeanIoU, self).__init__(num_classes=num_classes, name=name, dtype=dtype)

  def update_state(self, y_true, y_pred, sample_weight=None):
    y_pred = tf.math.argmax(y_pred, axis=-1)
    return super().update_state(y_true, y_pred, sample_weight)

class UpdatedIoU(tf.keras.metrics.IoU):
  def __init__(self, y_true=None, y_pred=None, num_classes=None, target_class_ids=None,
               name=None, dtype=None):
    super(UpdatedIoU, self).__init__(num_classes=num_classes,
target_class_ids=target_class_ids, name=name, dtype=dtype)

  def update_state(self, y_true, y_pred, sample_weight=None):
    y_pred = tf.math.argmax(y_pred, axis=-1)
    return super().update_state(y_true, y_pred, sample_weight)

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3, random_state=1)
print(f"Train size {len(X_train)} Test size {len(X_valid)}")
```

```
X_train, y_train = do_augmentation(X_train, y_train, target_shape_img,
                                   target_shape_mask, 10)
X_valid, y_valid = do_augmentation(X_valid, y_valid, target_shape_img,
                                   target_shape_mask, 10)

N_FILTERS = 64
INPUT_SHAPE = (SIZE, SIZE, 3)

unet = unet(n_filters=N_FILTERS, shape=INPUT_SHAPE, dropout=0.3)
unet.summary()

unet.compile(optimizer=tf.keras.optimizers.Adam(),
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy',
                       UpdatedMeanIoU(num_classes=3),
                       UpdatedIoU(num_classes=3, target_class_ids=[1]),
                       UpdatedIoU(num_classes=3, target_class_ids=[2])])

weights_v1 = {0: 1, 1: 3, 2: 2}

results = unet.fit(X_train, y_train, class_weight=weights_v1,
                   batch_size=15, epochs=40, validation_data=(X_valid, y_valid))

fig, axis = plt.subplots(1, 5, figsize=(16, 4))
axis[0].plot(results.history["loss"], color='r', label = 'train loss')
axis[0].plot(results.history["val_loss"], color='b', label = 'dev loss')
axis[0].set_title('Loss Comparison')
axis[0].legend()
axis[1].plot(results.history["accuracy"], color='r', label = 'train accuracy')
axis[1].plot(results.history["val_accuracy"], color='b', label = 'dev accuracy')
axis[1].set_title('Accuracy Comparison')
axis[1].legend()
axis[2].plot(results.history["updated_mean_io_u_4"], color='r', label = 'train
updated_mean_io_u')
axis[2].plot(results.history["val_updated_mean_io_u_4"], color='b', label = 'dev
updated_mean_io_u')
axis[2].set_title('Mean IoU Comparison')
axis[2].legend()
axis[3].plot(results.history["updated_io_u_8"], color='r', label = 'train updated_io_u')
axis[3].plot(results.history["val_updated_io_u_8"], color='b', label = 'dev updated_io_u')
axis[3].set_title('IoU Comparison for class 1')
axis[3].legend()
axis[4].plot(results.history["updated_io_u_9"], color='r', label = 'train updated_io_u')
axis[4].plot(results.history["val_updated_io_u_9"], color='b', label = 'dev updated_io_u')
axis[4].set_title('IoU Comparison for class 2')
axis[4].legend()
plt.show()

unet.evaluate(X_valid, y_valid)
```