

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повна назва кафедри, циклової комісії)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ РОЗПІЗНАВАННЯ ТРІЩИН
БУДІВЕЛЬ

SOFTWARE IMPLEMENTATION OF THE METHODS
FOR CRACK DETECTION IN BUILDINGS

Виконав: студент(ка) 4 курсу, групи КНТ-137

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Судаков В. Р.

(прізвище та ініціали)

Керівник Терещенко Е. В.

(прізвище та ініціали)

Рецензент Казурова А. Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Інститут, факультет ІРЕ, ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н., проф.

С. О. Субботін

“ ” 20__ року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Судакова Володимира Романовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи): Програмна реалізація методів розпізнавання тріщин будівель. Software Implementation of the Methods for Crack Detection in Buildings

керівник проєкту (роботи) Терещенко Еліна Валентинівна, к.ф.-м.н, доцент,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17 травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Аналіз програмних засобів.

3. Розробка і реалізація компонентів програми. 4. Експлуатація, тестування та експериментальне дослідження програми. 5. Керівництво програміста.

6. Керівництво оператора.

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

Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
1-6 Основна частина	Терещенко Е. В., доцент		
Нормконтроль	Липовець М. В., асистент		

7. Дата видачі завдання “1” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Розділ 5, розділ 6, додатки
7	Захист роботи.	9 тиждень	

Студент(ка)

(підпис)

Судаков В. Р.

(прізвище та ініціали)

Керівник проекту (роботи)

(підпис)

Терещенко Е. В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра: 92 с., 2 табл., 39 рис., 2 дод., 18 джерел.

МАШИННЕ НАВЧАННЯ, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВИЯВЛЕННЯ ТРІЩИН, PYTHON, TENSORFLOW, KERAS, PYSIDE, MATPLOTLIB, U-NET, OPENCV.

Об'єкт дослідження – процес виявлення тріщин на будівлях.

Предмет дослідження – реалізація існуючих методів виявлення тріщин на зображеннях будівель.

Мета роботи – створення програмного застосунку з інтуїтивно зрозумілою навігацією для автоматизації виявлення тріщин на зображеннях будівель методами глибинного навчання.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Python, середовище розробки PyCharm, текстовий формат обміну даних JSON, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10 Pro.

Результати. Створено програмний застосунок, керований кросплатформовим інтерпретатором Python, який за допомогою комп'ютерної нейронної мережі сканує зображення будівель та інших поверхонь на наявність тріщин.

Висновки. Реалізовано метод нейромережевого аналізу зображень на наявність тріщин та розроблено програму з графічним інтерфейсом для його використання.

Галузь використання – дослідження міського середовища, аналіз необхідності ремонтних робіт нерухомості, знаходження та попередження будівель аварійного стану.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 92 pages, 39 figures, 2 tables, 2 appendixes, 18 sources.

MACHINE LEARNING, ARTIFICIAL NEURAL NETWORKS, CONVOLUTIONAL NEURAL NETWORKS, IMAGE RECOGNITION, SOFTWARE, DETECTION OF CRACKS, PYTHON, TENSORFLOW, KERAS, PYSIDE, MATPLOTLIB, U-NET, OPENCV.

Object of study is a process of detecting cracks in buildings.

Subject of study is the implementation of existing methods for detecting cracks on the images of buildings.

The purpose of the work is to create a software system with intuitive navigation to automate the detection of cracks on the images of buildings by deep learning methods.

Materials, methods, and tools: structured and object-oriented programming, Python programming language, PyCharm IDE, JSON text exchange format, personal computer with Intel Core i5 processor running Microsoft Windows 10 Pro operating system.

Results. A software application controlled by the cross-platform Python interpreter, which uses a computer neural network to scan images of buildings and other surfaces for cracks.

Conclusions. The method of neural network analysis of images for the presence of cracks on it is implemented and a program with a graphical interface for its use is developed.

The field of use is research of the urban environment, analysis of the need for repair of real estate, finding and prevention of emergency buildings.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз предметної області.....	11
1.1 Актуальність та необхідність виявлення тріщин будівель.....	11
1.2 Огляд штучних нейронних мереж.....	12
1.3 Дослідження згорткових нейронних мереж.....	14
1.3.1 Архітектура згорткової мережі U-Net.....	15
1.4 Дослідження нейромережевих засобів аналізу зображень.....	16
1.5 Огляд інших методів, що вирішують аналогічні завдання.....	17
1.6 Постановка задачі розпізнавання тріщин на зображеннях.....	18
1.7 Висновки за розділом 1	19
2 Аналіз програмних засобів.....	20
2.1 Вибір інструментарію розробки додатку	20
2.1.1 Операційна система та мова програмування	20
2.1.2 Середовище розробки.....	22
2.1.3 Мова програмування Python	23
2.2 Огляд застосованих технологій	25
2.2.1 Огляд можливостей Qt	26
2.2.2 Реалізація згорткової НМ U-Net.....	26
2.3 Висновки за розділом 2	27
3 Розробка і реалізація компонентів програми	29

	7
3.1 Опис основних рішень щодо реалізації системи	29
3.2 Структура програми.....	29
3.3 Розробка класів та пакетів програми	31
3.4 Розробка програмних модулів	34
3.5 Розробка інтерфейсу користувача, графічні елементи	35
3.6 Опис процесу тренування (навчання) моделі	37
3.7 Опис процесу розпізнавання.....	38
3.8 Висновки за розділом 3	39
4 Експлуатація, тестування та експериментальне дослідження програми	40
4.1 Функціональні можливості програми	40
4.2 Діаграма послідовностей.....	42
4.3 Тестування програми	43
4.4 Напрямки подальшого розвитку технології.....	45
4.5 Висновки за розділом 4	45
5 Керівництво програміста.....	46
5.1 Призначення й умови застосування програми.....	46
5.2 Основні вимоги до системи	46
5.3 Характеристики програми.....	47
5.4 Початкові та вихідні дані	49
6 Керівництво оператора	50
6.1 Призначення програми	50
6.2 Умови виконання програми	50
6.3 Виконання програми.....	51
Висновки	53

	8
Перелік джерел посилання	55
Додаток А Текст програми.....	57
Додаток Б Слайди презентації	84

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- IDE – Integrated Development Environment;
- ISBI – International Symposium on Biomedical Imaging;
- JSON – JavaScript Object Notation – «Запис об'єктів JavaScript»;
- LGPL – GNU Lesser General Public License – «Загальна громадська ліцензія обмеженого використання GNU»;
- НМ – штучні нейронні мережі;
- ОС – операційна система;
- ПЗ – програмне забезпечення;
- ПК – персональний комп'ютер.

ВСТУП

В усьому світі за рік будується нескінченна кількість будівель та за кожною спорудою потім необхідно вести спостереження, контролювати її стан та придатність до експлуатації. Будь-який будівничий матеріал має властивість деградувати та втрачати цілісність із плином часу, тож проблема моніторингу стану будівель завжди буде актуальною.

Обстеження будівель і споруд – це визначення технічного стану об'єктів будівництва з метою оцінки їх відповідності основним вимогам до будівель і споруд, що визначені відповідним технічним регламентом, та вжиття обґрунтованих заходів до забезпечення надійності та безпеки під час експлуатації об'єкта протягом усього періоду їх існування [1].

Одним із найбільш розповсюджених типів пошкоджень будівель є тріщини. Проблема забезпечення максимально можливого строку служби, продовження термінів експлуатації будівель в умовах реальних обмежень є однією з найактуальніших проблем для вчених та технічних спеціалістів у всьому світі. Наслідки виникнення відмов, несправностей або дефектів можуть призвести до техногенних катастроф із втратами людей, фатальними для оточуючого середовища та із значними матеріальними збитками.

Метою даної роботи є розробка методів та моделей в рамках інформаційної технології моніторингу, діагностики та розпізнавання дефектів типу «тріщина» будівельних об'єктів у процесі адаптивної фільтрації на основі цифрових зображень в оптичному діапазоні як ПЗ «VultureEye».

Використання розробки дозволить за малі кошти проводити аналіз місцевості на наявність пошкоджених конструкцій та активно спостерігати за процесом сканування та збору інформації. В майбутньому можна вдосконалити систему повністю функціональною апаратною частиною у вигляді енергоефективних дронів з камерами та вдосконаленою моделлю нейронної мережі для аналізу отриманих зображень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність та необхідність виявлення тріщин будівель

На сьогоднішній день серед будівель, які перебувають в експлуатації, достатньо велика частка має пошкоджені конструкції. Одним із найбільш розповсюджених типів пошкоджень будівель є тріщини (рис. 1.1). Проблема забезпечення максимально можливого строку служби, продовження термінів експлуатації будівель в умовах реальних обмежень є однією з найактуальніших проблем для вчених та технічних спеціалістів у всьому світі. Наслідки виникнення відмов, несправностей або дефектів можуть призвести до техногенних катастроф із втратами людей, фатальними для оточуючого середовища та із значними матеріальними збитками. А як відомо, тріщини є однією із найбільших загроз та причиною виникнення інших дефектів [2]. Тому дослідження виявлення тріщин є актуальною задачею.



Рисунок 1.1 – Тріщина будинка [18]

1.2 Огляд штучних нейронних мереж

Штучні нейронні мережі – це математичні моделі, а також їх програмні або апаратні реалізації, побудовані за принципами подання й обробки інформації у біологічних нейронних мережах – мережах нервових кліток живого організму [3].

Їх як правило використовують у вирішенні інтелектуальних задач, для яких неможливо сформулювати точний алгоритм вирішення та необхідний аналітичний підхід. Серед таких задач для НМ виділяють: розпізнавання образів, класифікацію, передбачення та кластеризація (рис. 1.2).

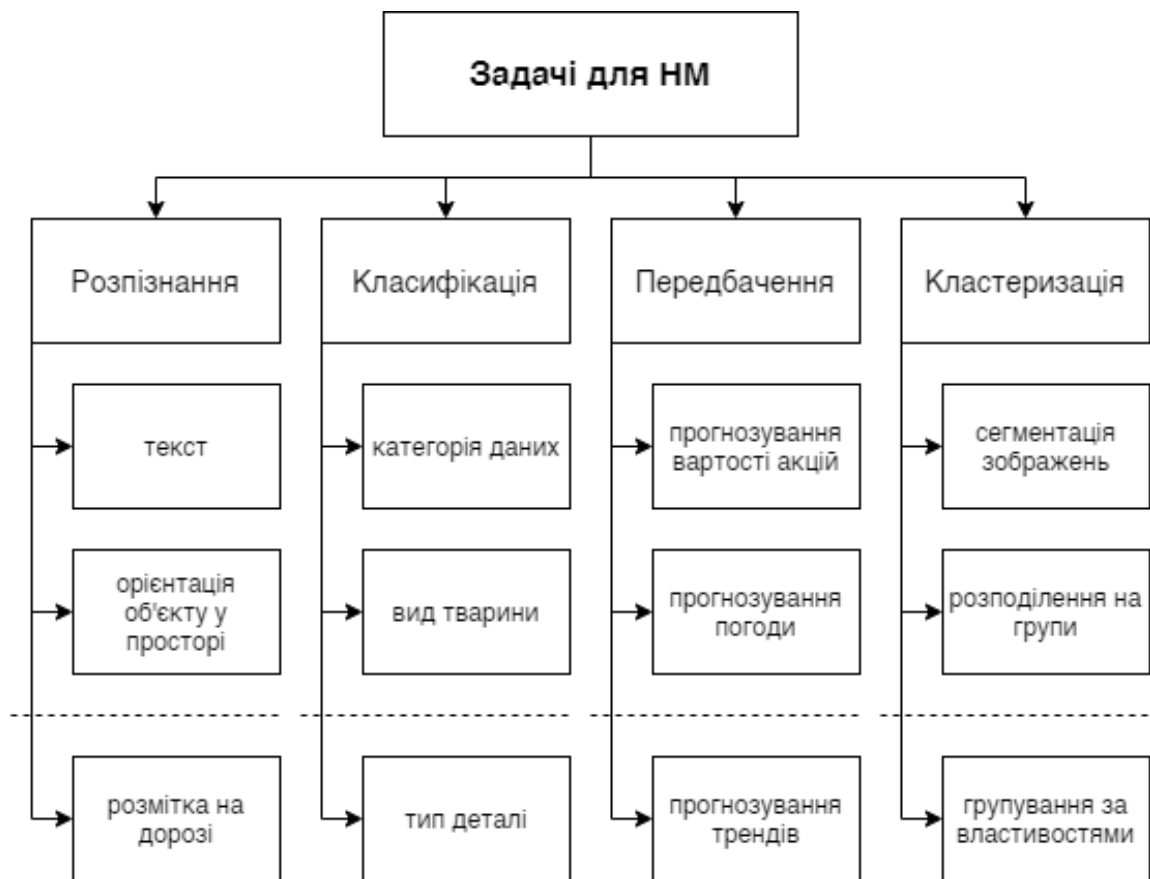


Рисунок 1.2 – Задачі для НМ

Ідея НМ запозичена із спрощеної моделі біологічного нейрона, тобто складається вона так само із нейронів, тільки штучних. Штучний нейрон є вузлом НМ, який може приймати сигнал з декількох входів, обробляє його та передає отриманий результат до єдиного виходу (рис. 1.3).

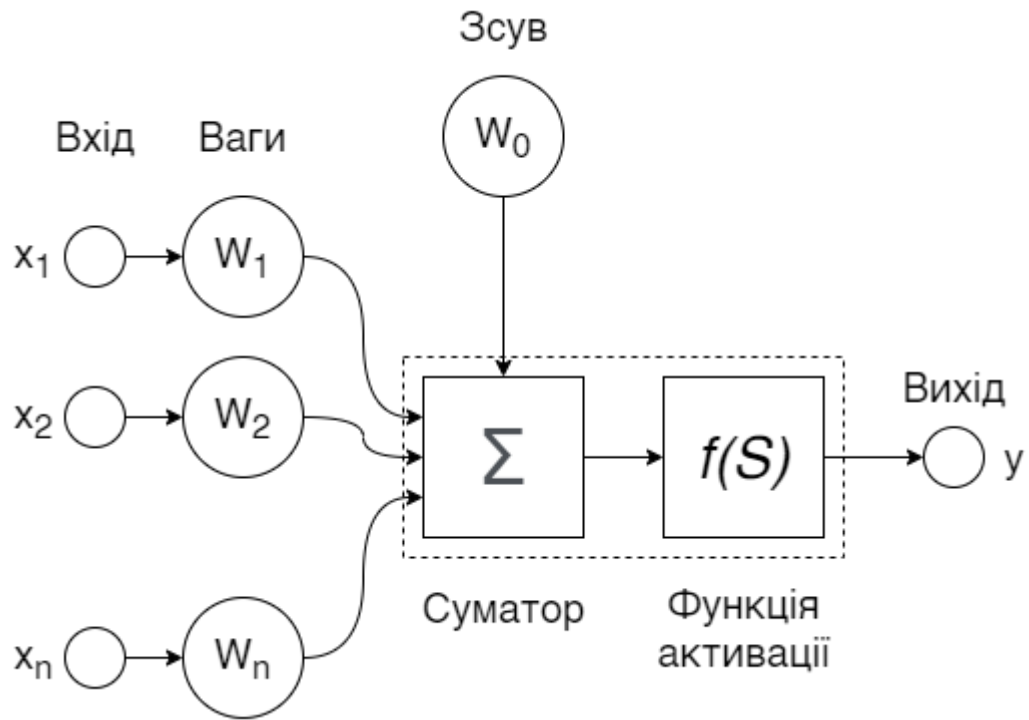


Рисунок 1.3 – Схема штучного нейрона

Математично нейрон являє собою ваговий суматор, який зазвичай представлений функцією зваженої суми таким чином (1.1):

$$S(w, x) = \sum_{i=1}^n w_i x_i + w_0 x_0 \quad (1.1)$$

На вхід до штучного нейрона подаються сигнали у представлені вектора $x = \{x_i\}, i=1, 2, \dots, n$, де n – кількість вхідних сигналів. Значення сигналів задають в інтервалі $[0, 1]$, вони можуть бути або дискретними, або аналоговими. Відповідно до кількості входів стільки ж мається і ваг ходів w_n , що представляють собою набір вагових коефіцієнтів $w = \{w_i\}$. Зважені вхідні сигнали та зсув w_0 передаються до суматора та потім до функції активації $f(S)$, яка підраховує скалярне значення для подання на вихід нейрона.

Так, з'єднуючи виходи одних нейронів зі входами інших, укупі штучні нейрони об'єднують у НМ. НМ в свою чергу змінює успішність вихідних значень через процес навчання, який зводиться до калібрування правильних значень у вагових коефіцієнтів.

1.3 Дослідження згорткових нейронних мереж

Згорткова нейронна мережа або convolutional neural network – це спеціальна архітектура НМ, яка була запропонована Яном Лекуном в одному з випусків научного журналу «Neural Computation» в 1989 році [4]. Головною особливістю згорткової нейронної мережі є наявність операції згортки [3]. В такій архітектурі кожен мережевий рівень виступає в якості фільтра виявлення на наявність певних ознак або шаблонів, присутніх у вхідних даних які представлено зображенням.

Згорткова НМ представляє собою багатошарову мережу прямого поширення без зворотних зв'язків. Загальна структура (рис. 1.4) може бути подана шарами:

- вхідний (input);
- згортковий (convolutional);
- агрегувальний (pooling);
- згладжувальний (flatten);
- повнозв'язний (fully connected);
- вихідний (output).



Рисунок 1.4 – Абстрактне представлення згорткової мережі

1.3.1 Архітектура згорткової мережі U-Net

U-net – це згорткова мережева архітектура (convolutional network architecture) для швидкої та точної сегментації зображень. До цього часу вона перевершила попередній найкращий метод (підхід вікна ковзання) на «ISBI challenge for segmentation of neuronal structures in electron microscopic stacks» [5]. Вона виграла у «Grand Challenge for Computer-Automated Detection of Caries in Bitewing Radiography at ISBI 2015» [6], а також «Cell Tracking Challenge at ISBI 2015» [7] за двома найскладнішими категоріями світлової мікроскопії «Phase contrast and DIC microscopy» з великим відривом [8].

Візуалізація архітектури НМ з офіційного сайту розробників представлена на рисунку 1.5 [9].

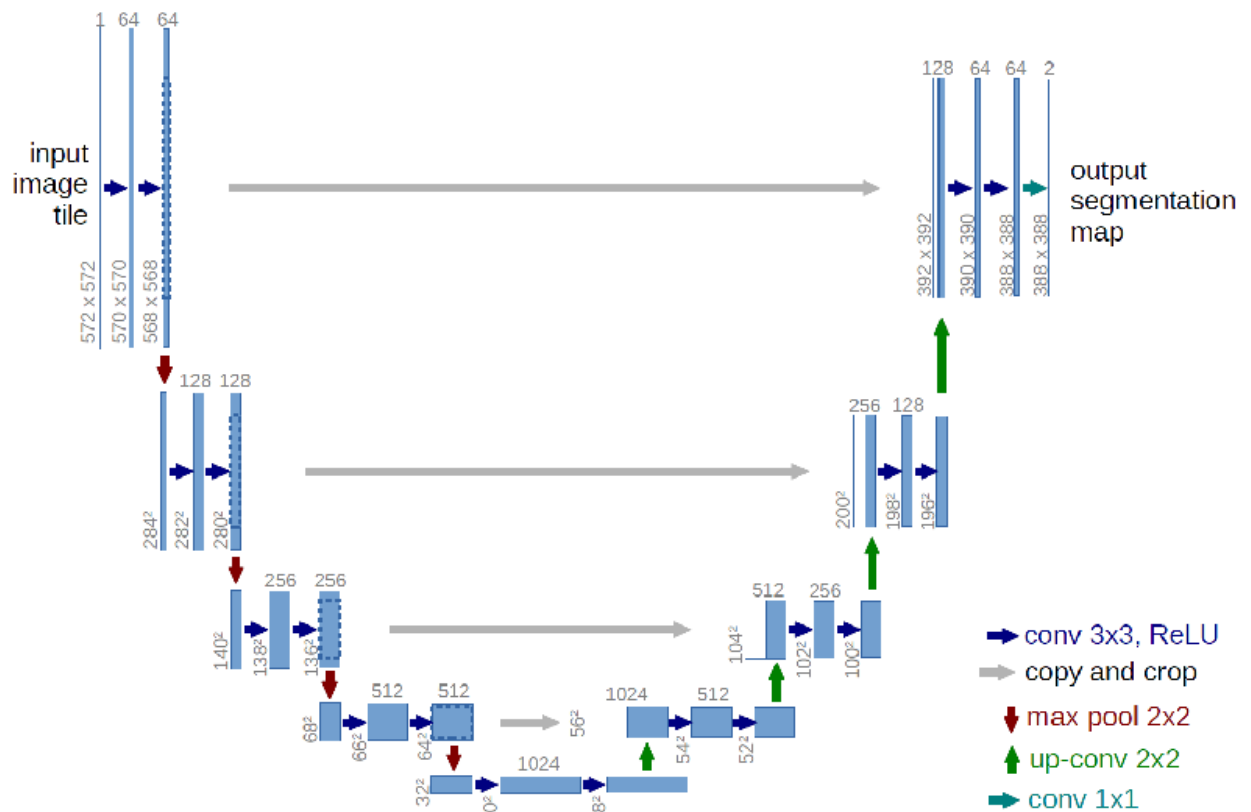


Рисунок 1.5 – Оригінальна архітектура U-Net [9]

1.4 Дослідження нейромережових засобів аналізу зображень

В останні роки штучні нейронні мережі здобули велику популярність у комп'ютерному зорі, який використовують для розпізнавання (класифікації) об'єктів на зображеннях та відео [10].

Існує багато варіацій штучних нейронних мереж, проте для задачі розпізнавання однією з найефективніших є згорткова мережа. Вона представляє собою один з видів багат шарових персептронів, який за своїми успішними результатами виявлення об'єктів на зображеннях вважається вдалим вибором для рішення задач з класифікації. По-перше такий персептрон не потребує багато попередніх обробок даних, по-друге результат роботи не залежить від орієнтації шуканого об'єкту у просторі на зображенні. Тобто при аналізі реальних зображень, які будуть мати погрішності та невдалий кут нахилу, точність роботи мережі не буде критично порушена.

Треба мати на увазі, що методи, засновані на використанні штучних нейронних мереж, мають як переваги, так і свої недоліки. Серед переваг можна виділити:

- виявлення декількох класів об'єктів одним підготовленим класифікатором;
- висока точність та надійність прогнозів виявлення та класифікації.

Серед недоліків:

- навіть досвідчений фахівець не одразу зможе обрати архітектуру мережі з оптимальними характеристиками (кількість нейронів, верств, зв'язків), необхідні попередні дослідження;
- висока обчислювальна складність, яка вводить свої обмеження через необхідність використання досить потужної техніки, яка придатна працювати з класифікаторами побудованими на основі штучних нейронних мереж;

- робота таких класифікаторів потребує додаткової оптимізації під конкретні системи, щоб забезпечити швидкодію у реальному часі (якщо є така потреба).

1.5 Огляд інших методів, що вирішують аналогічні завдання

На даний момент штучні нейронні мережі не є дуже популярним засобом у сфері технічного обстеження будівель. Для визначення стану та перспектив різних споруд використовуються більш традиційні методи наведені у таблиці 1.1.

Таблиця 1.1 – Методи виявлення тріщин

№	Метод	Опис
1	Візуальний	первинний огляд експертом
2	Ультразвуковий	виявлення невидимих недоліків матеріалів
3	Електромагнітний	застосовується для контролю електропровідних матеріалів у спорудах
4	Георадіолокаційний	частіше застосовують для аналізу фундаменту та ґрунту під ним
5	Радіометричний	визначення щільності матеріалів
6	Нейтронний	визначення вологості матеріалів
7	Відрив зі сколюванням	перевірка міцності матеріалів
8	Пневматичний	перевірка повітропроникності матеріалів
9	Тепловізійний	визначення рівня теплозахисту споруди, виявлення проблем з опаленням, перегрівом електропровідників
10	Нівелювання	виявлення об'ємної деформації, осідання

Усі перераховані методи є ресурсозатратними, довгими, потребують постійної залученості фахівців при роботі зі складним та коштовним обладнанням. Саме через це обстеження нерухомості зазвичай не проводять планово та регулярно, а тільки за дійсною необхідністю контролю. А саме обстеження обов'язково проводять тільки у випадках, коли необхідно [11]:

- а) зробити висновок про безпеку експлуатації самовільно будівлі;
- б) оцінити технічний стан будівлі при плануванні реконструкції;
- в) провести капітальний ремонт;

- г) провести перевірку після надзвичайних ситуацій;
- д) оформити договір купівлі-продажу;
- е) зіставити побудовану споруду з її проектом на відповідність.

Це ще раз підтверджує необхідність у створені інструменту, який дозволив би проводити регулярні заплановані обстеження та виявляти проблеми ще до виникнення більш серйозних ушкоджень. Такий інструмент не виключає використання вже існуючих методів та не замінить їх, натомість він збереже час фахівців та допоможе у прогнозуванні проведення детальних експертиз завчасно.

1.6 Постановка задачі розпізнавання тріщин на зображеннях

В даний час структурний стан конструкцій переважно перевіряється вручну, що є трудомістким, витратним та суб'єктивним процесом. З розвитком комп'ютерного зору з'являється можливість використовувати цифрові зображення для автоматизації процесу візуального аналізу. Метою цієї роботи є вивчення методів глибокого навчання для виявлення тріщин на зображеннях. Створюється набір даних із фотографіями переважно з цегляних та бетонних конструкцій, що містить складні фони та тріщини різних типів та розмірів. Розглядається мережа глибокого навчання U-net, що має забезпечити високу точність визначення дефектів на поверхнях.

Концепт передбачає завантаження пакета зображень об'єкта, далі проводиться ідентифікація і розпізнавання тріщин серед отриманих даних. Вивід отриманого аналізу міститься у зображеннях із масками тріщин де вони були розпізнані, щоб оператор міг підтвердити наявність дефекту та зробити висновки.

Для виявлення тріщин у роботі власноруч проводиться навчання моделі на підготовленому наборі даних (зображень та масок), щоб потім прогнозувати наявність тріщин на зображеннях та візуалізувати їх.

1.7 Висновки за розділом 1

Дослідження використання НМ у розпізнанні об'єктів на цифрових зображеннях дозволило сформулювати завдання дослідження. Необхідно зробити систему машинного зору, що дозволяє вирішити задачу розпізнавання об'єктів заданого класу: дефекту типу тріщина на будівлях.

На основі проведеного дослідження в даному розділі було вирішено реалізувати технологію розпізнавання об'єктів заданого класу використовуючи згорткові нейронні мережі.

Проведено аналіз архітектури та основних властивостей нейронних моделей архітектури U-Net. Отримані знання та практичні навички знадобляться для виконання роботи.

Готового програмного забезпечення, що в точності задовольняє потреби поставленої задачі не було виявлено, тому виникла необхідність розробити програму, яка дозволяє розпізнати об'єкт класу: дефект типу тріщина на будівлях, на основі існуючих НМ методів для розпізнавання образів на цифрових зображеннях.

Для досягнення мети роботи необхідно вирішити такі завдання:

- виконати аналіз вимог;
- обрати програмні засоби;
- виконати програмну реалізацію;
- провести тестування та експериментальне дослідження програми;
- розробити програмну документацію.

2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір інструментарію розробки додатку

2.1.1 Операційна система та мова програмування

Windows – сімейство комерційних операційних систем корпорації Microsoft, орієнтованих на управління за допомогою графічного інтерфейсу.

Microsoft – одна з найбільших транснаціональних компаній з виробництва пропрієтарного програмного забезпечення для різної обчислювальної техніки. Найпопулярнішими продуктами корпорації є операційні системи «Microsoft Windows» і офісний пакет «Microsoft Office».

Історія Windows бере свій початок в 1986 році, коли розроблено першу версію системи. Вона являла собою набір програм, що розширюють можливості існуючих операційних систем. В 1990 році вийшла нова версія - Windows 3.0, яка стала використовуватися на багатьох персональних комп'ютерах того часу. З того часу функціонал Windows безперервно зростає, як і кількість прихильників цієї ОС.

Сьогодні Windows встановлена на більшості персональних комп'ютерів і робочих станцій. Згідно з даними ресурсу Net Applications, станом на червень 2019 року, за управлінням операційних систем сімейства Windows працює близько на 87,89% ПК [12].

Windows 10 – операційна система для ПК і робочих станцій, розроблена корпорацією Microsoft. В Windows 10 з'явився довгий список важливих змін, серед найважливіших – суттєво покращено взаємодію з користувачем, значно вдосконалені заходи безпеки та багато іншого [13].

Система покликана стати єдиною для різних пристроїв, таких як персональні комп'ютери, планшети, смартфони та ін. Доступна єдина платформа розробки і єдиний магазин універсальних програм, сумісних з усіма підтримуваними пристроями.

Згідно зі статистичними даними сайту W3Schools, Windows 10 займає перше місце в світі серед операційних систем, що використовуються для доступу до мережі Інтернет, випередивши в квітні 2017 року попереднього лідера – Windows 7.

Мова Python – об'єктно-орієнтована мова програмування. Розроблена в 1990-1991 роках [14].

Python одна з найпопулярніших сучасних мов програмування. Python застосовують при роботі з програмами для ПК та створення складних веб-сервісів. З'явившись як розширювана скриптова мова, поступово вона стала дуже популярною.

Нові версії Python виходять порівняно часто, а поточні доопрацювання, виправлення помилок і розширення бібліотек ведуться практично на постійній основі.

Широкий функціонал Python забезпечений простим синтаксисом, суворим структуруванням, величезною кількістю фреймворків і бібліотек (їх число досягає декількох тисяч). Python перейняв багато від мов під чийм впливом розроблювався – ABC, Modula-3, C, C++, Smalltalk, Lisp, Fortran, Miranda, Java, Icon, спираючись на практику їх використання, вбирає в себе найкращі рішення та виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем.

Враховуючи факт, що Windows 10 є найвживанішою ОС і, одночасно, найновішою в лінійці ОС Windows програма буде розроблятися саме під дану ОС, хоч може й запуститися на Linux та macOS при попередній незначній доробці.

Оскільки як ОС, на якій буде працювати програма, було обрано Windows та розглядається поширення на Linux та macOS у майбутньому, було прийнято рішення використовувати одну з найбільш адаптованих мов програмування – Python.

2.1.2 Середовище розробки

PyCharm – повнофункціональне інтегроване середовище розробки з підтримкою популярних мов програмування, серед яких Python, JavaScript, CoffeeScript, TypeScript, HTML, CSS та ін. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Linux, macOS.

PyCharm офіційно представлено вперше у 2010 р. З того часу вона постійно доповнювалась і модернізувалась.

Функціональність PyCharm охоплює всі етапи розробки програмного забезпечення, надаючи сучасні інструменти для написання коду, проектування графічних інтерфейсів, збірки, налагодження і тестування програм. Можливості PyCharm можуть бути доповнені шляхом підключення необхідних розширень.

Редактор коду PyCharm підтримує підсвічування синтаксису, вставку фрагментів коду, відображення структури і пов'язаних функцій. Істотно прискорити роботу допомагає технологія IntelliSense – автозавершення коду під час введення.

Вбудований зневаджувач PyCharm використовується для пошуку і виправлення помилок у вихідному коді, в тому числі на низькому апаратному рівні. Інструменти діагностики дозволяють оцінити якість коду з точки зору продуктивності і використання пам'яті.

Дизайнер форм PyCharm незамінний при розробці програм з графічним інтерфейсом, допомагаючи спроектувати зовнішній вигляд майбутньої програми і роботу кожного елемента інтерфейсу.

Нарешті, PyCharm надає комплекс інструментів для автоматизації тестування додатків в частині перевірки роботи інтерфейсів, модульного і навантажувального тестування.

PyCharm має інтегровану підтримку розробки через тестування. Автоматичне формування тестових заглушок і багату інфраструктуру модульного тестування, володіє чудовими можливостями розширення, завдяки чому прямо в рамках PyCharm можуть бути використані різноманітні модулі [15].

Для командних проектів PyCharm пропонує підтримку групової роботи, дозволяючи виконувати спільне редагування і налагодження будь-якій частині коду в реальному часі.

Зважаючи на вищезазначені переваги PyCharm, а, також, інтегрованість в неї обраної мови програмування Python – саме PyCharm обрано в якості оптимального середовища розробки проекту.

2.1.3 Мова програмування Python

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією, розроблена в 1991 році Гвідо ван Россумом у якості привабливої платформи для швидкої розробки як звичайних програм, так і веб-застосунків, а також як засіб поєднування наявних компонентів.

Розробка мови програмування почалася в кінці 1980-х років. Перший випуск датовано 1991 роком.

Провідною ідеєю при розробці Python було забезпечення свободи розробника за рахунок надання йому можливості створювати додатки різних типів, здатні виконуватися на різних типах пристроїв і в різних середовищах. Офіційно володіє Python некомерційна організація Python Software Foundation. Гвідо ван Россум, як засновник Python досі спостерігає за її розвитком, вдосконалює і розширює можливості, приймає остаточні рішення щодо дизайну мови [16].

При розробці Python враховано наступні правила, які ще називають «The Zen of Python»:

- гарне краще за потворне;
- явне краще за неявне;
- просте краще за складне;
- складне краще за заплутане;
- плоске краще за вкладене;
- розріджене краще за щільне;
- легкість читання має значення;
- особливі випадки не є настільки особливими, щоб порушувати правила;
- хоча практичність є важливішою за бездоганність;
- помилки ніколи не повинні проходити непомітно;
- якщо їх приховування не прописано явно;
- зустрівши неоднозначність, опирайтесь спокусі вгадати;
- має бути один – і, бажано, тільки один – спосіб зробити це;
- хоча спочатку він може бути й не очевидним, якщо ви не голландець;
- зараз – краще, ніж ніколи;
- хоча ніколи, найчастіше, – краще, ніж просто зараз;
- якщо реалізацію важко пояснити – задум поганий;
- якщо реалізацію легко пояснити – можливо, задум добрий;
- простори імен – чудова річ, тож робімо їх більше.

Мову Python було обрано через її зазначені вище переваги і високу інтеграцію до PyCharm.

2.2 Огляд застосованих технологій

Використання готових технологій має ряд переваг.

По-перше, розробник нової програми знімає з себе турботу про реалізацію функціональності, закладеної в застосованій технології. Весь цикл розробки технологій здійснюється розробником даної технології. Метод повторного використання коду є тим механізмом, який дозволяє розробникам «встати на плечі гігантів» і швидко будувати нові складні системи з вже налагоджених компонентів.

Друга перевага виходить з самого повторення використання коду, що призводить до суттєвого зменшення розміру підсумкової програми, а при недостатній продуктивності носія і до швидкодії.

Для забезпечення не складного інтерфейсу, а, також підвищення якості і прискорення розробки програми застосовано компонент Qt – інтерфейс програмування додатків, з наближеними до стилю ОС Windows елементами керування. Для застосування компонентів Qt було використано модуль pySide6, що дозволяє використовувати весь функціонал фреймворку у програмах Python. Використання компоненту є безкоштовним.

Python є дуже популярним у сфері досліджень та використання НМ, тож має і багато бібліотек для роботи із ними, та всі мають свої переваги з недоліками. Серед найпопулярніших можна розглядати TensorFlow, PyTorch, Scikit-Learn. Серед усіх найпопулярнішим є TensorFlow – бібліотека з відкритим кодом розроблена Google [17]. Серед переваг над іншими можна виділити абсолютно необхідну можливість роботи зі згортковими нейронними мережами у TensorFlow, наявність готових методів для навчання моделей та наявність детальної технічної документації. Враховуючи вищезазначені факти найкращим кандидатом для роботи з нейронною мережею буде TensorFlow. Проте деякі маніпуляції із зображеннями простіше використовувати за допомогою запропонованих інструментів Scikit-Learn, тож можливе використання декількох бібліотек для різних цілей.

2.2.1 Огляд можливостей Qt

Qt – кросплатформовий інструментарій розробки ПЗ. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє керувати потоками, працювати з мережею та забезпечує кросплатформовий доступ до файлів.

У рамках розробки дипломного проєкту був використаний PySide – прив'язка мови Python до крос-платформного GUI інструментарію Qt. Це є безкоштовний для використання модуль, який можна вільно використовувати у своєму середовищі Python. На відміну від дуже схожого модулю PyQt PySide можна використовувати як у відкритих, так і в закритих комерційних проєктах, оскільки поширюється за ліцензією LGPL.

Qt містить різні елементи управління, які можна додавати у віджети: елементи управління, що відображають текстові поля, кнопки, списки, що розкриваються, перемикачі та навіть веб-сторінки.

Інтерфейс програмування додатків Qt було обрано через його відносно легке налаштування і адаптованість до різних ОС, в тому числі Windows.

2.2.2 Реалізація згорткової НМ U-Net

U-Net використовується не тільки для визначення класу на зображенні, але й для сегментації цього зображення для виділення його області і визначення його точної локалізації.

Архітектура складається зі шляхів звуження та розширення (рис. 2.1). Шлях звуження притаманний архітектурі згорткових мереж, який знижує

розширення зображення. Кожний етап шляху розширення підвищує дискредитацію карти властивостей.

Ця архітектура досягає найкращої продуктивності і точності при сегментації. Метод потребує лише невеликий набір зображень із готовими масками з поміченими сегментами для тренування, яке займає порівняно мало часу.

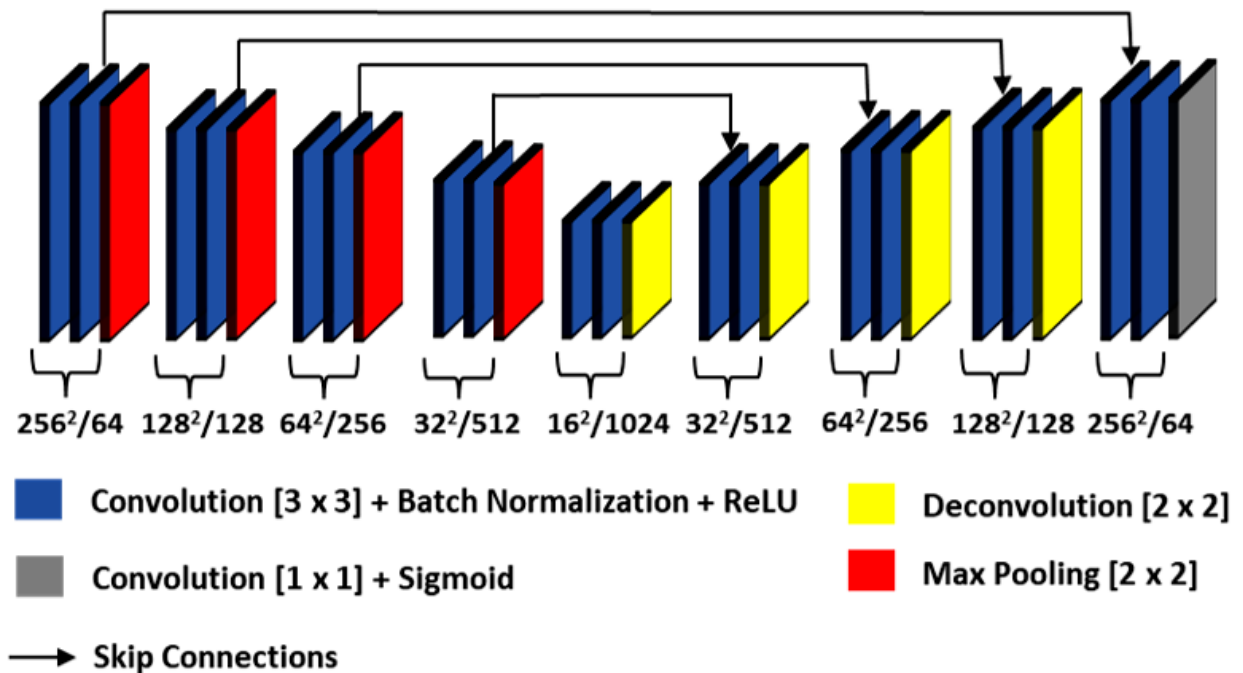


Рисунок 2.1 – Ілюстрація реалізації архітектури U-Net

2.3 Висновки за розділом 2

Було виконано аналіз вимог до проєкту відповідно до мети роботи та визначено програмні засоби для розробки програми.

Програма має передбачати можливість навчання моделей згорткових нейронних мереж (а саме мереж архітектури U-Net). Програма повинна передбачати можливі помилки при виконанні та повідомляти про них відповідними коректними помилками, при цьому мати простий користувацький інтерфейс з можливістю вдосконалення.

Мовою програмування було обрано Python та бібліотеки з модулями tensorflow, opencv-python, scikit-learn, matplotlib, PySide, оскільки вони задовольняють потреби для реалізації ПЗ і забезпечать готові рішення для різноманітного функціоналу. Середовищем розробки обрано PyCharm, бо він пропонує просунуті інструменти розробки, зневадження та контролю за середовищем виконання скриптів Python.

3 РОЗРОБКА І РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ПРОГРАМИ

3.1 Опис основних рішень щодо реалізації системи

Для реалізації системи було обрано мову програмування Python та бібліотеки: tensorflow, opencv-python, scikit-learn, matplotlib. Окрім цих ключових бібліотек також були використані допоміжні PySide, numpy, progressbar, pydot (Python інтерфейс для Graphviz's Dot), imutils, h5py.

Для стабільності оточення було створено віртуальне середовище Python 3.7 яке має свій набір встановлених модулів з чітко визначеними версіями які сумісні між собою.

Для навчання моделі було обрано потужну бібліотеку tensorflow описану у розділі 2. Ваги зберігаються у файлах з розширенням h5 та для роботи із форматом HDF5 використовувався модуль h5py.

Для візуалізації даних, роботи із зображеннями переважно було застосовано засоби представлені у модулях opencv-python та imutils.

3.2 Структура програми

Розроблене програмне забезпечення представляє собою 1 Python проєкт виконаний у PyCharm, який містить 4 папки, 6 Python пакетів стосовно процесів НМ, 26 Python файлів, 13 класів, 1 json файл конфігурацій, 1 текстовий файл зі списком залежностей для швидкої інсталяції необхідних модулів менеджером рір.

В каталозі output/hdf5 знаходяться файли підготовлених наборів даних для тренування та валідації train.hdf5 та val.hdf5 (рис. 3.1).

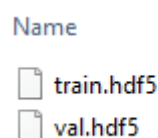


Рисунок 3.1 – Файли HDF5

Структуру проєкта наведено рисунку 3.2.

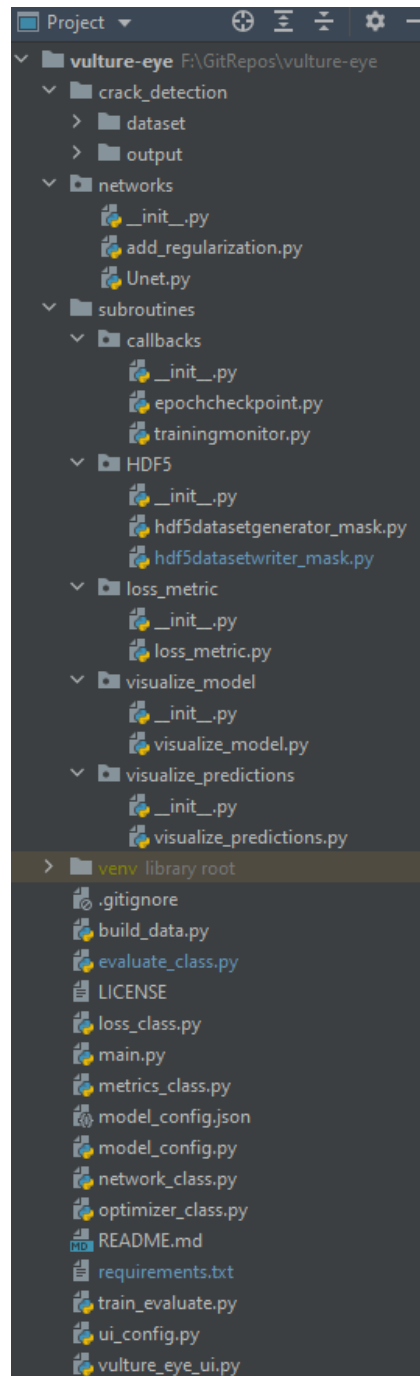


Рисунок 3.2 – Загальна структура розробленої програми

У корені проєкту «vulture-eye» зберігаються усі файли проєкту, в тому числі скрипти, вхідні дані для навчання моделей (зображення та маски), результати її роботи, ваги навчених нейронних мереж кожного покоління.

Директорія `crack_detection` містить у собі дві піддиректорії: `dataset` – дані для тренування моделей, `output` – файли навчених мереж та їх результати роботи.

Директорія `networks` була створена із розрахунком на можливість роботи з різними архітектурами нейронних мереж і містить у собі їх реалізації (на даний момент реалізацію U-Net).

Директорія `subroutines` містить у собі пакети `callbacks`, `HDF5`, `loss_metric`, `visualize_model`, `visualize_predictions`, які відповідальні загалом за процеси підготовки датасетів, моніторингу та візуалізації процесів навчання та розпізнавання.

Скрипт запускається файлом `main.py`, який ініціалізує графічне вікно Qt з якого можна обрати тип операції (підготовка датасету, навчання, розпізнавання).

3.3 Розробка класів та пакетів програми

В програмі наявні 16 класів, які взаємодіють між собою. Пояснення основних функцій класів наведено в п. 3.4.

Клас «`MainWindow`» є графічним інтерфейсом програми. Він відповідальний за відображення форми з якої можна запустити підготування даних, тренування моделі та розпізнавання (рис. 3.3).

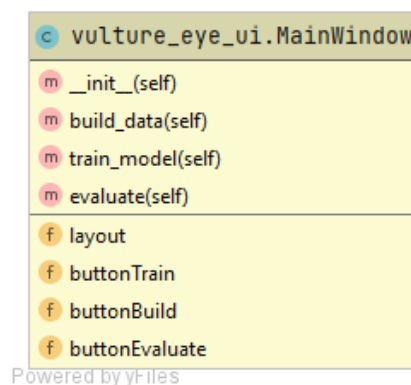


Рисунок 3.3 – Діаграма класу `MainWindow`

Клас «Config» відповідає за конфігурації моделі. Він забезпечує правильну роботу у трьох різних режимах та поєднує у собі усі основні параметри нейронної мережі (рис. 3.4).

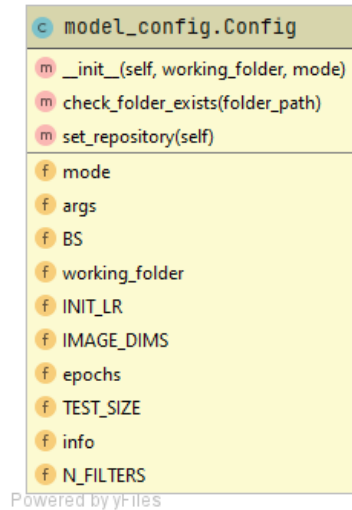


Рисунок 3.4 – Діаграма класу Config

Клас «Network» використовується як уніфікована обгортка над реалізацією моделі U-Net (та потенційно інших архітектур) для спрощеної роботи із нею (рис. 3.5).

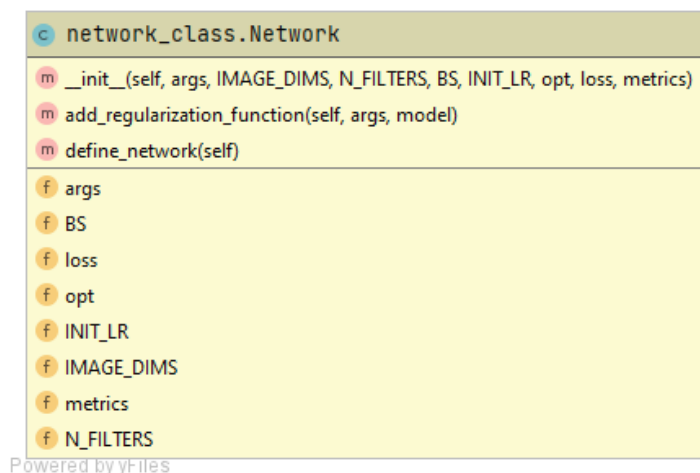


Рисунок 3.5 – Діаграма класу Network

Клас «EpochCheckpoint» створено для покращеного контролю над процесом навчання на кожній епосі моделі та зберігання ваг (рис. 3.63.3).

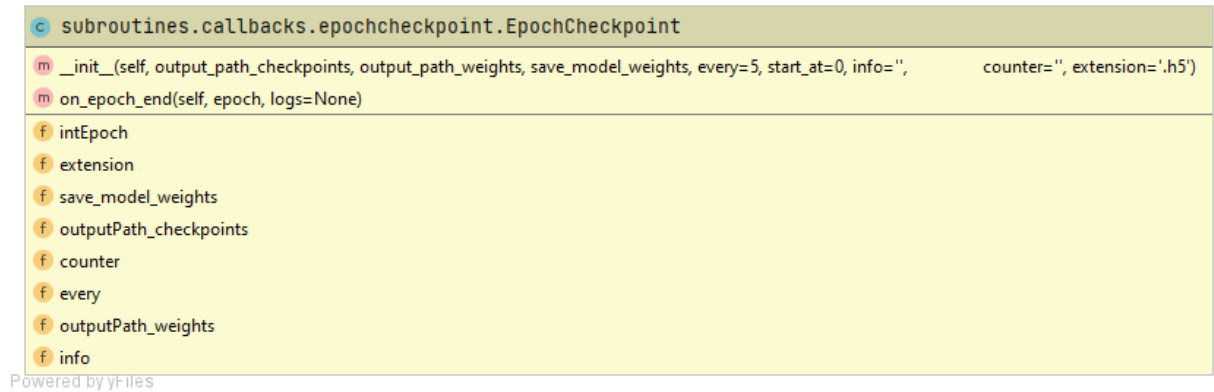


Рисунок 3.6 – Діаграма класу EpochCheckpoint

Клас «HDF5DatasetGeneratorMask» створено для генерації даних у форматі HDF5 (рис. 3.7).

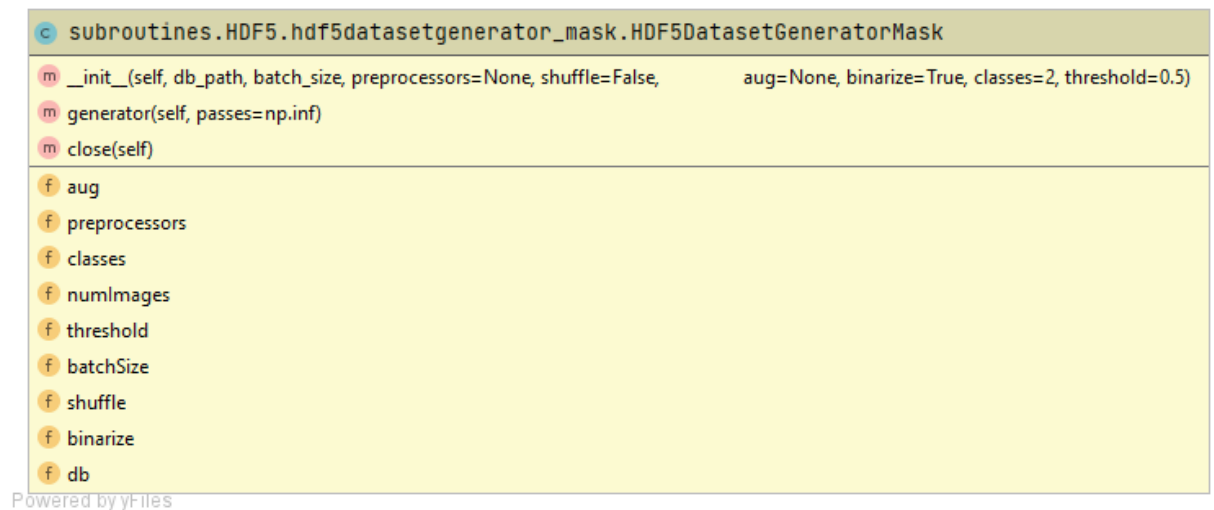


Рисунок 3.7 – Діаграма класу HDF5DatasetGeneratorMask

Клас «HDF5DatasetWriterMask» створено для роботи читання-запису над HDF5 базами даних які містять дані про датасети для навчання (рис. 3.8).

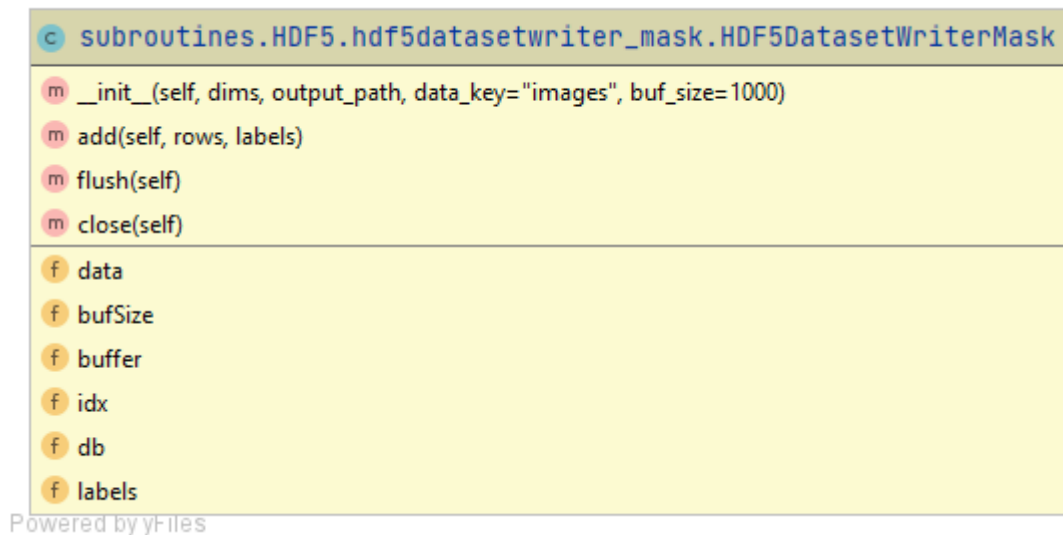


Рисунок 3.8 – Діаграма класу HDF5DatasetWriterMask

3.4 Розробка програмних модулів

Нижче будуть наведені основні пакети, що реалізують програмні застосунки, з прототипами основних функцій та методів. Повний код основних класів наведено у Додатку А.

Програма «VultureEye» ініціалізується наступним чином:

```

import sys
from PySide6.QtWidgets import QApplication
from vulture_eye_ui import MainWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ui = MainWindow()
    sys.exit(app.exec())
  
```

Використовується модуль PySide для створення Qt інтерфейсу з якого потім будуть доступні наступні етапи використання. Усі властивості інтерфейсу описані у класі MainWindow, який містить у собі елементи керування користувацького інтерфейсу.

З інтерфейсу задіюється клас DataBuilder або ModelManipulator в залежності від режиму роботи: підготовка датасету, тренування, розпізнавання.

Основною функцією `DataBuilder` є створення та виклик `HDF5DatasetWriterMask`, щоб передати йому керування над записом інформації про зразкові зображення та їх маски до HDF5 файлів для тренування та валідації.

```
writer = HDF5DatasetWriterMask((len(images_path), IMAGE_DIMS[0], IMAGE_DIMS[1],
IMAGE_DIMS[2]), output_path)
```

Клас `ModelManipulator` відповідає за навчання та розпізнавання завантаженою моделлю нейронної мережі використовуючи підготовлені матеріали після роботи `DataBuilder`. Його головною задачею є використання методів із пакету `tensorflow` та `HDF5DatasetGeneratorMask` для генерації даних на кожній епосі при навчанні, моніторингу процесу та виклику компонентів відповідних за його візуалізацію, та проведення процесу розпізнавання та його візуалізації.

```
model = LoadModel(args, IMAGE_DIMS, BS).load_pretrained_model()
evalGen = HDF5DatasetGeneratorMask(args['EVAL_HDF5'], BS, aug=None, shuffle=False,
binarize=args['binarize'])
```

Класи у файлах `loss_class.py`, `metrics_class.py`, `optimizer_class.py` використовуються для підрахунків точності моделі, метрики, оптимізації процесів, а саме формують необхідні структури використовуючи відповідні об'єкти із пакету `tensorflow`.

3.5 Розробка інтерфейсу користувача, графічні елементи

При розробці інтерфейсу враховано наявність нескладного інтерфейсу. Це забезпечено єдиним стилем програми, близькістю елементів керування програмою до елементів керування ОС Windows 10.

В програмі наявно 1 форма-інтерфейс: головна.

Оскільки файл містить небагато змісту, переважно опис для елементів одного типу (кнопка) для розуміння структури коду і функцій тут можливо навести весь код опису форми. Повний код разом із конфігураціями та викликом форми наведено в додатку А. Пояснення наведено після коду.

```

from PySide6 import QtCore, QtWidgets, QtGui

import ui_config
from build_data import DataBuilder
from train_evaluate import ModelManipulator

class MainWindow(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.buttonBuild = QtWidgets.QPushButton("Build Data")
        self.buttonBuild.clicked.connect(self.build_data)

        self.buttonTrain = QtWidgets.QPushButton("Train Model")
        self.buttonTrain.clicked.connect(self.train_model)

        self.buttonEvaluate = QtWidgets.QPushButton("Evaluate")
        self.buttonEvaluate.clicked.connect(self.evaluate)

        self.layout = QtWidgets.QVBoxLayout(self)
        self.layout.addWidget(self.buttonBuild)
        self.layout.addWidget(self.buttonTrain)
        self.layout.addWidget(self.buttonEvaluate)

        self.setWindowTitle(f'{ui_config.title} v{ui_config.version}')
        self.resize(ui_config.width, ui_config.height)
        self.show()

    def build_data(self):
        self.buttonBuild.setDisabled(True)
        DataBuilder()
        self.buttonBuild.setDisabled(False)

    def train_model(self):
        self.buttonTrain.setDisabled(True)
        ModelManipulator('train')
        self.buttonTrain.setDisabled(False)

    def evaluate(self):
        self.buttonEvaluate.setDisabled(True)
        ModelManipulator('evaluate')
        self.buttonEvaluate.setDisabled(False)

```

Цей код описує розміщення елементів на вікні та властивості елементів розміщених на ньому (кнопки). Для кращого розуміння та візуального сприйняття усі елементи названо зрозумілими іменами, які відображають що саме робить той чи інший елемент керування або метод.

Така конструкція є загальнозживаним рішенням, яке застосовують з метою підвищення читабельності коду.

Методи `build_data`, `train_model`, `evaluate` запускають відповідні процеси підготовки датасету, тренування моделі та розпізнавання.

Кожен графічний елемент інтерфейсу (кнопка, зображення, текстова область та ін.) має наступні властивості:

- `.location` – позиція елемента відносно форми;
- `.size` – розмір елемента на формі;
- `.text` – текстове повідомлення, яке міститься на елементі і відображається користувачеві.

В результаті виконання конструктору `MainWindow` формується інтерфейс вікна. За результатами обробки `vulture_eye_ui.py` формується інтерфейс головної форми програми (рис. 3.9).

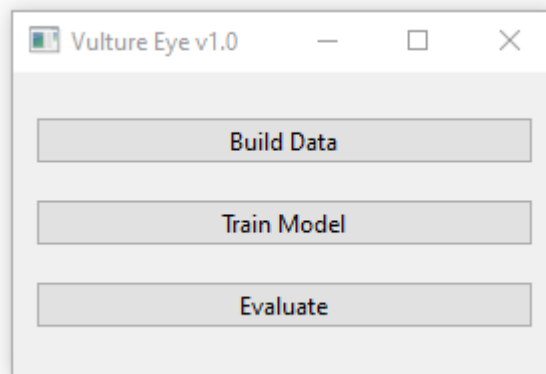


Рисунок 3.9 – Інтерфейс головної форми програми «VultureEye»

Використання потужного пакету `PySide` у майбутніх доробках програмного забезпечення дозволить задіяти усі можливості розглянутої платформи `Qt`, перенести на форми візуалізацію процесів та відображення результатів розпізнавання, та, звичайно, розробляти варіативні інтерфейси для потенційно нового функціоналу.

3.6 Опис процесу тренування (навчання) моделі

Перед початком навчання моделі необхідно скомпонувати дані у структуру із якою зможуть працювати методи представленого програмного забезпечення. Цей процес виконується файлом `build_data.py`. Він зчитує

підготовлені файли зображень і масок та записує структуровані дані у файли формату HDF5 у каталог output.

Переходячи до алгоритму тренування його можна поділити на 3 етапи:

- зчитування конфігурацій із файлу `model_config.py`;
- завантаження моделі із врахуванням цих самих параметрів файлом `train_evaluate.py` із визначенням усіх шляхів для зберігання вихідних файлів;
- тренування моделі за допомогою обгортки Network над U-Net, що використовує засоби tensorflow.

Для розділення логіки було розроблено класи з як різним рівнем абстракції, так і розподіленою відповідальністю за різні за сенсом процеси.

Так, наприклад, побудовою архітектури НМ з визначенням властивостей нейронів та шарів займається лише `Unet.py`, а для запису даних до файлів використовується `hdf5datasetgenerator_mask.py`. Більш детальний опис роботи цих класів представлений у п. 3.3 та 3.4.

В результаті роботи цього функціоналу натренована модель буде знаходитись у каталогах всередині директорії output: `model_json`, `hdf5`, `weights`. Опис моделі, датасету та ваг відповідно. Ці файли потім будуть використані при завантаженні навченої моделі для процесу розпізнавання.

3.7 Опис процесу розпізнавання

Розглянемо процес розпізнавання детальніше:

Етап 1. Ініціалізація:

- ініціалізація об'єкту класу `ModelManipulator` у режимі `evaluate`;
- завантаження моделі з файлів згенерованих на етапі навчання;
- завантаження файлів зображень.

Етап 2. Розпізнавання:

- зчитування зображення;

- генерація передбачень за допомогою засобів tensorflow (model.predict_generator);
- візуалізація результатів розпізнавання (visualize_predictions(args, predictions)), що збереже зображення на жорсткому диску.

3.8 Висновки за розділом 3

Виконавши роботу було розглянуто методи нейромережевого аналізу цифрових зображень та реалізовано ПЗ із використанням найбільш задовільного варіанту. Для комфортного користування було розроблено простий користувацький GUI.

Було описано структуру ПЗ, основні функції, класи та їхній принцип роботи. Продукт має вдалу модульну архітектуру, яку можливо розширювати для майбутнього функціоналу та інтегрувати у більші системи.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Функціональні можливості програми

Розроблена програма вирішує задачі розпізнавання тріщин будівель на цифрових зображеннях.

Розроблена програма має наступні функціональні можливості:

- самостійна підготовка датасету та навчання моделі на його базі;
- виконання розпізнавання об'єктів класу дефект типу тріщина з будь-якого ракурсу зйомки будівлі.

Розроблена програма здатна ефективно вирішувати описані задачі за прийнятний час. Навчання нейромережевої моделі архітектури U-Net виконувалося впродовж 10 епох за допомогою готових рішень реалізованих у пакеті tensorflow. В результаті проведеного навчання було збережено ваги навченої мережі, характеристика кожної епохи, історію зміни похибки, опис моделі та зведення параметрів шарів моделі.

Приклади результатів розпізнавання розробленої програми наведено далі (рис. 4.1). На рисунках можна побачити, що показники точності досить високі, та це враховуючи використання моделі натренованої тільки за 10 епох. Нейронна мережа досягає точності більше 88% на навчальному наборі даних та 96% на тестовому. Тривалість навчання приблизно година якщо навчати мережу впродовж 10 епох на підборі даних, який містить 240 зображень.

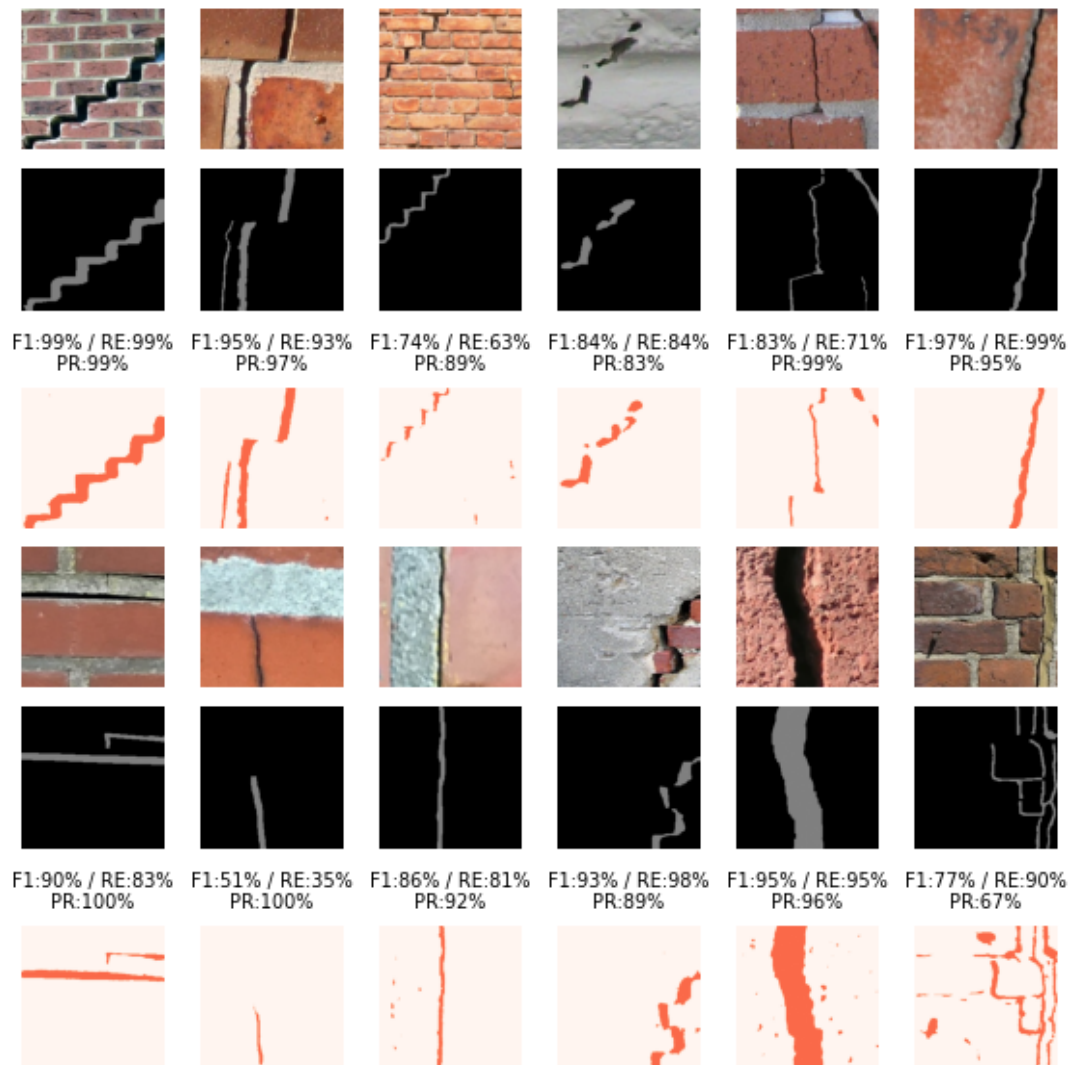


Рисунок 4.1 – Результати розпізнавання

Окрім візуалізації результатів після кожного запуску розпізнавання на етапі навчання проводиться замір та візуалізація похибки моделі (рис. 4.2).

Також для довідки генерується підсумок характеристик моделі з якою буде проводитись розпізнавання (рис. 4.3).

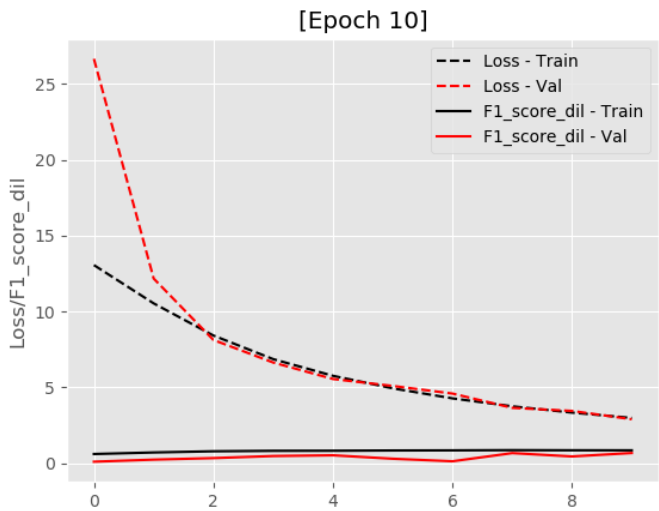


Рисунок 4.2 – Графік похибки

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
img (InputLayer)	[(None, 224, 224, 3)]	0	
conv2d (Conv2D)	(None, 224, 224, 64)	1792	img[0][0]
batch_normalization (BatchNormaliza	(None, 224, 224, 64)	256	conv2d[0][0]
activation (Activation)	(None, 224, 224, 64)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36928	activation[0][0]
batch_normalization_1 (BatchNor	(None, 224, 224, 64)	256	conv2d_1[0][0]
activation_1 (Activation)	(None, 224, 224, 64)	0	batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0	activation_1[0][0]
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73856	max_pooling2d[0][0]
batch_normalization_2 (BatchNor	(None, 112, 112, 128)	512	conv2d_2[0][0]
activation_2 (Activation)	(None, 112, 112, 128)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147584	activation_2[0][0]
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584	activation_2[0][0]
batch_normalization_15 (BatchNo	(None, 112, 112, 128)	512	conv2d_15[0][0]
activation_15 (Activation)	(None, 112, 112, 128)	0	batch_normalization_15[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 224, 224, 64)	73792	activation_15[0][0]
concatenate_3 (Concatenate)	(None, 224, 224, 128)	0	conv2d_transpose_3[0][0] activation_1[0][0]
conv2d_16 (Conv2D)	(None, 224, 224, 64)	73792	concatenate_3[0][0]
batch_normalization_16 (BatchNo	(None, 224, 224, 64)	256	conv2d_16[0][0]
activation_16 (Activation)	(None, 224, 224, 64)	0	batch_normalization_16[0][0]
conv2d_17 (Conv2D)	(None, 224, 224, 64)	36928	activation_16[0][0]
batch_normalization_17 (BatchNo	(None, 224, 224, 64)	256	conv2d_17[0][0]
activation_17 (Activation)	(None, 224, 224, 64)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 224, 224, 1)	65	activation_17[0][0]
Total params: 34,536,897			
Trainable params: 34,525,121			
Non-trainable params: 11,776			

Рисунок 4.3 – Зведення параметрів моделі

4.2 Діаграма послідовностей

Для розуміння сценаріїв використання програми було розроблено діаграму послідовностей, яка демонструє як використовуються компоненти програми користувачем.

Діаграму наведено на рис. 4.4.

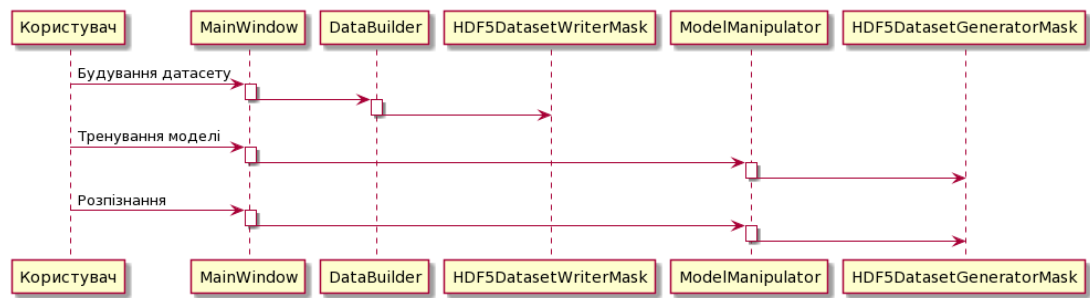


Рисунок 4.4 – Діаграма послідовностей

Опис діаграми послідовностей:

- а) Користувач запускає програму, звертаючись через графічний інтерфейс ОС або термінал;
- б) Користувач обирає одну з наявних кнопок, що запустить виконання у відповідному режимі.

Доступні режими роботи:

- будування датасету: програма генерує із набору зображень структуровані файли;
- тренування моделі: програма генерує файли моделі, ваги, візуалізує деяку статистику, як графік похибки епох;
- розпізнавання: програма аналізує вхідні зображення, генерує файли з візуалізацією результатів у вигляді порівняння оригінальної маски та прогнозу з числовим значенням точності.

4.3 Тестування програми

Для перевірки дієздатності розробленої програми та її відповідності до висунутих функціональних вимог необхідно виконати функціональне тестування (за принципом «чорної скриньки», не поглиблюючись у програмну реалізацію). Метою такого тестування є перевірка

функціональних можливостей програми порівнюючи отримані результати в залежності від приготованих вхідних даних.

Тестування програми проводилось за допомогою декількох перевірок на різних ПК під управлінням ОС Windows та встановленим Python і усіма необхідними модулями.

Список проведених перевірок:

- перевірка працездатності графічного інтерфейсу;
- перевірка стабільності роботи програми;
- перевірка використання ресурсів програмою;
- перевірка повідомлень при запуску в неналаштованому середовищі (за наявності необхідних модулів або версії Python).

За вимогами до розроблюваного програмного забезпечення було сформовано тест-кейси та проведено тестування за ними. Тест-кейси та результати тестувань наведено у таблиці 4.1.

Таблиця 4.1 – Результати тестувань

№	Статус	Сценарій	Очікуваний результат	Отриманий результат
1	Пройдено	Запуск програми в середовищі без необхідних модулів	Повідомлення про відсутність модуля	Очікуваний
2	Пройдено	Запуск програми у сконфігурованому середовищі	Користувач бачить головний інтерфейс	Очікуваний
3	Пройдено	Перевірка необхідних ресурсів системи	Програма використовує зазначені ресурси	Очікуваний
4	Пройдено	Будування датасету за відсутності зображень	Повідомлення про відсутність вхідних даних	Очікуваний
5	Пройдено	Будування датасету	Генеруються файли для тренування і валідації	Очікуваний
6	Пройдено	Тренування моделі за відсутності необхідними файлами	Повідомлення про відсутність файлів	Очікуваний
7	Пройдено	Тренування моделі	Генеруються усі необхідні файли моделі з вагами	Очікуваний
8	Пройдено	Розпізнавання за відсутності необхідних файлів моделі	Повідомлення про помилку завантаження	Очікуваний
9	Пройдено	Розпізнавання	Створення файлів з візуалізацією результатів	Очікуваний

Враховуючи результати тестувань наведені у таблиці 4.1 програма працює стабільно, пройшла усі тест-кейси та відповідає попередньо висунутим вимогам.

4.4 Напрямки подальшого розвитку технології

Розроблена програма в результаті виконання роботи вирішує поставлені задачі та показує задовільні результати.

Проведений аналіз результатів та високі показники точності розпізнавання тріщин будівель показують, що технологія має перспективи для подальшого розвитку. Можлива інтеграція у автоматичний програмно-апаратний комплекс із використанням безпілотних дронів та додавання можливості передачі результатів з таких систем на централізований сервер.

4.5 Висновки за розділом 4

Для зрозумілості структури та принципу роботи програми було складено діаграму послідовностей, яка візуально пояснює влаштування та взаємодію компонентів.

Для перевірки працездатності та правильності роботи ПЗ, а, також, відповідності його до висунутих вимог, було складено тест-кейси за якими перевірилась його робота. Усі тест-кейси були успішно пройдені, що свідчить про успішну розробку і завершення проєкту.

Згідно з проведеним експериментальним дослідженням, можна стверджувати, що розроблена програма придатна для вирішення задачі виявлення тріщин будівель на цифрових зображеннях. Це підтверджено на теорії та підкріплено отриманими практичними результатами.

5 КЕРІВНИЦТВО ПРОГРАМІСТА

5.1 Призначення й умови застосування програми

Найменування продукту – програма «VultureEye». Програма призначена для вирішення задачі виявлення тріщин будівель на цифрових зображеннях за допомогою нейромережових моделей архітектури U-Net.

Область застосування – дослідження міського середовища, аналіз необхідності ремонтних робіт нерухомості, знаходження та попередження будівель аварійного стану.

Дана розробка дає можливість користувачу швидко проаналізувати пакет зображень будівель на наявність тріщин.

Вимоги до складу і параметрів технічних засобів – для функціонування системи необхідний пристрій з наступними параметрами:

- мікропроцесор Intel Pentium 2,0 GHz та вище;
- вільне місце на жорсткому диску 5 Гб; ОЗУ 4 Гб; 1,6 GHz DDR3;
- клавіатура і маніпулятор типу «миша».

Вимоги до інформаційної та програмної сумісності – для функціонування програми необхідна наявність ОС Windows, версії 10 та вище з встановленим Python версії не нижче 3.7.

5.2 Основні вимоги до системи

Вимоги до функціональних характеристик програми «VultureEye»:

- програма повинна надавати можливість будувати датасети на основі підготовлених наборів зображень;
- програма повинна надавати можливість навчати нейронну мережу на основі підготовленого датасету;
- програма повинна генерувати проміжні результати тренування моделі (ваги епох, графік похибки);

- програма повинна розв'язувати задачу виявлення тріщини на зображенні за допомогою натренованої нейронної мережі.

Нефункціональні вимоги до програми:

- програма має реагувати на натискання клавіш миші та клавіатури з затримкою не більше 0.5с;
- програма повинна мати інтуїтивний графічний інтерфейс;
- програма повинна передбачати можливість невірно налаштованого середовища або вхідних файлів та повертати відповідні помилки.

5.3 Характеристики програми

Програма «VultureEye».

Обсяг пам'яті, який займає програма – 4 Гб. Може змінюватися в залежності від використаних зображень для вхідного датасету.

Програма міститься у пам'яті ПК.

Програма запускається у разі необхідності.

Перелік файлів, в яких знаходиться програма:

- main.py – головний файл ініціалізації програми;
- vulture_eye_ui.py – містить клас MainWindow;
- ui_config.py – містить конфігурацію для інтерфейсу;
- build_data.py – містить клас DataBuilder;
- train_evaluate.py – містить клас ModelManipulator
- evaluate_class.py – містить клас LoadModel;
- loss_class.py – містить клас Loss;
- metrics_class.py – містить клас Metrics;
- model_config.py – містить клас Config;
- network_class.py – містить клас Network;
- optimizer_class.py – містить клас Optimizer;
- Unet.py – містить реалізацію архітектури U-Net;
- add_regularization.py – містить метод add_regularization;

- epochcheckpoint.py – містить клас EpochCheckpoint;
- trainingmonitor.py – містить клас TrainingMonitor;
- hdf5datasetgenerator_mask.py – містить клас

HDF5DatasetGeneratorMask;

- hdf5datasetwriter_mask.py – містить клас HDF5DatasetWriterMask;
- loss_metric.py – містить методи відповідальні за міру похибок;
- visualize_model.py – містить методи для візуалізації моделі;
- visualize_predictions.py – містить методи для візуалізації результатів.

Візуальне представлення у файловому менеджері представлено на рисунку 5.1.

Name	Date modified	Type	Size
.git	5/28/2021 6:58 PM	File folder	
.idea	6/2/2021 7:40 AM	File folder	
__pycache__	5/16/2021 10:41 PM	File folder	
crack_detection	5/16/2021 11:19 PM	File folder	
networks	5/16/2021 11:19 PM	File folder	
subroutines	5/16/2021 12:46 AM	File folder	
venv	5/16/2021 6:51 PM	File folder	
.gitignore	5/15/2021 9:06 PM	Text Document	2 KB
build_data.py	5/16/2021 10:33 PM	Python File	4 KB
evaluate_class.py	5/31/2021 6:26 PM	Python File	3 KB
LICENSE	5/15/2021 9:06 PM	File	2 KB
loss_class.py	5/16/2021 3:08 PM	Python File	1 KB
main.py	5/16/2021 12:23 AM	Python File	1 KB
metrics_class.py	5/16/2021 3:17 AM	Python File	1 KB
model_config.json	5/16/2021 9:56 PM	JSON Source File	1 KB
model_config.py	5/16/2021 9:59 PM	Python File	11 KB
network_class.py	5/16/2021 3:09 PM	Python File	2 KB
optimizer_class.py	6/2/2021 3:34 AM	Python File	1 KB
README.md	5/16/2021 11:13 PM	Markdown Source...	1 KB
requirements.txt	5/28/2021 7:02 PM	Text Document	1 KB
train_evaluate.py	5/16/2021 10:29 PM	Python File	8 KB
ui_config.py	5/16/2021 10:22 PM	Python File	1 KB
vulture_eye_ui.py	5/16/2021 10:41 PM	Python File	2 KB

Рисунок 5.1 – Структура програми у файловому менеджері

5.4 Початкові та вихідні дані

Початковими даними для роботи програми є набір даних для навчання який являє собою набір цифрових зображень, файли зі статичною інформацією про процес навчання моделі та файли ваг епох моделі за увесь час її тренування.

Вихідними даними є візуалізовані результати виявлення тріщин будівель на цифрових зображеннях, файли з відомостями про модель, збалансовані ваги, графік втрат точності на різних епохах навчання та підсумок параметрів моделі, фізично збережені на жорсткому диску.

6 КЕРІВНИЦТВО ОПЕРАТОРА

6.1 Призначення програми

Найменування продукту – програма «VultureEye». Програма призначена для вирішення задачі виявлення тріщин будівель на цифрових зображеннях за допомогою нейромережових моделей архітектури U-Net.

Дана розробка дає можливість користувачу швидко проаналізувати пакет зображень будівель на наявність тріщин, а саме:

- програма повинна надавати можливість будувати датасети на основі підготовлених наборів зображень;
- програма повинна надавати можливість навчати нейронну мережу на основі підготовленого датасету;
- програма повинна генерувати проміжні результати тренування моделі (ваги епох, графік похибки);
- програма повинна розв'язувати задачу виявлення тріщини на зображенні за допомогою натренованої нейронної мережі.

6.2 Умови виконання програми

Вимоги до складу і параметрів технічних засобів – для функціонування системи необхідний пристрій з наступними параметрами:

- мікропроцесор Intel Pentium 2,0 GHz та вище;
- вільне місце на жорсткому диску 5 Гб; ОЗУ 4 Гб; 1,6 GHz DDR3;
- клавіатура і маніпулятор типу «миша».

Вимоги до інформаційної та програмної сумісності – для функціонування програми необхідна наявність ОС Windows, версії 10 та вище з встановленим Python версії не нижче 3.7.

6.3 Виконання програми

Для початку роботи необхідно запустити файл main.py.

На рисунку 6.1 наведено інтерфейс програми «VultureEye» з позначками кнопок керування.

Нижче наведено пояснення позначок.

Build Data – Кнопка, що будує структуровані датасети із заготовлених наборів зображень.

Train Model – Кнопка, що надає можливість запустити тренування моделі на основі підготовленого датасету.

Evaluate – Кнопка, що надає можливість запустити розпізнавання зображень для виявлення на них тріщин будівель.

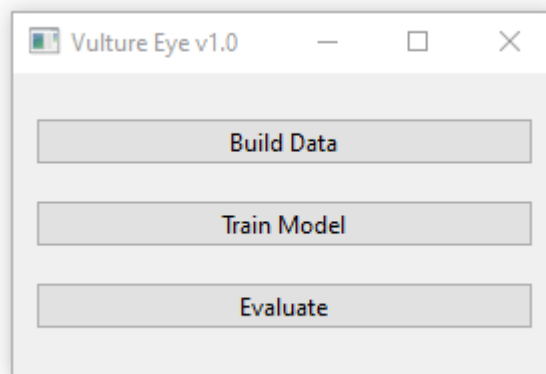


Рисунок 6.1 – Інтерфейс програми «VultureEye»

Після завершення роботи програма зберігає налаштування встановлені користувачем у файлах конфігурації та вихідних файлах моделі.

Результати роботи можливо знайти у каталозі output/predictions (рис. 6.2).

Графік похибки епох знаходиться одразу в каталозі output (рис. 6.3).



Рисунок 6.2 – Результати розпізнавання програми «VultureEye»

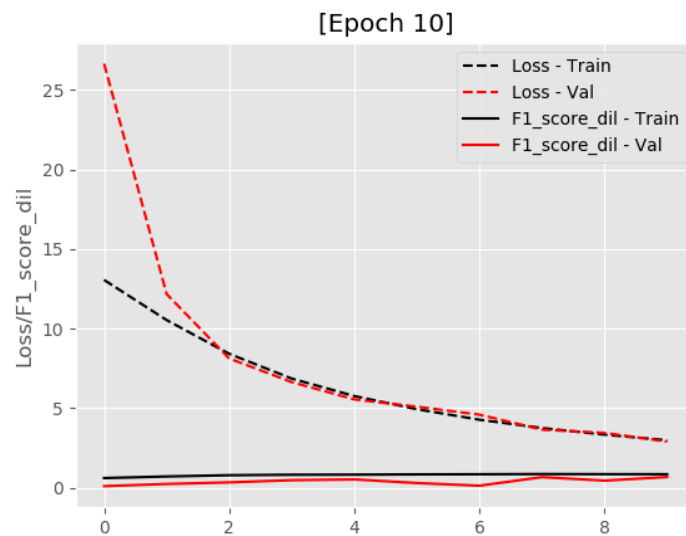


Рисунок 6.3 – Похибка впродовж навчання у програмі «VultureEye»

ВИСНОВКИ

Під час розробки програми для дипломної кваліфікаційної роботи бакалавра було реалізовано рішення на основі існуючих нейромережевих методів розпізнавання об'єктів на зображеннях для рішення задачі виявлення тріщин будівель.

Проведено детальний аналіз предметної області та існуючих методів рішення схожих задач. Визначено, що для побудови програмного забезпечення треба використати згорткову нейронну мережу архітектури U-Net. Виконано дослідження архітектури та основних властивостей з перевагами при використанні цієї нейронної мережі для вирішення задач поставленого типу.

Для проєктування була обрана мова програмування Python, бо відповідала основним вимогам та можливостям, що висувалися до мови програмування.

Середовищем розробки обрано PyCharm, бо надає широкі можливості розробнику в проєктуванні коду, розробці дизайну інтерфейсу користувача, значно спрощені деякі операції та має високу адаптованість до ОС Windows.

Реалізовано структуру програми та необхідні класи для її роботи. Застосовано реалізацію архітектури НМ U-Net, обрано набори навчальних та перевірних даних. Розроблено інтуїтивно зрозумілий графічний інтерфейс користувача. Виконано експериментальне дослідження та тестування, побудовано діаграму послідовностей.

Під час розробки програмного коду було вивчено та застосовано:

- роботу з Qt елементами інтерфейсу у скриптах Python;
- існуючі методи розпізнавання об'єктів на цифрових зображеннях.

Створений проєкт відповідає меті розробки, всім вимогам, що були пред'явлені до програми (безпомилково працює під керуванням ОС Windows, версії 10 з встановленим Python версії 3.7, супроводжується

керівництвом програміста, керівництвом оператора, пояснювальною запискою на розроблені додатки, вихідним кодом програми та ін.).

За результатами роботи можна стверджувати, що програма здатна розпізнавати тріщини будинків із точністю щонайменше 96% при мінімальній тривалості навчання яка може бути покращена ще більше.

Програма має зацікавити будівничі компанії або компанії, що надають дослідницькі дані таким будівничим компаніям, бо задовольняє їх потреби у процесі збору даних.

Шляхи удосконалення:

- класифікація тріщин, детальна довідка про їх площу, довжину, інше;
- покращення виявлення порівняно тонких тріщин;
- вдосконалення графічного інтерфейсу, вивід результатів у табличному вигляді;
- додавання мовних пакетів;
- інтеграція із автоматичним програмно-апаратним комплексом із застосуванням дронів.

Основні ідеї щодо застосування роботи було представлено в тезах доповіді науково-практичної конференції Тиждень науки – 2021, Судаков В. Р. «Розробка програмного забезпечення для реалізації методів розпізнавання тріщин будівель».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про затвердження Порядку проведення обстеження прийнятих в експлуатацію об'єктів будівництва: [Електрон. ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/257-2017-%D0%BF#Text>
2. Case Study on Cracks in Public Buildings and their Remedies, [Electronic resource]. – Access mode: <https://www.ijsr.net/archive/v6i5/ART20173175.pdf>
3. Субботін, С. О. Нейронні мережі: теорія та практика: навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євєнчук, 2020. – 184 с.
4. Backpropagation applied to handwritten zip code recognition / [Y. Lecun, B. Boser, JS. Denker et al.] // Neural Computation. – 1989. – Vol. 1, № 4. – P. 541–551.
5. ISBI Challenge: Segmentation of neuronal structures in EM stacks, [Electronic resource]. – Access mode: http://brainiac2.mit.edu/isbi_challenge/
6. Grand Challenges in Dental X-ray Image Analysis, [Electronic resource]. – Access mode: <http://www-ontust.edu.tw/~cweiwang/ISBI2015/challenge2/index.html>
7. Cell Tracking Challenge, [Electronic resource]. – Access mode: http://www.codesolorzano.com/celltrackingchallenge/Cell_Tracking_Challenge/Welcome.html
8. Our U-net wins two Challenges at ISBI 2015, [Electronic resource]. – Access mode: <https://lmb.informatik.uni-freiburg.de/people/ronneber/isbi2015/>
9. U-Net: Convolutional Networks for Biomedical Image Segmentation, [Electronic resource]. – Access mode: <https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>
10. Субботін, С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навч. посіб. / С. О. Субботін. Запоріжжя: ЗНТУ, 2008. – 341 с.

11. Технічне обстеження будівель і споруд, [Electronic resource]. – Access mode: <https://mb.expert/tehnichne-obstezhennya-budivel-i-sporud/>
12. Чисті програми: Windows. [Електрон. ресурс]. – Режим доступу: <https://venturebeat.com/2019/09/01/net-applications-windows-10-windows-7-market-share>
13. Ботт Е. Windows 10 для ІТ-спеціалістів, пер. с англ.: [Текст]/ Е. Ботт – М.: Економ Паблишерз, 2016. – 173 с.
14. The History of Python: A Brief Timeline of Python, [Electronic resource]. – Access mode: <https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html>
15. PyCharm Features. [Electronic resource]. – Access mode: <https://www.jetbrains.com/pycharm/features/>
16. Python Creator Scripts Inside Google, [Electronic resource]. – Access mode: <https://www.eweek.com/development/python-creator-scripts-inside-google/>
17. TensorFlow. [Electronic resource]. – Access mode: <https://opensource.google/projects/tensorflow>
18. Трещины в домах и дырявые крыши. [Электрон. ресурс]. – Режим доступа: <https://ria56.ru/posts/treshhiny-v-domax-i-dyryavye-kryshi-zhiteli-krasnoshhekovo-zhaluyutsya-na-kommunalnyj-ad.htm>

ДОДАТОК А
Текст програми

main.py

```
import sys
from PySide6.QtWidgets import QApplication
from vulture_eye_ui import MainWindow
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ui = MainWindow()
    sys.exit(app.exec())
```

vulture_eye_ui.py

```
from PySide6 import QtCore, QtWidgets, QtGui
```

```
import ui_config
from build_data import DataBuilder
from train_evaluate import ModelManipulator
```

```
class MainWindow(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()

        self.buttonBuild = QtWidgets.QPushButton("Build Data")
        self.buttonBuild.clicked.connect(self.build_data)

        self.buttonTrain = QtWidgets.QPushButton("Train Model")
        self.buttonTrain.clicked.connect(self.train_model)

        self.buttonEvaluate = QtWidgets.QPushButton("Evaluate")
        self.buttonEvaluate.clicked.connect(self.evaluate)

        self.layout = QtWidgets.QVBoxLayout(self)
        self.layout.addWidget(self.buttonBuild)
        self.layout.addWidget(self.buttonTrain)
        self.layout.addWidget(self.buttonEvaluate)

        self.setWindowTitle(f'{ui_config.title} v{ui_config.version}')
        self.resize(ui_config.width, ui_config.height)
        self.show()

    def build_data(self):
        self.buttonBuild.setDisabled(True)
        DataBuilder()
        self.buttonBuild.setDisabled(False)

    def train_model(self):
        self.buttonTrain.setDisabled(True)
        ModelManipulator('train')
        self.buttonTrain.setDisabled(False)

    def evaluate(self):
        self.buttonEvaluate.setDisabled(True)
        ModelManipulator('evaluate')
        self.buttonEvaluate.setDisabled(False)
```

ui_config.py

```
title = 'Vulture Eye'
```

```
version = 1.0
width = 800
height = 600
```

model_config.py

```
import os
import sys
import json

class Config:
    def __init__(self, working_folder, mode):
        self.working_folder = working_folder
        self.mode = mode # 'train', 'evaluate', 'build_data'

        with open('model_config.json', 'r') as f:
            config = json.load(f)
            self.info = config['info'] # String that will be used as prefix of any
output files
            self.IMAGE_DIMS = config['IMAGE_DIMS'] # Dimensions of the images that
will be fed to the network
            self.BS = config['BS'] # Batch size
            self.epochs = config['epochs'] # Number of epochs to train the model
            self.INIT_LR = config['INIT_LR'] # Initial learning rate

            self.N_FILTERS = 64 # Number of filters for Unet

            # When building the HDF5 database from the images of the dataset
            # the dataset will be split into train and validation
            # define the portion (0 to 1) of the whole dataset that will be used for
validation
            self.TEST_SIZE = 0.40

            # The parameters of the configuration used will be stored in the dictionary
args
            self.args = {
                'model': 'Unet', # 'Deeplabv3', 'Unet', 'DeepCrack', 'sm_Unet_mobilenet'
                'regularization': 0.001, # regularization (for Unet and SM networks):
None or insert a value (e.g. 0.0001)
                'opt': 'Adam', # Optimizer: 'SGD' or 'Adam' or 'RMSprop'
                'aug': False, # Whether data augmentation will be used: True or False
                'dropout': None, # Dropout (for Unet and SM networks): None or insert a
value (e.g. 0.5)
                'batchnorm': True, # Batch normalization (for Unet): True or False
                'save_model_weights': 'weights', # Whether to save the whole model or
only the weights: 'model' or 'weights'
                'init': 'he_normal', # Layer weight initializers (for Unet)
kernel_initializer: 'he_normal' or 'glorot_uniform' or 'random_uniform'
                'encoder_weights': 'imagenet', # If 'imagenet' is used the backbone
network will be initialized with weights pretrained on ImageNet None or 'imagenet'
(for SM networks)
                'start_epoch': 0, # TrainingMonitor
                'every': 5, # TrainingMonitor
                'metric_to_plot': 'F1_score_dil', # Metric to be used for
TrainingMonitor and ModelCheckpoint
                'binarize': True, # Whether data generators will binarize masks: True or
False
                'loss': 'WCE' # Loss Function
            }
}
```

```

# Define Loss Function
# 'focal_loss'
# 'WCE': Weighted Cross-Entropy
# 'BCE': Binary Cross-Entropy
# 'F1_score_loss'
# 'F1_score_loss_dil': Background pixels predicted as cracks (FP) are
considered as
#
#                               TP if they are a few pixels apart from the annotated
cracks
#
#                               Refer to the Journal paper for extra clarification
# define alpha and gamma values only when focal loss is used
if self.args['loss'] == 'focal_loss':
    self.args['focal_loss_a'] = 0.25 # alpha value when focal loss is used
default: 0.25
    self.args['focal_loss_g'] = 2 # gamma value when focal loss is used.
default: 2.
# Define WCE_beta only when WCE loss is used
elif self.args['loss'] == 'WCE': # Weighted Cross-Entropy
    # For extra documentation regarding 'WCE_beta' see the following link
    #
https://www.tensorflow.org/api\_docs/python/tf/nn/weighted\_cross\_entropy\_with\_logits
    self.args['WCE_beta'] = 10 # weight of the positive class (i.e. crack
class)

@staticmethod
def check_folder_exists(folder_path):
    """
    check if folder exists and if not create it
    """

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

def set_repository(self):
    # Pass the path where the repository is stored to the args
    self.args['main'] = self.working_folder

    # Where the dataset is stored
    self.args['dataset'] = os.path.join(self.args['main'], 'dataset')

    # A new folder will be created where any output will be stored in
    self.args['output'] = os.path.join(self.args['main'], 'output')

    # The dataset will be stored in HDF5 format here
    self.args['hdf5'] = os.path.join(self.args['output'], 'hdf5')

    # The saved model will be stored here
    self.args['checkpoints'] = os.path.join(self.args['output'], 'checkpoints')

    # The saved weights will be stored here
    self.args['weights'] = os.path.join(self.args['output'], 'weights')

    # The serialized model (JSON) will be stored here
    self.args['model_json_folder'] = os.path.join(self.args['output'],
'model_json')

    # Predictions will be stored here
    self.args['predictions'] = os.path.join(self.args['output'], 'predictions')

    # Create the folders
    folders = [

```

```

        self.args['hdf5'],
        self.args['checkpoints'],
        self.args['weights'],
        self.args['model_json_folder'],
        self.args['predictions']
    ]
    for f in folders:
        self.check_folder_exists(f)

    # Save the HDF5 file to different folder according to IMAGE_DIMS
    temp =
    f'{self.info}_{self.IMAGE_DIMS[0]}_{self.IMAGE_DIMS[1]}_{self.IMAGE_DIMS[2]}'
    self.check_folder_exists(os.path.join(self.args['hdf5'], temp))
    # Define the output for the train and validation HDF5 files
    self.args['TRAIN_HDF5'] = os.path.join(self.args['hdf5'], temp, 'train.hdf5')
    self.args['VAL_HDF5'] = os.path.join(self.args['hdf5'], temp, 'val.hdf5')
    # In case you need to test the model on a set other than the validation set,
    define the EVAL_HDF5 suitably
    self.args['EVAL_HDF5'] = os.path.join(self.args['hdf5'], temp, 'val.hdf5')

    # Define the path that the patches of images and masks are stored
    self.args['images'] = os.path.join(self.args['dataset'],
    f'{self.info}_{self.IMAGE_DIMS[0]}_images')
    self.args['masks'] = os.path.join(self.args['dataset'],
    f'{self.info}_{self.IMAGE_DIMS[0]}_masks')

    # Path to the counter file
    self.args['counter_file'] = os.path.join(self.args['output'], 'counter.txt')

    # Different configurations when mode is 'train' or evaluate'
    if self.mode == 'train':
        # When running different trials, the output files will not overwrite the
        existing ones
        # The output files of each trial will have a different suffix
        #
        # Check in the output folder if there is a file named 'counter.txt'
        # The file should contain only a number that will be used as counter
        # If the counter file doesn't exist, use the os.getpid() as counter

        # If 'counter.txt' exists read the counter
        if os.path.exists(self.args['counter_file']):
            with open(self.args['counter_file'], 'r') as file:
                self.args['counter'] = int(file.read())

        # Ask whether to change counter. acceptable answers are 'y' or 'n'
        # If anything else is given as input, ask again
        while True:
            self.args['counter_check'] = input(f'Shall I change the counter
            \"{self.args[\"counter_file\"]}\" [y/n]: ')
            if self.args['counter_check'] == 'y' or
            self.args['counter_check'] == 'n':
                break
            else:
                continue

        # If input was 'y', change the counter
        if self.args['counter_check'] == 'y':
            self.args['counter'] += 1
            with open(self.args['counter_file'], 'w') as file:
                file.write(str(self.args['counter']))

```

```

# If the counter file doesn't exist, use the os.getpid() as counter
else:
    self.args['counter'] = os.getpid()

# Print the counter
print('Counter for training: ', self.args['counter'])

# Store results (i.e. metrics, loss) to CSV format
# Check if the counter value was used before
# If it was used before ask the user whether to proceed
# If 'n' is passed, the analysis will be terminated
self.args['CSV_PATH'] = os.path.join(self.args['output'],
f'{self.info}_{self.args["counter"]}.out')
if os.path.exists(self.args['CSV_PATH']):
    print(f'The counter {self.args["counter"]} has been used
before\nShould the analysis continue [y/n]:')
    check = input()
    if check == 'n':
        print('The analysis will be terminated')
        sys.exit(1)

# Plot Loss/Metrics during training
self.args['FIG_PATH'] = os.path.join(self.args['output'],
f'{self.info}_{self.args["counter"]}.png')
# Serialize results (i.e. metrics, loss) to JSON
# If None is given, no serialization will take place
self.args['JSON_PATH'] = os.path.join(self.args['output'],
f'{self.info}_{self.args["counter"]}.json')
# Plot the architecture of the network
self.args['architecture'] = os.path.join(self.args['output'],
f'{self.info}_architecture_{self.args["counter"]}.png')
# Store the summary of the model to txt file
self.args['summary'] = os.path.join(self.args['output'],
f'{self.info}_summary_{self.args["counter"]}.txt')

elif self.mode == 'evaluate':
    # Define the counter suitably in order to read the correct JSON file etc.
    with open(self.args['counter_file'], 'r') as file:
        self.args['counter'] = int(file.read())
    # Define the file with the pretrained weights or the model with weights
    that will be used to evaluate model
    # e.g. 'crack_detection_1_epoch_7_F1_score_dil_0.762.h5'
    self.args['pretrained_filename'] =
'crack_detection_1_epoch_10_F1_score_dil_0.795.h5'
    # Define the subfolder where predictions will be stored
    self.args['predictions_subfolder'] =
os.path.join(self.args['predictions'], self.args['pretrained_filename'])
    # Define whether to dilate ground truth mask for the calculation of
    Precision metric
    # Background pixels predicted as cracks (FP) are considered as TP if they
    are a few
    # pixels apart from the annotated cracks. Refer to the Journal paper for
    extra clarification
    self.args['predictions_dilate'] = True # True or False

# Configurations to be used both for 'training' and 'evaluation' modes
if (self.mode == 'train') or (self.mode == 'evaluate'):
    # The path for the serialized model to JSON
    self.args['model_json'] = os.path.join(self.args['model_json_folder'],
f'{self.info}_{self.args["counter"]}.json')
    return self.args

```

model_config.json

```
{
  "info": "crack_detection",
  "IMAGE_DIMS": [224, 224, 3],
  "BS": 4,
  "epochs": 10,
  "INIT_LR": 0.0005
}
```

build_data.py

```
import os
import sys
from sklearn.model_selection import train_test_split
from skimage.transform import resize
from imutils import paths
import numpy as np
import progressbar
import cv2

from subroutines.HDF5 import HDF5DatasetWriterMask
from model_config import Config

class DataBuilder:
    def __init__(self):
        folder = {}
        # Use this to easily run the code in different directories/devices
        folder['initial'] = os.path.dirname(os.path.realpath(__file__))
        # The path where the program is stored
        folder['main'] = os.path.join(folder['initial'], 'crack_detection')

        # if folder['main'] == '', then the current working directory will be used
        if folder['main'] == '':
            folder['main'] = os.getcwd()

        sys.path.append(folder['main'])

        cnf = Config(folder['main'], 'build_data')
        args = cnf.set_repository()

        # Set some parameters
        IMAGE_DIMS = cnf.IMAGE_DIMS

        # grab the paths to the images and masks
        train_paths = list(paths.list_images(args['images']))
        train_labels = list(paths.list_images(args['masks']))

        # perform stratified sampling from the training set to build the
        # testing split from the training data
        split = train_test_split(
            train_paths,
            train_labels,
            test_size=cnf.TEST_SIZE,
            random_state=42
        )
        (train_paths, val_paths, train_labels, val_labels) = split

        # construct a list pairing the training, validation, and testing
```

```

# image paths along with their corresponding labels and output HDF5
# files
datasets = [
    ('train', train_paths, train_labels, args['TRAIN_HDF5']),
    ('val', val_paths, val_labels, args['VAL_HDF5'])
]

# loop over the dataset tuples
for (d_type, images_path, masks_path, output_path) in datasets:
    # create HDF5 writer
    print('[INFO] building {}'.format(output_path))
    writer = HDF5DatasetWriterMask((len(images_path), IMAGE_DIMS[0],
IMAGE_DIMS[1], IMAGE_DIMS[2]), output_path)

    # initialize the progress bar
    widgets = [
        'Building Dataset: ',
        progressbar.Percentage(),
        '\n',
        progressbar.Bar(),
        '\n',
        progressbar.ETA()
    ]
    pbar = progressbar.ProgressBar(
        maxval=len(images_path),
        widgets=widgets
    ).start()

    # loop over the image paths
    for (i, (im_path, mask_path)) in enumerate(zip(images_path, masks_path)):
        # load the image and process it
        image = cv2.imread(im_path)

        # resize image if dimensions are different
        if IMAGE_DIMS != image.shape:
            image = resize(image, (IMAGE_DIMS), mode='constant',
preserve_range=True)

        # normalize intensity values: [0,1]
        image = image / 255

        # label
        mask = cv2.imread(mask_path, 0)

        # resize image if dimensions are different
        if IMAGE_DIMS[0:2] != mask.shape:
            mask = resize(mask, (IMAGE_DIMS[0], IMAGE_DIMS[1]),
mode='constant', preserve_range=True)

        # normalize intensity values: [0,1]
        mask = np.expand_dims(mask, axis=-1)
        mask = mask / 255

        # add the image and label to the HDF5 dataset
        writer.add([image], [mask])

        # update progress bar
        pbar.update(i)

    # close the progress bar and the HDF5 writer
    pbar.finish()

```

```
writer.close()
```

train_evaluate.py

```
import sys
import os
import tensorflow as tf

from subroutines.HDF5 import HDF5DatasetGeneratorMask
from model_config import Config

class ModelManipulator:
    def __init__(self, mode):
        folder = {}
        # Use this to easily run the code in different directories/devices
        folder['initial'] = os.path.dirname(os.path.realpath(__file__))
        # The path where the program is stored
        folder['main'] = os.path.join(folder['initial'], 'crack_detection')

        # if folder['main'] == '', then the current working directory will be used
        if folder['main'] == '':
            folder['main'] = os.getcwd()

        sys.path.append(folder['main'])

        cnf = Config(folder['main'], mode)
        args = cnf.set_repository()

        # Set some parameters
        IMAGE_DIMS = cnf.IMAGE_DIMS
        BS = cnf.BS
        epochs = cnf.epochs
        INIT_LR = cnf.INIT_LR
        N_FILTERS = cnf.N_FILTERS
        info = cnf.info

        # When using DeepCrack, eager execution needs to be enabled
        if args['model'] == 'DeepCrack':
            tf.enable_eager_execution()

        if cnf.mode == 'train':
            from tensorflow.keras.preprocessing.image import ImageDataGenerator
            from tensorflow.keras.callbacks import ModelCheckpoint
            from tensorflow.keras.callbacks import CSVLogger

            from subroutines.callbacks import EpochCheckpoint
            from subroutines.callbacks import TrainingMonitor
            from subroutines.visualize_model import visualize_model

            # Prepare model for training

            # Define metrics
            from metrics_class import Metrics
            metrics = Metrics(args).define_Metrics()

            # Define loss
            from loss_class import Loss
            loss = Loss(args).define_loss()

            # Define optimizer
```

```

from optimizer_class import Optimizer
opt = Optimizer(args, INIT_LR).define_optimizer()

# Define Network and compile model
from network_class import Network
model = Network(args, IMAGE_DIMS, N_FILTERS, BS, INIT_LR, opt, loss,
metrics).define_network()

# Visualize model
try:
    visualize_model(model, args['architecture'], args['summary'])
except:
    from subroutines.visualize_model import visualize_model_tf
    visualize_model_tf(model, args['architecture'], args['summary'])

# Data augmentation for training and validation sets
if args['aug']:
    aug = ImageDataGenerator(
        rotation_range=20,
        zoom_range=0.15,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.15,
        horizontal_flip=True,
        fill_mode='nearest'
    )
else:
    aug = None

# Load data generators
trainGen = HDF5DatasetGeneratorMask(args['TRAIN_HDF5'], BS, aug=aug,
shuffle=False, binarize=args['binarize'])
valGen = HDF5DatasetGeneratorMask(args['VAL_HDF5'], BS, aug=aug,
shuffle=False, binarize=args['binarize'])

# Callback that streams epoch results to a CSV file
# https://keras.io/api/callbacks/csv_logger/
csv_logger = CSVLogger(args['CSV_PATH'], append=True, separator=';')

# serialize model to JSON
try:
    model_json = model.to_json()
    with open(args['model_json'], 'w') as json_file:
        json_file.write(model_json)
except:
    pass

# Define whether the whole model or the weights only will be saved from
the ModelCheckpoint
# Refer to the documentation of ModelCheckpoint for extra details
# https://keras.io/api/callbacks/model_checkpoint/
temp =
f'{{info}}_{{args["counter"]}}_epoch_{{epoch}}_{{args["metric_to_plot"]}}_{{val_{{args["metric_to_plot"]}}:.3f}}.h5'

if args['save_model_weights'] == 'model':
    ModelCheckpoint_file = os.path.join(args['checkpoints'], temp)
    save_weights_only = False
elif args['save_model_weights'] == 'weights':
    ModelCheckpoint_file = os.path.join(args['weights'], temp)
    save_weights_only = True

```

```

epoch_checkpoint = EpochCheckpoint(
    args['checkpoints'],
    args['weights'],
    args['save_model_weights'],
    every=args['every'],
    start_at=args['start_epoch'],
    info=info,
    counter=args['counter']
)

training_monitor = TrainingMonitor(
    args['FIG_PATH'],
    json_path=args['JSON_PATH'],
    start_at=args['start_epoch'],
    metric=args['metric_to_plot']
)

model_checkpoint = ModelCheckpoint(
    ModelCheckpoint_file,
    monitor=f'val_{args["metric_to_plot"]}',
    verbose=1,
    save_best_only=True,
    mode='max',
    save_weights_only=save_weights_only
)

# Construct the set of callbacks
callbacks = [
    csv_logger,
    epoch_checkpoint,
    training_monitor,
    model_checkpoint
]

# Train the network

H = model.fit_generator(
    trainGen.generator(),
    steps_per_epoch=trainGen.numImages // BS,
    validation_data=valGen.generator(),
    validation_steps=valGen.numImages // BS,
    epochs=epochs,
    max_queue_size=BS * 2,
    callbacks=callbacks,
    verbose=1
)

elif cnf.mode == 'evaluate':
    # load pretrained model/weights
    from evaluate_class import LoadModel

    model = LoadModel(args, IMAGE_DIMS, BS).load_pretrained_model()

    # Do not use data augmentation when evaluating model: aug=None
    evalGen = HDF5DatasetGeneratorMask(args['EVAL_HDF5'], BS, aug=None,
    shuffle=False, binarize=args['binarize'])

    # Use the pretrained model to generate predictions for the input samples
    from a data generator
    predictions = model.predict_generator(

```

```

        evalGen.generator(),
        steps=evalGen.numImages // BS + 1,
        max_queue_size=BS * 2,
        verbose=1
    )

    # Define folder where predictions will be stored
    predictions_folder = os.path.join(args['predictions'],
args['pretrained_filename'])
    # Create folder where predictions will be stored
    cnf.check_folder_exists(predictions_folder)

    # Visualize predictions
    # Create a plot with original image, ground truth and prediction
    # Show the metrics for the prediction
    # Output will be stored in a subfolder of the predictions folder
    (args['predictions_subfolder'])
    from subroutines.visualize_predictions import visualize_predictions
    visualize_predictions(args, predictions)

```

evaluate_class.py

```

import sys
import os
from tensorflow.keras.models import model_from_json

class LoadModel:
    def __init__(self, args, IMAGE_DIMS, BS):
        self.args = args
        self.IMAGE_DIMS = IMAGE_DIMS
        self.BS = BS

    def load_pretrained_model(self):
        # Load pretrained model
        # This option is not supported for the current version of the code for the
'evaluation' mode
        # Print an explanatory comment and exit
        if self.args['save_model_weights'] == 'model':
            raise ValueError("The option to load a model is not supported for the
'evaluation' mode." +
                "In case you need to use the pretrained model to perform
predictions, then" +
                "train the model with the option: args['save_model_weights'] ==
'weights'" +
                "\nThe analysis will be terminated")

        # Load model from JSON file and then load pretrained weights
        else:
            # load json and create model
            json_file = open(self.args['model_json'], 'r')
            loaded_model_json = json_file.read()
            json_file.close()
            model = model_from_json(loaded_model_json)

            # load weights into new model
            model.load_weights(os.path.join(self.args['weights'],
self.args['pretrained_filename']))

        return model

```

loss_class.py

```
import sys

class Loss:
    def __init__(self, args):
        self.args = args

    def define_loss(self):
        sys.path.append(self.args['main'])

        if self.args['loss'] == 'focal_loss':
            from subroutines.loss_metric import focal_loss
            loss = focal_loss(alpha=self.args['focal_loss_a'],
                              gamma=self.args['focal_loss_g'])

        elif self.args['loss'] == 'WCE':
            from subroutines.loss_metric import weighted_cross_entropy
            loss = weighted_cross_entropy(beta=self.args['WCE_beta'])

        elif self.args['loss'] == 'F1_score_loss':
            from subroutines.loss_metric import F1_score_loss
            loss = F1_score_loss

        elif self.args['loss'] == 'F1_score_loss_dil':
            from subroutines.loss_metric import F1_score_loss_dil
            loss = F1_score_loss_dil

        elif self.args['loss'] == 'Binary_Crossentropy':
            import tensorflow as tf
            loss = tf.keras.losses.BinaryCrossentropy()

        return loss
```

metrics_class.py

```
import sys
import tensorflow.keras

class Metrics:
    def __init__(self, args):
        self.args = args

    def define_Metrics(self):

        sys.path.append(self.args['main'])

        from subroutines.loss_metric import recall
        from subroutines.loss_metric import precision
        from subroutines.loss_metric import precision_dil
        from subroutines.loss_metric import F1_score
        from subroutines.loss_metric import F1_score_dil

        # Prepare metrics
        metrics = [
            tensorflow.keras.metrics.BinaryAccuracy(name='accuracy'),
            recall,
            precision,
            precision_dil,
```

```

        F1_score,
        F1_score_dil
    ]

```

```

    return metrics

```

network_class.py

```

import sys

```

```

class Network:

```

```

    def __init__(self, args, IMAGE_DIMS, N_FILTERS, BS, INIT_LR, opt, loss, metrics):
        self.args = args
        self.IMAGE_DIMS = IMAGE_DIMS
        self.N_FILTERS = N_FILTERS
        self.BS = BS
        self.INIT_LR = INIT_LR
        self.opt = opt
        self.loss = loss
        self.metrics = metrics

```

```

    def add_regularization_function(self, args, model):
        """
        If args['regularization'] is not None, add regularization and then return the
        model
        """

```

```

        if self.args['regularization'] is not None:
            from networks import add_regularization
            model = add_regularization(model, self.args['regularization'])

```

```

        return model

```

```

    def define_network(self):
        sys.path.append(self.args['main'])

```

```

        if self.args['model'] == 'Unet':
            from networks import Unet

```

```

            model = Unet(self.IMAGE_DIMS, n_filters=self.N_FILTERS,
                dropout=self.args['dropout'],
                batchnorm=self.args['batchnorm'],
                regularization=self.args['regularization'],
                kernel_initializer=self.args['init'])

```

```

        else:
            print(f'Unsupported model: {self.args["model"]}. Exiting...')
            sys.exit()

```

```

        model.compile(optimizer=self.opt, loss=self.loss, metrics=[self.metrics])

```

```

        return model

```

optimizer_class.py

```

import sys

```

```

class Optimizer:

```

```

    def __init__(self, args, INIT_LR):
        self.args = args
        self.INIT_LR = INIT_LR

```

```
def define_optimizer(self):

    sys.path.append(self.args['main'])

    if self.args['opt'] == 'Adam':
        from tensorflow.keras.optimizers import Adam
        opt = Adam(self.INIT_LR)

    elif self.args['opt'] == 'SGD':
        from tensorflow.keras.optimizers import SGD
        opt = SGD(self.INIT_LR)

    elif self.args['opt'] == 'RMSprop':
        from tensorflow.keras.optimizers import RMSprop
        opt = RMSprop(self.INIT_LR)

    return opt
```

add_regularization.py

```
import tensorflow.keras
from tensorflow.keras.regularizers import l2
import os
import tempfile
from tensorflow.keras.models import model_from_json

def add_regularization(model, regularization):
    """
    The addition of regularization is based on the implementation shown in the link:
    https://sthalles.github.io/keras-regularizer/
    """

    regularizer = l2(regularization)

    if not isinstance(regularizer, tensorflow.keras.regularizers.Regularizer):
        print("Regularizer must be a subclass of tf.keras.regularizers.Regularizer")
        return model

    for layer in model.layers:
        for attr in ['kernel_regularizer']:
            if hasattr(layer, attr):
                setattr(layer, attr, regularizer)

    # When we change the layers attributes, the change only happens in the model
    # config file
    model_json = model.to_json()

    # Save the weights before reloading the model.
    tmp_weights_path = os.path.join(tempfile.gettempdir(), 'tmp_weights.h5')
    model.save_weights(tmp_weights_path)

    # load the model from the config
    model = model_from_json(model_json)

    # Reload the model weights
    model.load_weights(tmp_weights_path, by_name=True)
    return model
```

Unet.py

```

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.regularizers import l2

def conv2d_block(input_tensor, n_filters, kernel_size=3, batchnorm=True,
regularization=None,
                  kernel_initializer='he_normal'):
    # first layer
    # check whether to add regularization
    if regularization is None:
        kernel_regularizer = None
    else:
        kernel_regularizer = l2(regularization)

    x = Conv2D(
        filters=n_filters,
        kernel_size=(kernel_size, kernel_size),
        kernel_initializer=kernel_initializer,
        padding='same',
        kernel_regularizer=kernel_regularizer
    )(input_tensor)

    # check whether to add batch normalization
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # second layer
    x = Conv2D(
        filters=n_filters,
        kernel_size=(kernel_size, kernel_size),
        kernel_initializer=kernel_initializer,
        padding='same',
        kernel_regularizer=kernel_regularizer
    )(x)

    # check whether to add batch normalization
    if batchnorm:
        x = BatchNormalization()(x)
    x = Activation('relu')(x)

    return x

def Unet(IMAGE_DIMS, n_filters=16, dropout=0.25, batchnorm=True, regularization=None,
kernel_initializer='he_normal'):
    # define the input for the model
    input_img = Input(IMAGE_DIMS, name='img')

    # contracting path
    c1 = conv2d_block(

```

```

        input_img,
        n_filters=n_filters * 1,
        kernel_size=3,
        batchnorm=batchnorm,
        regularization=regularization,
        kernel_initializer=kernel_initializer
    )
    p1 = MaxPooling2D((2, 2))(c1)
    if dropout is not None:
        p1 = Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters=n_filters * 2, kernel_size=3,
batchnorm=batchnorm)
    p2 = MaxPooling2D((2, 2))(c2)
    if dropout is not None:
        p2 = Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters=n_filters * 4, kernel_size=3,
batchnorm=batchnorm)
    p3 = MaxPooling2D((2, 2))(c3)
    if dropout is not None:
        p3 = Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters=n_filters * 8, kernel_size=3,
batchnorm=batchnorm)
    p4 = MaxPooling2D(pool_size=(2, 2))(c4)
    if dropout is not None:
        p4 = Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters=n_filters * 16, kernel_size=3,
batchnorm=batchnorm)

    # expansive path
    u6 = Conv2DTranspose(n_filters * 8, (3, 3), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    if dropout is not None:
        u6 = Dropout(dropout)(u6)

    c6 = conv2d_block(u6, n_filters=n_filters * 8, kernel_size=3,
batchnorm=batchnorm)

    u7 = Conv2DTranspose(n_filters * 4, (3, 3), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    if dropout is not None:
        u7 = Dropout(dropout)(u7)

    c7 = conv2d_block(u7, n_filters=n_filters * 4, kernel_size=3,
batchnorm=batchnorm)

    u8 = Conv2DTranspose(n_filters * 2, (3, 3), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    if dropout is not None:
        u8 = Dropout(dropout)(u8)

    c8 = conv2d_block(u8, n_filters=n_filters * 2, kernel_size=3,
batchnorm=batchnorm)

    u9 = Conv2DTranspose(n_filters * 1, (3, 3), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    if dropout is not None:
        u9 = Dropout(dropout)(u9)

```

```
c9 = conv2d_block(u9, n_filters=n_filters * 1, kernel_size=3,
batchnorm=batchnorm)
```

```
outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
model = Model(inputs=[input_img], outputs=[outputs])
return model
```

epochcheckpoint.py

```
from tensorflow.keras.callbacks import Callback
import os
```

```
class EpochCheckpoint(Callback):
    def __init__(self, output_path_checkpoints, output_path_weights,
save_model_weights, every=5, start_at=0, info='',
        counter='', extension='.h5'):
        # call the parent constructor
        super(Callback, self).__init__()

        # store the base output path for the model, the number of
        # epochs that must pass before the model is serialized to
        # disk and the current epoch value
        self.outputPath_checkpoints = output_path_checkpoints
        self.outputPath_weights = output_path_weights
        self.save_model_weights = save_model_weights
        self.every = every
        self.intEpoch = start_at
        self.info = info
        self.counter = counter
        self.extension = extension

    def on_epoch_end(self, epoch, logs=None):
        # check to see if the model should be serialized to disk
        if logs is None:
            logs = {}
        if (self.intEpoch + 1) % self.every == 0:
            if self.save_model_weights == 'model':
                folder_output = self.outputPath_checkpoints
            else:
                folder_output = self.outputPath_weights

            # define the name of saved model/weights
            if self.info == '':
                p = os.path.sep.join([
                    folder_output,
                    '{}_epoch_{}'.format(self.counter, self.intEpoch + 1,
self.extension)
                ])
            else:
                p = os.path.sep.join([
                    folder_output,
                    '{}_{}_epoch_{}'.format(self.info, self.counter, self.intEpoch
+ 1, self.extension)
                ])

            # check whether to save the whole model or only the weight
            if self.save_model_weights == 'model':
                self.model.save(p, overwrite=True)
```

```

        elif self.save_model_weights == 'weights':
            self.model.save_weights(p)

# increment the internal epoch counter
self.intEpoch += 1

```

trainingmonitor.py

```

from tensorflow.keras.callbacks import BaseLogger
import matplotlib.pyplot as plt
import numpy as np
import json
import os

class TrainingMonitor(BaseLogger):
    def __init__(self, fig_path, json_path=None, start_at=0, metric='accuracy'):
        # store the output path for the figure, the path to the JSON serialized file,
        # and the starting epoch
        super(TrainingMonitor, self).__init__()
        self.figPath = fig_path
        self.jsonPath = json_path
        self.startAt = start_at
        self.metric = metric
        self.history = {}

    def on_train_begin(self, logs={}):
        # initialize the history dictionary
        self.history = {}

        # if the JSON history path exists, load the training history
        if self.jsonPath is not None:
            if os.path.exists(self.jsonPath):
                self.history = json.loads(open(self.jsonPath).read())

            # check to see if a starting epoch was supplied
            if self.startAt > 0:
                # loop over the entries in the history log and trim any entries
                # that are past the starting epoch
                for k in self.history.keys():
                    self.history[k] = self.history[k][:self.startAt]

    def on_epoch_end(self, epoch, logs={}):
        # loop over the logs and update the loss, accuracy, etc. for the entire
        # training process
        for (k, v) in logs.items():
            l = self.history.get(k, [])
            l.append(float(v))
            self.history[k] = l

        # check to see if the training history should be serialized to file
        if self.jsonPath is not None:
            with open(self.jsonPath, 'w') as f:
                f.write(json.dumps(self.history))

        # ensure at least two epochs have passed before plotting (epoch starts at
        # zero)
        if len(self.history['loss']) > 1:
            # plot the training loss and accuracy
            N = np.arange(0, len(self.history['loss']))
            plt.style.use('ggplot')

```

```

plt.figure()
plt.plot(N, self.history['loss'], 'k--', label='Loss - Train')
plt.plot(N, self.history['val_loss'], 'r--', label='Loss - Val')

plt.plot(N, self.history[self.metric], 'k-', label=self.metric + ' -
Train')
plt.plot(N, self.history['val_' + self.metric], 'r-', label=self.metric +
' - Val')

plt.title('[Epoch {}].format(len(self.history['loss']))))

plt.xlabel('Epoch #')
plt.ylabel('Loss/{}'.format(self.metric))
plt.legend()

# save the figure
plt.savefig(self.figPath)
plt.close()

```

hdf5datasetgenerator.py

```

import numpy as np
import h5py

class HDF5DatasetGeneratorMask:
    def __init__(self, db_path, batch_size, preprocessors=None, shuffle=False,
                 aug=None, binarize=True, classes=2, threshold=0.5):
        # store the batch size, preprocessors, and data augmentor,
        # whether or not the labels should be binarized, along with
        # the total number of classes
        self.batchSize = batch_size
        self.preprocessors = preprocessors
        self.aug = aug
        self.binarize = binarize
        self.classes = classes
        self.shuffle = shuffle

        # open the HDF5 database for reading and determine the total
        # number of entries in the database
        self.db = h5py.File(db_path, 'r+')
        self.numImages = self.db["labels"].shape[0]
        # threshold to binarize mask
        self.threshold = threshold

    def generator(self, passes=np.inf):
        # initialize the epoch count
        epochs = 0

        # keep looping infinitely -- the model will stop once we have
        # reach the desired number of epochs
        while epochs < passes:
            # loop over the HDF5 dataset
            for i in np.arange(0, self.numImages, self.batchSize):
                # extract the images and labels from the HDF dataset
                images = self.db["images"][i: i + self.batchSize]
                labels = self.db["labels"][i: i + self.batchSize]

                # check to see if our preprocessors are not None
                if self.preprocessors is not None:
                    # initialize the list of processed images
                    proc_images = []

```

```

        # loop over the images
        for image in images:
            # loop over the preprocessors and apply each
            # to the image
            for p in self.preprocessors:
                image = p.preprocess(image)

            # update the list of processed images
            proc_images.append(image)

        # update the images array to be the processed
        # images
        images = np.array(proc_images)

    # if the data augmenator exists, apply it
    if self.aug is not None:

        # perform the same transformation to image and mask
        seed = 2018

        image_generator = self.aug.flow(images, seed=seed,
batch_size=self.batchSize, shuffle=self.shuffle)
        mask_generator = self.aug.flow(labels, seed=seed,
batch_size=self.batchSize, shuffle=self.shuffle)

        train_generator = zip(image_generator, mask_generator)
        (images, labels) = next(train_generator)

        # process labels so that they have only 0 or 1
        if self.binarize:
            for ii in range(0, len(labels)):
                labels[ii] = np.where(labels[ii]>self.threshold, 1., 0.)

        # yield a tuple of images and labels
        yield (images, labels)

    # increment the total number of epochs
    epochs += 1

def close(self):
    # close the database
    self.db.close()

```

hdf5datasetwriter_mask.py

```

import h5py
import os

class HDF5DatasetWriterMask:
    def __init__(self, dims, output_path, data_key="images", buf_size=1000):
        # check to see if the output path exists, and if so, raise
        # an exception
        if os.path.exists(output_path):
            raise ValueError("The supplied 'outputPath' already "
                "exists and cannot be overwritten. Manually delete "
                "the file before continuing.", output_path)

        # open the HDF5 database for writing and create two datasets:
        # one to store the images/features and another to store the

```

```

# class labels
self.db = h5py.File(output_path, "w")
self.data = self.db.create_dataset(data_key, dims, dtype="float")
self.labels = self.db.create_dataset("labels", dims[:-1] + (1,),
dtype="float")

# store the buffer size, then initialize the buffer itself
# along with the index into the datasets
self.bufSize = buf_size
self.buffer = {"data": [], "labels": []}
self.idx = 0

def add(self, rows, labels):
    # add the rows and labels to the buffer
    self.buffer["data"].extend(rows)
    self.buffer["labels"].extend(labels)

    # check to see if the buffer needs to be flushed to disk
    if len(self.buffer["data"]) >= self.bufSize:
        self.flush()

def flush(self):
    # write the buffers to disk then reset the buffer
    i = self.idx + len(self.buffer["data"])
    self.data[self.idx:i] = self.buffer["data"]
    self.labels[self.idx:i] = self.buffer["labels"]
    self.idx = i
    self.buffer = {"data": [], "labels": []}

def close(self):
    # check to see if there are any other entries in the buffer
    # that need to be flushed to disk
    if len(self.buffer["data"]) > 0:
        self.flush()

    # close the dataset
    self.db.close()

```

loss_metric.py

```

import tensorflow.keras.backend as K
import tensorflow as tf
import numpy as np
import cv2

def dilation2d(img4D):
    """
    Function to dilate mask
    It is used to dilate the y_true when calculating precision in order to allow for
    some tolerance

    Background pixels predicted as cracks (False Positives) were considered as True
    Positives
    if they were a few pixels apart from the annotated cracks
    """

    # The greater the kernel size, the greater extent of the dilation applied
    kernel_size = 5

    with tf.compat.v1.variable_scope('dilation2d'):

```

```

        kernel = tf.zeros((kernel_size, kernel_size, 1))
        output4D = tf.nn.dilation2d(img4D, filters=kernel, strides=(1, 1, 1, 1),
rates=(1, 1, 1, 1), padding='SAME')

```

```

    return output4D

```

```

# Weighted Cross-Entropy (WCE) Loss
# It is based on the implementation found in the link below:
# https://jeune-research.tistory.com/entry/Loss-Functions-for-Image-Segmentation-
Distribution-Based-Losses

```

```

def weighted_cross_entropy(beta):
    def convert_to_logits(y_pred):
        y_pred = tf.clip_by_value(y_pred, tf.keras.backend.epsilon(), 1 -
tf.keras.backend.epsilon())

```

```

        return tf.math.log(y_pred / (1 - y_pred))

```

```

    def loss(y_true, y_pred):
        y_pred = convert_to_logits(y_pred)
        loss_val = tf.nn.weighted_cross_entropy_with_logits(logits=y_pred,
labels=y_true, pos_weight=beta)

```

```

        return tf.reduce_mean(loss_val)

```

```

    return loss

```

```

# Focal Loss

```

```

# It is based on the implementation found in the link below:

```

```

# https://github.com/umbertogriffo/focal-loss-keras

```

```

def focal_loss(gamma=2., alpha=.25):

```

```

    """

```

```

    Binary form of focal loss.

```

```

        FL(p_t) = -alpha * (1 - p_t)**gamma * log(p_t)

```

```

        where p = sigmoid(x), p_t = p or 1 - p depending on if the label is 1 or 0,
respectively.

```

```

    References:

```

```

        https://arxiv.org/pdf/1708.02002.pdf

```

```

    Usage:

```

```

        model.compile(loss=[binary_focal_loss(alpha=.25, gamma=2)],
metrics=["accuracy"], optimizer=adam)

```

```

    """

```

```

def binary_focal_loss_fixed(y_true, y_pred):

```

```

    """

```

```

        :param y_true: A tensor of the same shape as `y_pred`

```

```

        :param y_pred: A tensor resulting from a sigmoid

```

```

        :return: Output tensor.

```

```

    """

```

```

    pt_1 = tf.where(tf.equal(y_true, 1), y_pred, tf.ones_like(y_pred))

```

```

    pt_0 = tf.where(tf.equal(y_true, 0), y_pred, tf.zeros_like(y_pred))

```

```

    epsilon = K.epsilon()

```

```

    # Clip to prevent NaN's and Inf's

```

```

    pt_1 = K.clip(pt_1, epsilon, 1. - epsilon)

```

```

    pt_0 = K.clip(pt_0, epsilon, 1. - epsilon)

```

```

    return -K.sum(alpha * K.pow(1. - pt_1, gamma) * K.log(pt_1)) \
        - K.sum((1 - alpha) * K.pow(pt_0, gamma) * K.log(1. - pt_0))

```

```

return binary_focal_loss_fixed

# F1-score Loss
def F1_score_loss(y_true, y_pred):
    smooth = 1.
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = y_true_f * y_pred_f
    score = (2. * K.sum(intersection) + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f)
+ smooth)
    return 1. - score

# F1-score Loss with dilated y_true mask
def F1_score_loss_dil(y_true, y_pred):
    smooth = 1.
    # Dilate y_true
    y_true_dil = dilation2d(y_true)
    y_true_dil = K.flatten(y_true_dil)

    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = y_true_dil * y_pred_f
    score = (2. * K.sum(intersection) + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f)
+ smooth)
    return 1. - score

# Recall Metric
def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall_val = true_positives / (possible_positives + K.epsilon())
    return recall_val

# Precision Metric
def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision_val = true_positives / (predicted_positives + K.epsilon())
    return precision_val

# Precision Metric with dilated y_true mask
def precision_dil(y_true, y_pred):
    # Dilate y_true
    y_true = dilation2d(y_true)

    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision_val = true_positives / (predicted_positives + K.epsilon())
    return precision_val

# F1-score Metric
def F1_score(y_true, y_pred):
    precision_val = precision(y_true, y_pred)
    recall_val = recall(y_true, y_pred)

```

```

    return 2 * ((precision_val * recall_val) / (precision_val + recall_val +
K.epsilon()))

# F1-score Metric with dilated y_true mask
def F1_score_dil(y_true, y_pred):
    precision_val = precision_dil(y_true, y_pred)
    recall_val = recall(y_true, y_pred)
    return 2 * ((precision_val * recall_val) / (precision_val + recall_val +
K.epsilon()))

# Define metrics to be used for evaluation of the trained model using NumPy instead
of tensorflow tensors
def dilate_mask(mask, threshold=0.5, iterations=1):
    """
    receives mask and returns dilated mask
    """

    kernel = np.ones((5, 5), np.uint8)
    mask_dilated = mask.copy()
    mask_dilated = cv2.dilate(mask_dilated, kernel, iterations=iterations)
    # Binarize mask after dilation
    mask_dilated = np.where(mask_dilated > threshold, 1., 0.)

    return mask_dilated

def recall_np(y_true, y_pred, threshold=0.5):
    eps = 1e-07
    y_true_f = y_true.flatten().astype('float32')
    half = (np.ones(y_true_f.shape) * threshold).astype('float32')
    y_pred_f = np.greater(y_pred.flatten(), half).astype('float32')
    true_positives = (y_true_f * y_pred_f).sum()
    possible_positives = y_true_f.sum()
    recall_val = (true_positives + eps) / (possible_positives + eps)
    return recall_val

def precision_np(y_true, y_pred, threshold=0.5):
    eps = 1e-07
    y_true_f = y_true.flatten().astype('float32')
    half = (np.ones(y_true_f.shape) * threshold).astype('float32')
    y_pred_f = np.greater(y_pred.flatten(), half).astype('float32')
    true_positives = (y_true_f * y_pred_f).sum()
    predicted_positives = y_pred_f.sum()
    precision_val = (true_positives + eps) / (predicted_positives + eps)

    return precision_val

def F1_score_np(recall_val, precision_val):
    eps = 1e-07
    return 2 * ((precision_val * recall_val) / (precision_val + recall_val + eps))

```

visualize_model.py

```

from contextlib import redirect_stdout

def model_summary(model, folder_summary):

```

```

"""
save the summary to txt
"""

with open(folder_summary, 'w') as f:
    with redirect_stdout(f):
        model.summary()

def visualize_model(model, folder_plot, folder_summary):
    """
    from contextlib import redirect_stdout
    save the summary to txt,
    plot the model to png
    """

    from tensorflow.keras.utils import plot_model

    plot_model(model, to_file=folder_plot, show_shapes=True)
    model_summary(model, folder_summary)

def visualize_model_tf(model, folder_plot, folder_summary):
    """
    for models built with tf.keras use this function to visualize them
    from contextlib import redirect_stdout
    save the summary to txt,
    plot the model to png
    """

    import tensorflow as tf

    tf.keras.utils.plot_model(model, folder_plot, show_shapes=True)
    model_summary(model, folder_summary)

```

visualize_predictions.py

```

import sys
import os
import numpy as np
import h5py
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

def visualize_predictions(args, predictions, threshold=0.5):
    """
    threshold: use this value to binarize the ground truth mask
    """
    sys.path.append(args['main'])
    from subroutines.loss_metric import dilate_mask
    from subroutines.loss_metric import recall_np
    from subroutines.loss_metric import precision_np
    from subroutines.loss_metric import F1_score_np
    # Load database with images and masks
    db = h5py.File(args['EVAL_HDF5'], 'r+')
    num_images = db['images'].shape[0]
    # Show the crack as grey
    color = 0.5 # 0.5:grey, 1: white
    # Size of plot
    plt_size = [3, 3]

```

```

# Loop over the images and produce a plot with original image, ground truth and
prediction
for i in range(0, num_images):

    # Filename to store plot for each image
    plt_file = os.path.join(args['predictions_subfolder'], f'prediction_{i}.png')

    # Ground truth
    gt = (db['labels'][i].squeeze()) * color

    # Define im array
    im = (np.zeros(db['images'][i].shape)).astype('uint8')
    im[:, :, 0] = (db['images'][i][:, :, 2] * 255).astype('uint8')
    im[:, :, 1] = (db['images'][i][:, :, 1] * 255).astype('uint8')
    im[:, :, 2] = (db['images'][i][:, :, 0] * 255).astype('uint8')

    # Check whether to dilate ground truth mask for the calculation of Precision
metric
    if args['predictions_dilate']:
        y_true_precision = dilate_mask(db['labels'][i])
    else:
        y_true_precision = db['labels'][i]

    # Calculate metrics for each image
    recall = recall_np(db['labels'][i], predictions[i])
    precision = precision_np(y_true_precision, predictions[i])
    f1_score = F1_score_np(recall, precision)
    # Format metrics
    recall = int(round(recall, 2) * 100)
    precision = int(round(precision, 2) * 100)
    f1_score = int(round(f1_score, 2) * 100)
    plt_subtitle = 'F1:{0}% / RE:{1}%\nPR:{2}%'.format(f1_score, recall,
precision)

    # Create figure with 3 subplots
    fig = plt.figure()
    fig.set_size_inches(plt_size)
    ax1 = plt.subplot2grid((1, 1), (0, 0))
    divider = make_axes_locatable(ax1)
    ax2 = divider.append_axes('bottom', size="100%", pad=0.1)
    ax3 = divider.append_axes('bottom', size="100%", pad=0.4)

    # Original image
    ax1.imshow(im)
    # Ground truth
    ax2.imshow(gt, vmin=0, vmax=1, cmap='gray')
    # Prediction
    prediction = ((predictions[i].squeeze() > threshold) * 1) * color
    ax3.imshow(prediction, vmin=0, vmax=1, cmap='Reds')

    # Set title for prediction
    ax3.set_title(plt_subtitle, fontsize=7)

    # Remove axes
    ax1.axis('off')
    ax2.axis('off')
    ax3.axis('off')
    plt.tight_layout()
    plt.savefig(plt_file, bbox_inches='tight', dpi=100, pad_inches=0.05)
    plt.close()

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів
Дипломна кваліфікаційна робота бакалавра

Програмна реалізація методів розпізнавання тріщин будівель

Software implementation of the methods for crack detection in buildings

Виконав
ст. гр. КНТ-137

В. Р. Судаков

Керівник
к.ф.м.н., доцент

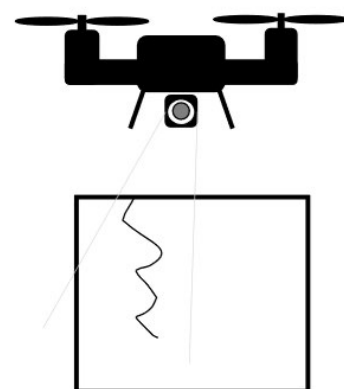
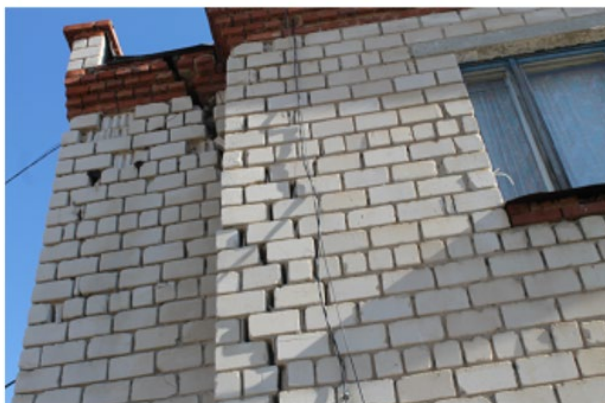
Е. В. Терещенко

2021

1

Рисунок Б.1 – Слайд презентації № 1

Актуальність дипломної роботи



2

Рисунок Б.2 – Слайд презентації № 2

Мета та задачі дипломної роботи

Мета роботи – створення програмного застосунку з інтуїтивно зрозумілою навігацією для автоматизації виявлення тріщин на зображеннях будівель методами глибинного навчання.

Об’єкт дослідження – процес виявлення тріщин на будівлях.

Предмет дослідження – реалізація існуючих методів виявлення тріщин на зображеннях будівель.

3

Рисунок Б.3 – Слайд презентації № 3

Аналіз предметної області



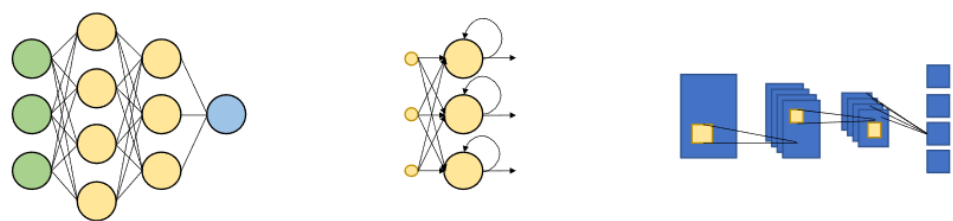
Розповсюджені задачі для нейронних мереж

4

Рисунок Б.4 – Слайд презентації № 4

Згорткові мережі

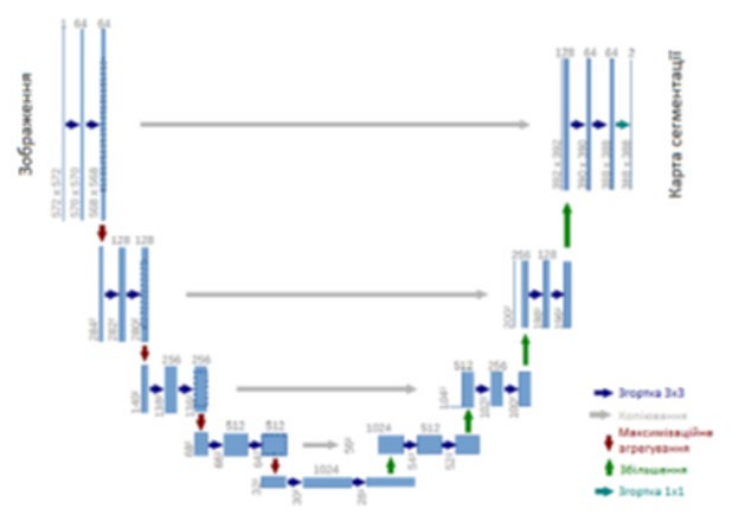
Штучні мережі Artificial (ANN)	Рекурентні мережі Recurrent (RNN)	Згорткові мережі Convolution (CNN)
Розмір зображення значно збільшує кількість параметрів для навчання	Параметри поширені між різними етапами, що зменшує їх кількість	Концепт спільного використання параметрів
Не враховують просторові властивості зображення	Доречніше для обробки природної мови	Використовують сильну просторово локальну кореляцію
Не враховують закономірність у послідовних даних	Здатні виявляти закономірність у послідовних даних	Досягають кращого узагальнення на задачах бачення



5

Рисунок Б.5 – Слайд презентації № 5

U-Net



Архітектура U-Net

6

Рисунок Б.6 – Слайд презентації № 6

Вимоги до продукту



7

Рисунок Б.7 – Слайд презентації № 7

Вибір мови програмування, середовища розробки

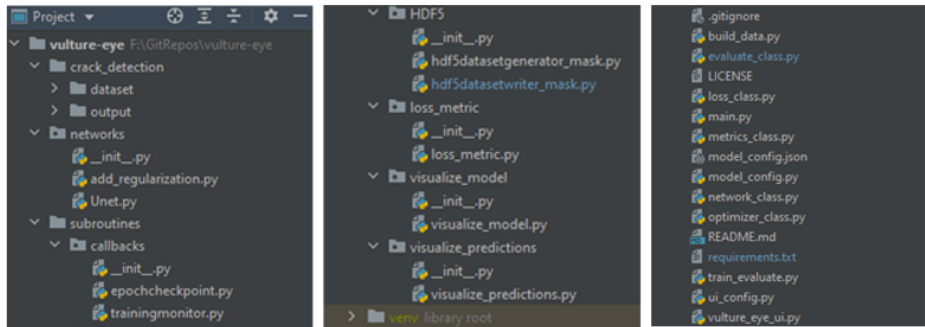
	Python	R	C#	C++	Java
Бібліотеки для навчання НМ	багато	середньо	мало	мало	середньо
Бібліотеки для аналізу даних	багато	багато	мало	мало	мало
Документація	багато, якісна	середньо, не зручна	багато, якісна	багато, складна	середньо, складна
Реалізація програм з GUI	легко, просунутий	обмежений	легко, просунутий	складно, просунутий	складно, просунутий

	PyCharm	Spyder	Idle	Sublime	Wing
Ціна	безкошт.	безкошт.	безкошт.	\$80	\$100
Зневаджувач	так	так	так	не зручний	так
Розширення	так	мало	ні	так	так
VCS	git	ні	ні	плагіни	так

8

Рисунок Б.8 – Слайд презентації № 8

Схема структури програми

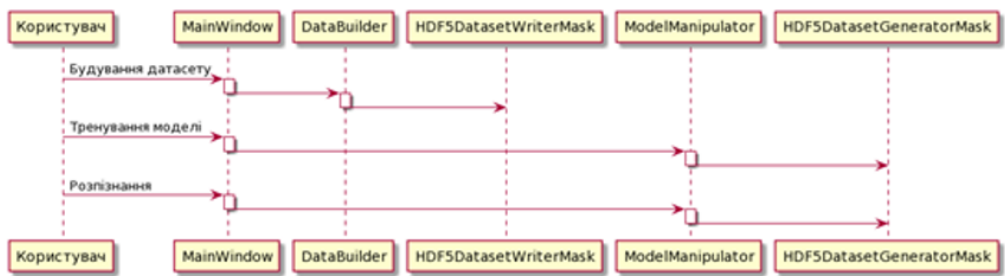


Загальна структура розробленої програми

9

Рисунок Б.9 – Слайд презентації № 9

Схема функціонування розробленої програми

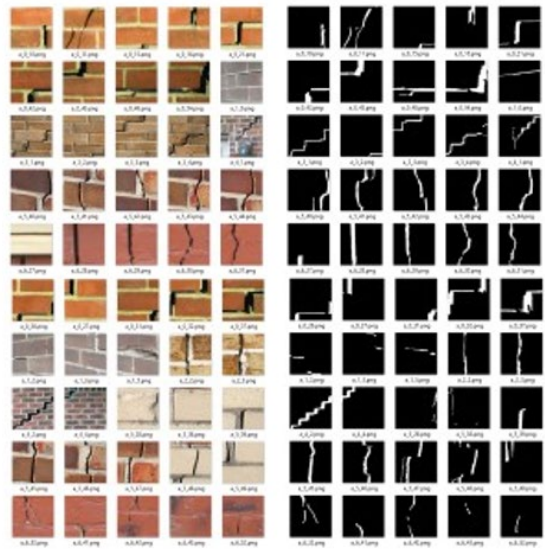


Діаграма послідовностей

10

Рисунок Б.10 – Слайд презентації № 10

Формати даних



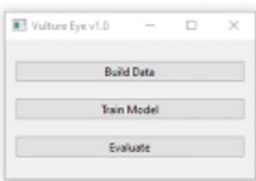
- HDF5
- JSON
- TXT

Зображення тріщин та їх маски

11

Рисунок Б.11 – Слайд презентації № 11

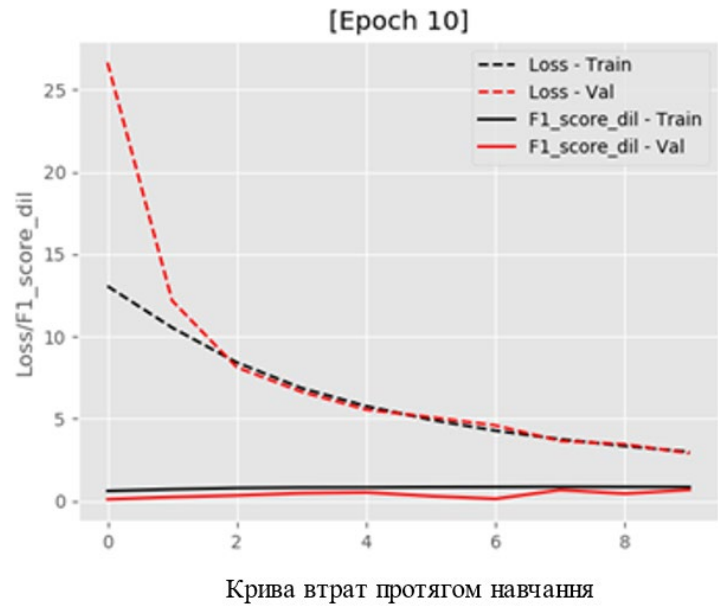
Використання програми



Name	Date modified	Type	Size
git	5/25/2023 1:56 PM	File folder	
idea	6/2/2023 7:40 AM	File folder	
__pycache__	5/25/2023 1:54 PM	File folder	
mask_annotation	5/25/2023 1:50 PM	File folder	
networks	5/25/2023 1:50 PM	File folder	
submodels	5/25/2023 1:46 AM	File folder	
utils	5/25/2023 1:50 PM	File folder	
__pycache__	5/25/2023 1:56 PM	File folder	2 KB
build_data.py	5/25/2023 1:53 PM	Python file	4 KB
evaluate_model.py	5/25/2023 1:56 PM	Python file	2 KB
LICENSE	5/25/2023 1:56 PM	File	2 KB
loss_function.py	5/25/2023 1:56 PM	Python file	1 KB
main.py	5/25/2023 1:52 AM	Python file	1 KB
metrics_classification.py	5/25/2023 1:57 AM	Python file	1 KB
model_config.py	5/25/2023 1:56 PM	Python file	1 KB
model_config.py	5/25/2023 1:56 PM	Python file	1 KB
network_utils.py	5/25/2023 1:56 PM	Python file	2 KB
optimizer_utils.py	6/2/2023 1:54 AM	Python file	1 KB
README.md	5/25/2023 1:53 PM	Markdown Source	1 KB
requirements.txt	5/25/2023 1:56 PM	Text Document	1 KB
train_data.py	5/25/2023 1:52 PM	Python file	8 KB
utils.py	5/25/2023 1:52 PM	Python file	1 KB
vulture_eye.py	5/25/2023 1:43 PM	Python file	2 KB

Model: "model"				
layer (type)	Output Shape	Param #	Connected to	
InputLayer	(None, 224, 224, 3)	0		
conv0d (Conv2D)	(None, 224, 224, 64) 1792		InputLayer[0]	
batch_normalization (BatchNorma	(None, 224, 224, 64) 256		conv0d[0]	
activation_1 (Activation)	(None, 224, 224, 64) 0		batch_normalization[0]	
conv0d_1 (Conv2D)	(None, 224, 224, 64) 16384		activation_1[0]	
batch_normalization_1 (BatchNorm	(None, 224, 224, 64) 256		conv0d_1[0]	
activation_2 (Activation)	(None, 224, 224, 64) 0		batch_normalization_1[0]	
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64) 0		activation_2[0]	
conv0d_2 (Conv2D)	(None, 112, 112, 128) 73856		max_pooling2d[0]	
batch_normalization_2 (BatchNorm	(None, 112, 112, 128) 512		conv0d_2[0]	
activation_3 (Activation)	(None, 112, 112, 128) 0		batch_normalization_2[0]	
conv0d_3 (Conv2D)	(None, 112, 112, 128) 16384		activation_3[0]	
conv0d_4 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_3[0]	
batch_normalization_3 (BatchNorm	(None, 112, 112, 128) 512		conv0d_4[0]	
activation_4 (Activation)	(None, 112, 112, 128) 0		batch_normalization_3[0]	
conv0d_5 (Conv2D)	(None, 112, 112, 128) 16384		activation_4[0]	
conv0d_6 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_5[0]	
batch_normalization_4 (BatchNorm	(None, 112, 112, 128) 512		conv0d_6[0]	
activation_5 (Activation)	(None, 112, 112, 128) 0		batch_normalization_4[0]	
conv0d_7 (Conv2D)	(None, 112, 112, 128) 16384		activation_5[0]	
conv0d_8 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_7[0]	
batch_normalization_5 (BatchNorm	(None, 112, 112, 128) 512		conv0d_8[0]	
activation_6 (Activation)	(None, 112, 112, 128) 0		batch_normalization_5[0]	
conv0d_9 (Conv2D)	(None, 112, 112, 128) 16384		activation_6[0]	
conv0d_10 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_9[0]	
batch_normalization_6 (BatchNorm	(None, 112, 112, 128) 512		conv0d_10[0]	
activation_7 (Activation)	(None, 112, 112, 128) 0		batch_normalization_6[0]	
conv0d_11 (Conv2D)	(None, 112, 112, 128) 16384		activation_7[0]	
conv0d_12 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_11[0]	
batch_normalization_7 (BatchNorm	(None, 112, 112, 128) 512		conv0d_12[0]	
activation_8 (Activation)	(None, 112, 112, 128) 0		batch_normalization_7[0]	
conv0d_13 (Conv2D)	(None, 112, 112, 128) 16384		activation_8[0]	
conv0d_14 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_13[0]	
batch_normalization_8 (BatchNorm	(None, 112, 112, 128) 512		conv0d_14[0]	
activation_9 (Activation)	(None, 112, 112, 128) 0		batch_normalization_8[0]	
conv0d_15 (Conv2D)	(None, 112, 112, 128) 16384		activation_9[0]	
conv0d_16 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_15[0]	
batch_normalization_9 (BatchNorm	(None, 112, 112, 128) 512		conv0d_16[0]	
activation_10 (Activation)	(None, 112, 112, 128) 0		batch_normalization_9[0]	
conv0d_17 (Conv2D)	(None, 112, 112, 128) 16384		activation_10[0]	
conv0d_18 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_17[0]	
batch_normalization_10 (BatchNorm	(None, 112, 112, 128) 512		conv0d_18[0]	
activation_11 (Activation)	(None, 112, 112, 128) 0		batch_normalization_10[0]	
conv0d_19 (Conv2D)	(None, 112, 112, 128) 16384		activation_11[0]	
conv0d_20 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_19[0]	
batch_normalization_11 (BatchNorm	(None, 112, 112, 128) 512		conv0d_20[0]	
activation_12 (Activation)	(None, 112, 112, 128) 0		batch_normalization_11[0]	
conv0d_21 (Conv2D)	(None, 112, 112, 128) 16384		activation_12[0]	
conv0d_22 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_21[0]	
batch_normalization_12 (BatchNorm	(None, 112, 112, 128) 512		conv0d_22[0]	
activation_13 (Activation)	(None, 112, 112, 128) 0		batch_normalization_12[0]	
conv0d_23 (Conv2D)	(None, 112, 112, 128) 16384		activation_13[0]	
conv0d_24 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_23[0]	
batch_normalization_13 (BatchNorm	(None, 112, 112, 128) 512		conv0d_24[0]	
activation_14 (Activation)	(None, 112, 112, 128) 0		batch_normalization_13[0]	
conv0d_25 (Conv2D)	(None, 112, 112, 128) 16384		activation_14[0]	
conv0d_26 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_25[0]	
batch_normalization_14 (BatchNorm	(None, 112, 112, 128) 512		conv0d_26[0]	
activation_15 (Activation)	(None, 112, 112, 128) 0		batch_normalization_14[0]	
conv0d_27 (Conv2D)	(None, 112, 112, 128) 16384		activation_15[0]	
conv0d_28 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_27[0]	
batch_normalization_15 (BatchNorm	(None, 112, 112, 128) 512		conv0d_28[0]	
activation_16 (Activation)	(None, 112, 112, 128) 0		batch_normalization_15[0]	
conv0d_29 (Conv2D)	(None, 112, 112, 128) 16384		activation_16[0]	
conv0d_30 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_29[0]	
batch_normalization_16 (BatchNorm	(None, 112, 112, 128) 512		conv0d_30[0]	
activation_17 (Activation)	(None, 112, 112, 128) 0		batch_normalization_16[0]	
conv0d_31 (Conv2D)	(None, 112, 112, 128) 16384		activation_17[0]	
conv0d_32 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_31[0]	
batch_normalization_17 (BatchNorm	(None, 112, 112, 128) 512		conv0d_32[0]	
activation_18 (Activation)	(None, 112, 112, 128) 0		batch_normalization_17[0]	
conv0d_33 (Conv2D)	(None, 112, 112, 128) 16384		activation_18[0]	
conv0d_34 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_33[0]	
batch_normalization_18 (BatchNorm	(None, 112, 112, 128) 512		conv0d_34[0]	
activation_19 (Activation)	(None, 112, 112, 128) 0		batch_normalization_18[0]	
conv0d_35 (Conv2D)	(None, 112, 112, 128) 16384		activation_19[0]	
conv0d_36 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_35[0]	
batch_normalization_19 (BatchNorm	(None, 112, 112, 128) 512		conv0d_36[0]	
activation_20 (Activation)	(None, 112, 112, 128) 0		batch_normalization_19[0]	
conv0d_37 (Conv2D)	(None, 112, 112, 128) 16384		activation_20[0]	
conv0d_38 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_37[0]	
batch_normalization_20 (BatchNorm	(None, 112, 112, 128) 512		conv0d_38[0]	
activation_21 (Activation)	(None, 112, 112, 128) 0		batch_normalization_20[0]	
conv0d_39 (Conv2D)	(None, 112, 112, 128) 16384		activation_21[0]	
conv0d_40 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_39[0]	
batch_normalization_21 (BatchNorm	(None, 112, 112, 128) 512		conv0d_40[0]	
activation_22 (Activation)	(None, 112, 112, 128) 0		batch_normalization_21[0]	
conv0d_41 (Conv2D)	(None, 112, 112, 128) 16384		activation_22[0]	
conv0d_42 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_41[0]	
batch_normalization_22 (BatchNorm	(None, 112, 112, 128) 512		conv0d_42[0]	
activation_23 (Activation)	(None, 112, 112, 128) 0		batch_normalization_22[0]	
conv0d_43 (Conv2D)	(None, 112, 112, 128) 16384		activation_23[0]	
conv0d_44 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_43[0]	
batch_normalization_23 (BatchNorm	(None, 112, 112, 128) 512		conv0d_44[0]	
activation_24 (Activation)	(None, 112, 112, 128) 0		batch_normalization_23[0]	
conv0d_45 (Conv2D)	(None, 112, 112, 128) 16384		activation_24[0]	
conv0d_46 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_45[0]	
batch_normalization_24 (BatchNorm	(None, 112, 112, 128) 512		conv0d_46[0]	
activation_25 (Activation)	(None, 112, 112, 128) 0		batch_normalization_24[0]	
conv0d_47 (Conv2D)	(None, 112, 112, 128) 16384		activation_25[0]	
conv0d_48 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_47[0]	
batch_normalization_25 (BatchNorm	(None, 112, 112, 128) 512		conv0d_48[0]	
activation_26 (Activation)	(None, 112, 112, 128) 0		batch_normalization_25[0]	
conv0d_49 (Conv2D)	(None, 112, 112, 128) 16384		activation_26[0]	
conv0d_50 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_49[0]	
batch_normalization_26 (BatchNorm	(None, 112, 112, 128) 512		conv0d_50[0]	
activation_27 (Activation)	(None, 112, 112, 128) 0		batch_normalization_26[0]	
conv0d_51 (Conv2D)	(None, 112, 112, 128) 16384		activation_27[0]	
conv0d_52 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_51[0]	
batch_normalization_27 (BatchNorm	(None, 112, 112, 128) 512		conv0d_52[0]	
activation_28 (Activation)	(None, 112, 112, 128) 0		batch_normalization_27[0]	
conv0d_53 (Conv2D)	(None, 112, 112, 128) 16384		activation_28[0]	
conv0d_54 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_53[0]	
batch_normalization_28 (BatchNorm	(None, 112, 112, 128) 512		conv0d_54[0]	
activation_29 (Activation)	(None, 112, 112, 128) 0		batch_normalization_28[0]	
conv0d_55 (Conv2D)	(None, 112, 112, 128) 16384		activation_29[0]	
conv0d_56 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_55[0]	
batch_normalization_29 (BatchNorm	(None, 112, 112, 128) 512		conv0d_56[0]	
activation_30 (Activation)	(None, 112, 112, 128) 0		batch_normalization_29[0]	
conv0d_57 (Conv2D)	(None, 112, 112, 128) 16384		activation_30[0]	
conv0d_58 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_57[0]	
batch_normalization_30 (BatchNorm	(None, 112, 112, 128) 512		conv0d_58[0]	
activation_31 (Activation)	(None, 112, 112, 128) 0		batch_normalization_30[0]	
conv0d_59 (Conv2D)	(None, 112, 112, 128) 16384		activation_31[0]	
conv0d_60 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_59[0]	
batch_normalization_31 (BatchNorm	(None, 112, 112, 128) 512		conv0d_60[0]	
activation_32 (Activation)	(None, 112, 112, 128) 0		batch_normalization_31[0]	
conv0d_61 (Conv2D)	(None, 112, 112, 128) 16384		activation_32[0]	
conv0d_62 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_61[0]	
batch_normalization_32 (BatchNorm	(None, 112, 112, 128) 512		conv0d_62[0]	
activation_33 (Activation)	(None, 112, 112, 128) 0		batch_normalization_32[0]	
conv0d_63 (Conv2D)	(None, 112, 112, 128) 16384		activation_33[0]	
conv0d_64 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_63[0]	
batch_normalization_33 (BatchNorm	(None, 112, 112, 128) 512		conv0d_64[0]	
activation_34 (Activation)	(None, 112, 112, 128) 0		batch_normalization_33[0]	
conv0d_65 (Conv2D)	(None, 112, 112, 128) 16384		activation_34[0]	
conv0d_66 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_65[0]	
batch_normalization_34 (BatchNorm	(None, 112, 112, 128) 512		conv0d_66[0]	
activation_35 (Activation)	(None, 112, 112, 128) 0		batch_normalization_34[0]	
conv0d_67 (Conv2D)	(None, 112, 112, 128) 16384		activation_35[0]	
conv0d_68 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_67[0]	
batch_normalization_35 (BatchNorm	(None, 112, 112, 128) 512		conv0d_68[0]	
activation_36 (Activation)	(None, 112, 112, 128) 0		batch_normalization_35[0]	
conv0d_69 (Conv2D)	(None, 112, 112, 128) 16384		activation_36[0]	
conv0d_70 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_69[0]	
batch_normalization_36 (BatchNorm	(None, 112, 112, 128) 512		conv0d_70[0]	
activation_37 (Activation)	(None, 112, 112, 128) 0		batch_normalization_36[0]	
conv0d_71 (Conv2D)	(None, 112, 112, 128) 16384		activation_37[0]	
conv0d_72 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_71[0]	
batch_normalization_37 (BatchNorm	(None, 112, 112, 128) 512		conv0d_72[0]	
activation_38 (Activation)	(None, 112, 112, 128) 0		batch_normalization_37[0]	
conv0d_73 (Conv2D)	(None, 112, 112, 128) 16384		activation_38[0]	
conv0d_74 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_73[0]	
batch_normalization_38 (BatchNorm	(None, 112, 112, 128) 512		conv0d_74[0]	
activation_39 (Activation)	(None, 112, 112, 128) 0		batch_normalization_38[0]	
conv0d_75 (Conv2D)	(None, 112, 112, 128) 16384		activation_39[0]	
conv0d_76 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_75[0]	
batch_normalization_39 (BatchNorm	(None, 112, 112, 128) 512		conv0d_76[0]	
activation_40 (Activation)	(None, 112, 112, 128) 0		batch_normalization_39[0]	
conv0d_77 (Conv2D)	(None, 112, 112, 128) 16384		activation_40[0]	
conv0d_78 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_77[0]	
batch_normalization_40 (BatchNorm	(None, 112, 112, 128) 512		conv0d_78[0]	
activation_41 (Activation)	(None, 112, 112, 128) 0		batch_normalization_40[0]	
conv0d_79 (Conv2D)	(None, 112, 112, 128) 16384		activation_41[0]	
conv0d_80 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_79[0]	
batch_normalization_41 (BatchNorm	(None, 112, 112, 128) 512		conv0d_80[0]	
activation_42 (Activation)	(None, 112, 112, 128) 0		batch_normalization_41[0]	
conv0d_81 (Conv2D)	(None, 112, 112, 128) 16384		activation_42[0]	
conv0d_82 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_81[0]	
batch_normalization_42 (BatchNorm	(None, 112, 112, 128) 512		conv0d_82[0]	
activation_43 (Activation)	(None, 112, 112, 128) 0		batch_normalization_42[0]	
conv0d_83 (Conv2D)	(None, 112, 112, 128) 16384		activation_43[0]	
conv0d_84 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_83[0]	
batch_normalization_43 (BatchNorm	(None, 112, 112, 128) 512		conv0d_84[0]	
activation_44 (Activation)	(None, 112, 112, 128) 0		batch_normalization_43[0]	
conv0d_85 (Conv2D)	(None, 112, 112, 128) 16384		activation_44[0]	
conv0d_86 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_85[0]	
batch_normalization_44 (BatchNorm	(None, 112, 112, 128) 512		conv0d_86[0]	
activation_45 (Activation)	(None, 112, 112, 128) 0		batch_normalization_44[0]	
conv0d_87 (Conv2D)	(None, 112, 112, 128) 16384		activation_45[0]	
conv0d_88 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_87[0]	
batch_normalization_45 (BatchNorm	(None, 112, 112, 128) 512		conv0d_88[0]	
activation_46 (Activation)	(None, 112, 112, 128) 0		batch_normalization_45[0]	
conv0d_89 (Conv2D)	(None, 112, 112, 128) 16384		activation_46[0]	
conv0d_90 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_89[0]	
batch_normalization_46 (BatchNorm	(None, 112, 112, 128) 512		conv0d_90[0]	
activation_47 (Activation)	(None, 112, 112, 128) 0		batch_normalization_46[0]	
conv0d_91 (Conv2D)	(None, 112, 112, 128) 16384		activation_47[0]	
conv0d_92 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_91[0]	
batch_normalization_47 (BatchNorm	(None, 112, 112, 128) 512		conv0d_92[0]	
activation_48 (Activation)	(None, 112, 112, 128) 0		batch_normalization_47[0]	
conv0d_93 (Conv2D)	(None, 112, 112, 128) 16384		activation_48[0]	
conv0d_94 (Conv2D)	(None, 112, 112, 128) 16384		conv0d_93[0]	
batch_normalization_48 (BatchNorm	(None, 112, 112, 128) 512		conv0d_94[0]	
activation_49 (Activation)	(None, 112, 112, 128) 0		batch_normalization_48[0]	
conv0				

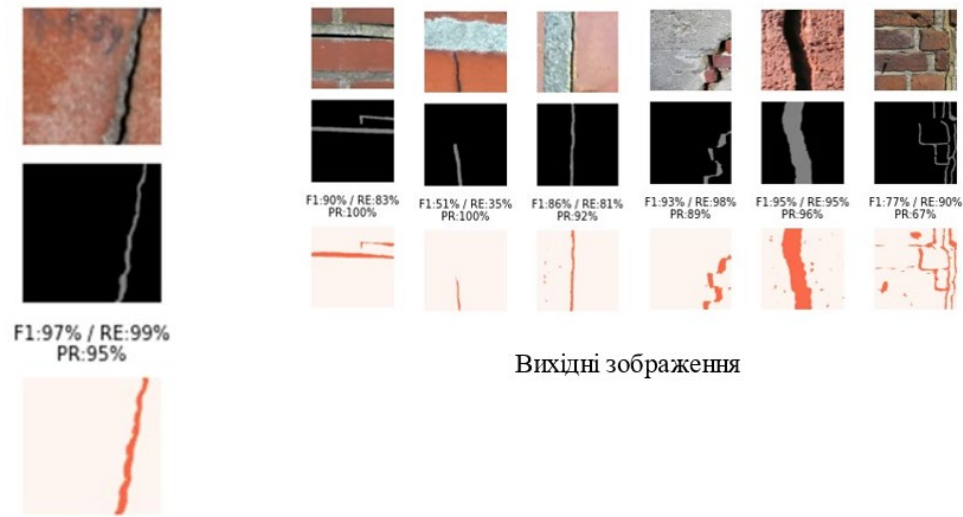
Графік похибки



13

Рисунок Б.13 – Слайд презентації № 13

Результати роботи



14

Рисунок Б.14 – Слайд презентації № 14

Висновки

За результатами виконання дипломної кваліфікаційної роботи бакалавра вирішено актуальну інженерну задачу виявлення тріщин будівель за допомогою методів в основу яких закладено використання неймережевих технологій.

Практична цінність розробленого програмного забезпечення полягає в тому, що воно надає гнучкі можливості навчання неймережевої моделі U-Net. Програма має зацікавити будівничі компанії або компанії, що надають дослідницькі дані таким будівничим компаніям, бо задовольняє їх потреби у процесі збору даних.

В подальшому буде додано можливість роботи з програмним забезпеченням за допомогою командного рядка, що дозволить виконувати одночасну обробку великої кількості вхідних даних. Окрім цього буде додано нові екземпляри до навчальної вибірки, що дозволить покращити виявлення порівняно тонких тріщин, які може виявляти програма. Також планується покращення графічного інтерфейсу та додавання мовних пакетів.

Основні ідеї щодо застосування роботи було представлено в тезах доповіді науково-практичної конференції Тиждень науки – 2021, Судаков В. Р. «Розробка програмного забезпечення для реалізації методів розпізнавання тріщин будівель».

15

Рисунок Б.15 – Слайд презентації № 15

Дякую за увагу

16

Рисунок Б.16 – Слайд презентації № 16