Шифр: CNN-DEFECTS

РОЗРОБКА ГЛИБОКОЇ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ПОШКОДЖЕНЬ НА МЕТАЛЕВИХ ПОВЕРХНЯХ ТА ДОСЛІДЖЕННЯ ВПЛИВУ ОСВІТЛЕННЯ НА РЕЗУЛЬТАТ РОЗПІЗНАВАННЯ

ЗМІСТ

ВСТУП

Розпізнавання дефектів металевих поверхонь за допомогою аналізу їх зображень є перспективним, але складним науково-технічним завданням. Причина складності – наявність багатьох чинників, які впливають як на формування самих дефектів, так і на процес їх розпізнавання. Зокрема, поверхня може містити різні види дефектів, які залежать від способу виготовлення та умов експлуатації. Геометрія та форма цих дефектів також є різноманітною, що додатково ускладнює їх виявлення. Особливості морфології самої поверхні зразка також можуть впливати на результат розпізнавання: у ряді випадків допустимі поверхневі утвори можуть бути візуально схожі на пошкодження. Отже, пошкодження можуть мати різний розмір, форму, текстуру. Тому розробити універсальний та ефективний алгоритм розпізнавання для кількісної оцінки пошкоджень складно.

Останнє десятиліття найкращих результатів у розпізнаванні об'єктів на зображеннях досягнуто за допомогою нейронних мереж [1-3]. Вони дозволяють сформувати під час навчання достатньо повний набір ознак, притаманних пошкодженням, і ефективно їх виявляти. Сегментоване зображення дозволяє не тільки локалізувати пошкодження у просторі, але й обчислити цілий ряд кількісних параметрів, які описують дефекти (площу, розміри, коефіцієнт форми, нахил тощо).

Важливим фактором, який впливає на процес виявлення об'єктів на зображеннях, є освітлення. Нерівномірне або недостатнє освітлення суттєво впливає на отримане камерою зображення, візуально підсилюючи або послаблюючи різні морфологічні або поверхневі утвори. Як наслідок – нейромережева модель може мати додаткові похибки для такого виду зображень, а розраховані кількісні параметри можуть суттєво змінюватись.

Тому при розпізнаванні важливим завданням є дослідження впливу рівня освітленості на дослідний зразок. Це дозволяє конкретизувати можливості

навченої моделі, а також знайти оптимальні умови, при яких модель даватиме повторюваний та точний результат.

Таким чином, **актуальність** вибраної теми наукової роботи зумовлена:

1) актуальністю завдання розпізнавання поверхневих дефектів за допомогою фотозображень для різних галузей промисловості та різних видів техніки;
2) підтвердженими високоточними результатами нейронних мереж при вирішенні завдань класифікації та сегментації зображень, можливістю пристосування нейромережі до різноманітних видів шуканих об'єктів;
3) важливістю дослідження впливу освітленості дослідного зразка на результат розпізнавання дефектів нейронною мережею та, як наслідок, відхилення значень розрахованих кількісних параметрів дефектів;
4) важливістю розробки програмного забезпечення для лабораторних досліджень зразків з нерівномірною та змінною освітленістю, яке дозволяє автоматизувати процес отримання та аналізу результатів, що описують дефекти на поверхні.

Мета і завдання дослідження. Метою є вибір архітектури та тренування нейронної мережі для розпізнавання дефектів; розробка застосунку для розпізнавання зображень за допомогою нейронної мережі; дослідження за допомогою розроблених компонентів впливу освітленості дослідного зразка на кінцевий результат розпізнавання; виявлення статистичних залежностей розкиду розрахованих кількісних параметрів дефектів; виявлення діапазону освітленості, який забезпечує найкращий рівень розпізнавання та стійкий повторюваний результат.

Робота є теоретично-прикладною і передбачає навчання нейронної мережі та розробку архітектури відповідного програмного рішення, її імплементацію, проведення експериментів по отриманню зображень пошкодженої поверхні із змінною освітленістю та дослідження зображень, отриманих у результаті експерименту.

Для досягнення поставленої мети передбачено вирішення наступних завдань:

- аналізу існуючих архітектур нейронних мереж для розпізнавання та аналізу зображень;
- формування навчальної множини зображень з поверхневими пошкодженнями для тренування нейронної мережі;
- навчання нейронної мережі для розпізнавання пошкоджень (вирішення завдання семантичної сегментації);
- розробку застосунку для розпізнавання зображень та розрахунку множини кількісних параметрів, що описують розпізнані пошкодження;
- проведення експерименту для отримання зображень сталевої поверхні з пошкодженнями при різних рівнях освітленості поверхні;
- дослідження за допомогою розробленого застосунку отриманих експериментальних зображень.

Об’єктом дослідження є цифрові зображення металевої поверхні з пошкодженнями, отримані при різних параметрах освітленості.

Предметом дослідження є варіація кількісних параметрів пошкоджень, розпізнаних за допомогою нейромережевої моделі на дослідному зразку при його змінному освітленні.

Методика дослідження передбачає формування вибірки даних для навчання нейронної мережі; навчання згорткової нейронної мережі для розпізнавання дефектів на зображеннях металевої поверхні; розробку Python-застосунку для розпізнавання отриманих зображень, розрахунку кількісних параметрів пошкоджень і візуалізації їх статистичного розподілу; проведення за допомогою розроблених програмних компонентів експериментів з виявлення пошкоджень на металевій поверхні при різних рівнях освітлення; дослідження отриманих зображень за допомогою розробленого застосунку; та аналізу отриманих даних.

Прикладною метою роботи є навчання нейромережі та розробка застосунку для розпізнавання поверхневих пошкоджень на зображенні за допомогою глибокої згорткової нейронної мережі, розрахунку кількісних параметрів

пошкоджень та дослідження отриманих зображень. Для розробки вибрано мову програмування Python. Роботу з нейромережевими моделями реалізовано за допомогою бібліотеки Keras, яка надає зручний інтерфейс над бібліотекою TensorFlow від компанії Google. Для безпосередньої роботи з зображеннями (читання, побудова тензорів, фільтрування, поворот тощо) використано бібліотеку OpenCV для Python.

РОЗДІЛ 1. ВИБІР АРХІТЕКТУРИ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ДЕФЕКТІВ ТА ЇЇ НАВЧАННЯ

1.1. Аналіз нейромережевих архітектур

На даний час розроблено достатньо багато моделей нейронних мереж, які використовують для розпізнавання об'єктів на зображеннях. У більшості з них на низькому рівні базовою архітектурою виступає згорткова модель. Вона є основою більшості моделей для класифікації та сегментації зображень, так як дозволяє ефективно формувати карти ознак, притаманні шуканим об'єктам. На вищому рівні абстракції базовою є модель для класифікації. Вона є основою складніших моделей для пошуку об'єктів та семантичної сегментації, оскільки забезпечує виявлення особливостей шуканих об'єктів. Для вирішення завдань класифікації зображень хороші результати задекларовано при застосуванні архітектур ResNet [4], DenseNet [5], Inception [6-8] та EfficientNet [9], оскільки вони дозволяють побудову глибоких моделей з великою кількістю шарів, що дозволяє виявляти більше характерних ознак об'єктів. Існують також нейромережі з архітектурами, оптимізованими для мобільних платформ (MobileNet, EfficientNet). Вони мають дещо меншу точність розпізнавання, ніж складніші ResNet чи DenseNet, але разом з цим є значно простішими, менш вимогливими до апаратного забезпечення та швидшими.

Модель ResNet була запропонована у 2015 році розробниками Microsoft і дозволила будувати глибокі згорткові мережі з високою швидкістю навчання [4]. Такі нейромережі відразу показали дуже високі результати у завданнях класифікації та сегментації зображень. Модель ResNet складається з послідовно з'єднаних залишкових блоків. Кожен такий блок містить три шари, які виконують згортання. Але головною особливістю моделей ResNet є так звані "швидкі з'єднання", які пропускають один або кілька шарів і з'єднують вхід одного шару з виходом іншого. Застосування швидких з'єднань дозволяє усунути проблему спадаючого градієнта та одночасно суттєво збільшити глибину моделі. Ця

перевага дає можливість оптимізувати мережі ResNet, оскільки помилка навчання не так різко зростає при збільшенні глибини моделі.

Для виявлення на зображеннях множин пікселів, що належать шуканим об'єктам (завдання семантичної сегментації) використовують архітектури, які складаються з енкодера та декодера. Енкодер дозволяє сформувати карту характерних для шуканих об'єктів ознак, а декодер – спроектувати розпізнані на зображенні ознаки назад на початкове зображення, підкреслюючи цим ділянки з об'єктами.

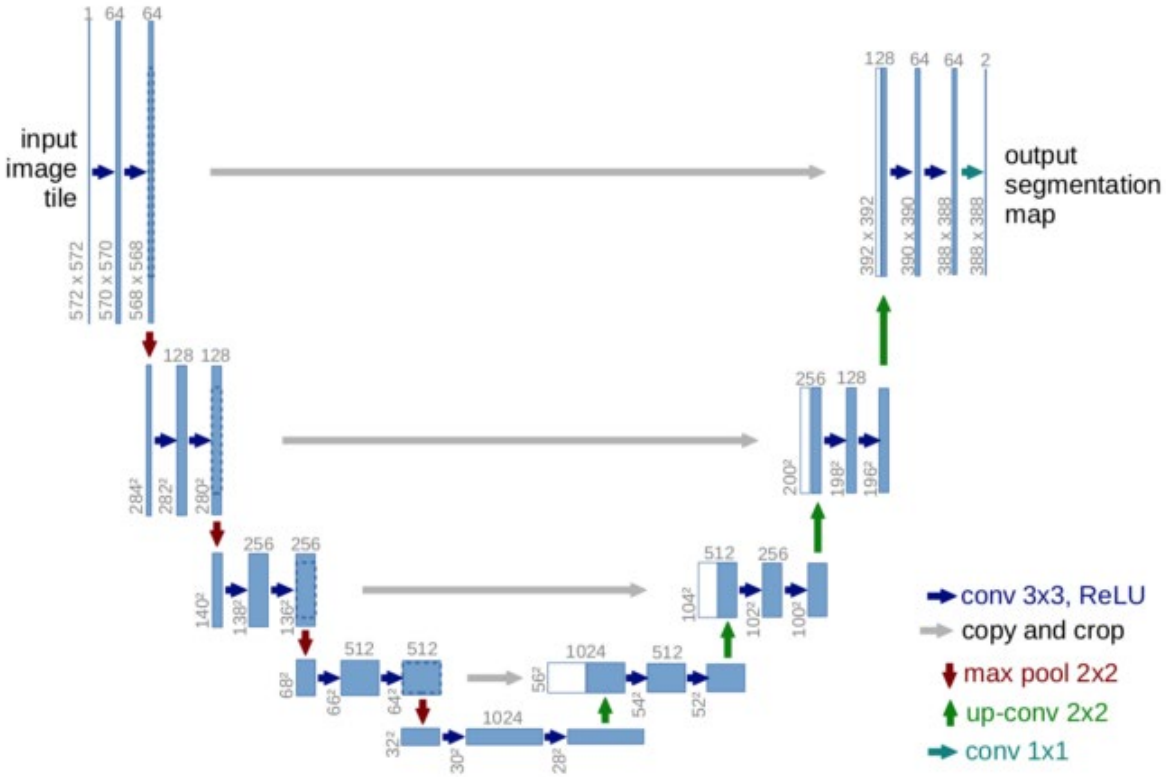


Рис. 1. Загальна архітектура нейромережі U-net. Синіми прямокутниками зображено багатоканальні карти ознак. Кількість каналів зазначено над ними. Білими прямокутниками позначено скопійовані карти ознак. Розмір карти ознак зазначено біля нижнього лівого краю. Стрілки показують напрямок операцій

Однією з найпоширеніших моделей семантичної сегментації на основі енкодера та декодера є U-net та похідні від неї моделі (рис. 1). Вони довели свою

ефективність у різних сферах, де вирішується завдання сегментації зображень [10-14]. При цьому у ролі енкодера можна використати раніше згадані згорткові моделі, які мають високу точність.

Нейромережа U-net розроблена у 2015 році для обробки медичних зображень. Її архітектура отримана в результаті розвитку і вдосконалення звичайних згорткових нейронних мереж. Для задач дефектометрії важливо не тільки класифікувати зображення на такі що містять і не містять дефекти. Важливо також мати точну локалізацію пошкодження. Це дозволяє не тільки покращити контроль якості продукції, а й розрахувати кількісні характеристики пошкоджень (площу, напрям тощо). Що, у свою чергу, дає можливість краще розуміти природу пошкоджень і розробити кроки для їх усунення. Для досягнення цього нейромережа U-net забезпечує семантичну сегментацію зображення, де цього кожен піксель зображення класифікується як приналежний до одного з класів пошкоджень, або до неушкодженої ділянки. При цьому вхідне та вихідне зображення мають однаковий розмір.

Тому для розпізнавання пошкоджень на зображеннях вирішили застосувати не оригінальну версію моделі U-net, а архітектуру U-net з основою ResNet, яка неодноразово доводила свою ефективність при обробці зображень. Так як метою роботи є розробка інструмента для лабораторних досліджень, які не є вимогливими до часу опрацювання зображення але чутливими до точності розпізнавання, для подальшої розробки вибрали енкодер на основі ResNet152.

1.2. Навчальні дані

Навчальну вибірку для нейронної мережі сформували на основі двох баз зображень, які є у загальному доступі:

- зображення стального прокату від компанії "Сєвєрсталь" [15];
- зображення дефектів з дослідження "Виявлення дефектів на поверхні стального прокату " [16, 17].

Основу навчальної вибірки становили зображення, надані сталеливарною і гірничодобувною компанією "Сєвєрсталь" в рамках змагання з аналітики та моделювання, організованого на платформі Kaggle у 2019 році [15]. Початкова база даних Severstal містить зображення у градаціях сірого кольору розміром 1600×256 пікселів.

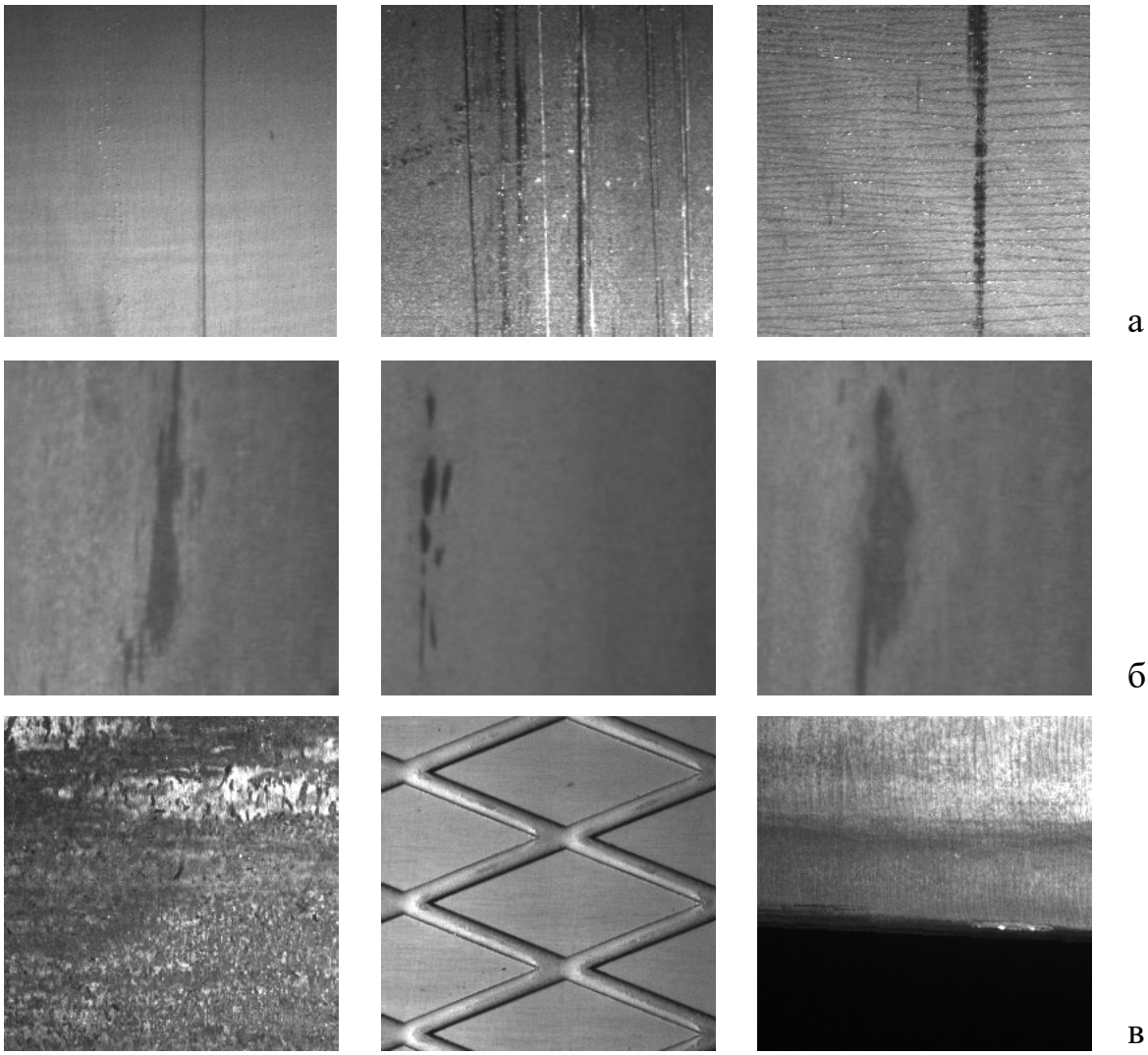


Рис. 2. Приклади зображень з пошкодженнями (а, б), та без пошкоджень (в) які використано для навчання нейронної мережі

Другу вибірку даних взято із платформи Kaggle із дослідження "Saliency detection for strip steel surface defects" [17]. База зображень опублікована в 2020 році. Вона містить три види пошкоджень: Inclusion, Patches, and Scratches і в загальному містить 900 зображень пошкоджень розміром 200×200 пікселів. Їх

також приведено до розміру 256×256 за допомогою алгоритму бікубічної інтерполяції.

На рис. 2 приведено приклади вхідних зображень пошкоджень із використаних джерел (а,в – "Сєвєрсталь", б – "Saliency detection").

Навчальні зображення були відібрані, перевірені та розмічені групою експертів. Загальний обсяг навчальної вибірки становив 11000 зображень (5800 з дефектами різних форм та напрямків, і 5200 без дефектів). Навчальну вибірку розділено на три частини: тестову (вона складала 10% від загальної кількості зображень), валідаційну (15%) і навчальну (75%). Навчальну і валідаційну вибірки використовували при навчанні нейромережі, а тестову – для оцінювання натренованої моделі.

1.3. Метрики якості моделі

Для оцінювання якості сегментації використовували метрику DSC – коефіцієнт подібності Дайса (Dice similarity coefficient, рис. 3):

$$DSC = \frac{2\left|Y_{true} \cap Y_{pred}\right|}{\left|Y_{true}\right| + \left|Y_{pred}\right|} \tag{1.1}$$

де Y_{true} – дійсний об'єкт пошкодження відповідно до розмітки, Y_{pred} – об'єкт пошкодження, отриманий після сегментації.

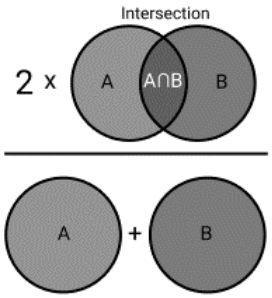


Рис. 3. Ілюстрація розрахунку метрики якості сегментації DSC

На практиці для розрахунку DSC використовували вираз:

$$DSC = \frac{2\sum Y_{true} \cdot Y_{pred} + \varepsilon}{\sum Y_{true} + \sum Y_{pred} + \varepsilon} \quad (1.2)$$

де $\varepsilon = 1$ – допоміжний доданок для запобігання діленню на нуль.

Y_{true} є масивом цілих чисел, де 0 позначає піксель фону, а 1 – піксель пошкодження. З вихідних сигмоїдних нейронів вихідного шару отримуємо дійсні значення в діапазоні [0,1]. При тестуванні після порогування Y_{pred} також міститиме цілі значення. Але під час навчання оптимальний поріг невідомий, тому Y_{pred} міститиме дійсні значення, в результаті чого значення метрики буде меншим.

Оскільки вихідні значення нейромережі знаходяться в діапазоні [0,1], то для обчислення найкращих тестових метрик подібності DSC їх розраховували для різних порогів.

Крім метрики DSC використовували метрики для оцінювання бінарної класифікації пікселів на класи (дефект – не дефект): $Precision$, $Recall$, $F1-score$.

Метрика $Recall$ показує, яка частина пікселів, що належать до пошкодження, розпізнана правильно:

$$Recall = \frac{TP}{(TP + FN)}. \quad (1.3)$$

Метрика $Precision$ демонструє, яка частина пікселів, що належать до пошкоджень, насправді визнані належними до пошкоджень:

$$Precision = \frac{TP}{(TP + FP)}. \quad (1.4)$$

Метрика $F1$ є інтегрованою і її можна інтерпретувати як середньозважене значення $Recall$ та $Precision$. Оцінка $F1$ досягає найкращого значення при 1, а найгіршого – при 0. Відносні внески $Recall$ та $Precision$ в оцінку $F1$ одинакові.

$$F1 = 2 \cdot \frac{(Precision \cdot Recall)}{(Precision + Recall)}. \quad (1.5)$$

1.4. Навчання нейронної мережі для розпізнавання пошкоджень

При навчанні нейронної мережі важливо вибрати оптимізатор. Це алгоритм чи метод, який використовується для зміни параметрів нейронної мережі, зокрема ваг нейронів з метою досягнення найменших втрат. Оптимізатор реалізує стратегію зменшення втрат і дозволяє досягти найточнішого результату. Використовували класичний оптимізатор SGD (Stochastic gradient descent) з моментом Нестерова, який дозволяє прискорити оптимізатор у правильному напрямку і згладжує коливання функції втрат. Метод стохастичного градієнта є одним з найчастіше застосовуваних для оптимізації при навчанні глибоких нейронних мереж [18-20].

При тренуванні моделі використовували функцію фокальних втрат [21]. Її зручно використовувати у випадку незбалансованих даних, коли кількість зображень різних класів суттєво відрізняється. Фокальна функція втрат зменшує вагу добре класифікованих прикладів і приділяє більше уваги тим зразкам, які представлені менше.

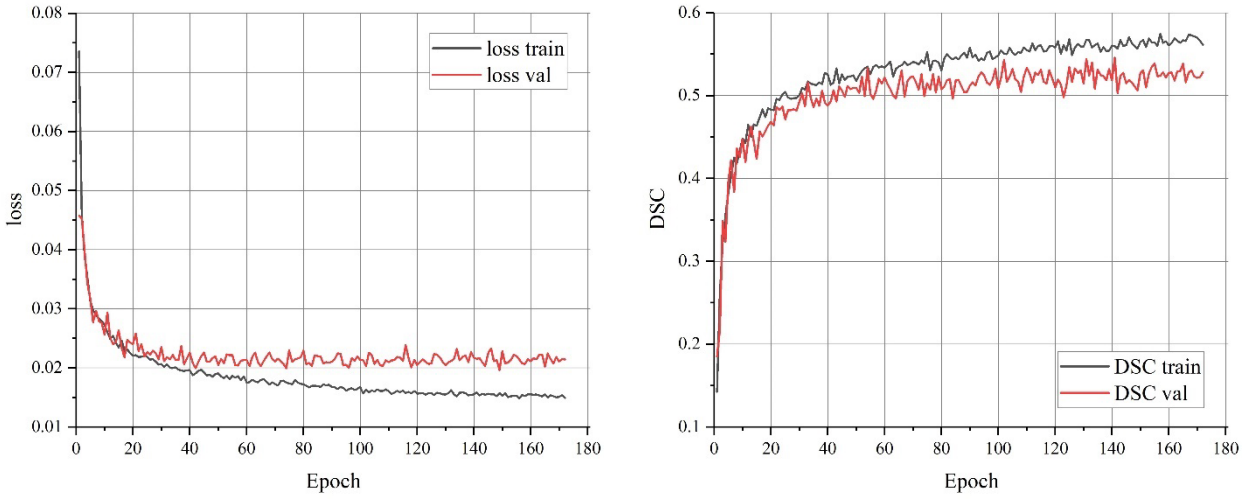
Щоб нейромережа при тренуванні не починала формування карт ознак випадковим чином, застосували техніку перенесення навчання. Для цього перед навчанням нейромережу було ініціалізовано вагами, отриманими при навчанні на базі зображень ImageNet, яка в загальному містить більше 1.4 мільйона зображень, розділених на 1000 різних класів.

Навчальна вибірка містить зображення різних розмірів. Тому під час навчання на зображеннях, більших за 256×256 пікселів, випадковим чином вибирались ділянки розміром 256×256 пікселів (відповідно до розміру вхідного шару нейромережі), з яких формувався тензор, що подавався на вхід. Крім цього, застосували техніку аугментації (вона дозволяє досягти кращих показників кінцевої моделі). Для цього кожен кадр (зображення) випадковим чином трансформували (використали горизонтальне або вертикальне відображення, а також поворот на кратний 90^{o} кут). Використаний підхід дозволив суттєво урізноманітнити навчальні дані і забезпечив умови, при яких навчальні пакети на практиці ніколи не повторюються.

Нейромережеві моделі сегментації було реалізовано мовою Python 3.8 за допомогою бібліотек TensorFlow та Keras. Для обробки зображень використано бібліотеку OpenCV. Нейронну мережу навчали та тестували на робочій станції з процесором Intel Core i7-2600 CPU, 32 GiB RAM та двома відеокартами NVIDIA GeForce GTX 1060 з 6 GiB відеопам'яті.

При навчанні нейронної мережі після кожної епохи проміжну модель зберігали. Якщо функція втрат не зменшувалась протягом 10 попередніх епох, швидкість навчання зменшували на 20%. В кінці кожної епохи крім моделі зберігали навчальні та валідаційні метрики якості DSC, $Precision$, $Recall$ та $F1-score$. Навчання припиняли, коли швидкість навчання ставала меншою за 0.0001, або функція втрат не покращувала своє значення протягом 20 епох.

Як остаточну модель вибрали модель з найкращою метрикою DSC в тестовому наборі даних. Для неї тестова метрика $DSC@0.55 = 0.93$. Динаміку навчання моделі показано на рис. 4.



a b

Рис. 4. Динаміка зміни функції втрат (а) і метрики DSC (б) при навчанні моделі

Оскільки під час навчання значення оптимальної межі порогування невідоме, то метрику DSC розраховували на основі поточних дійсних значень вихідного шару. Тому величина валідаційних метрик суттєво менша за кінцеву

метрику, отриману на тестових даних для величини порогу 0.55. Графіки демонструють поступовий процес навчання, але приблизно після 100 епох показники досягають певної межі і суттєво не змінюються.

РОЗДІЛ 2. МЕТОДИКА ДОСЛІДЖЕННЯ ЗОБРАЖЕНЬ ПОВЕРХНЕВИХ ПОШКОДЖЕНЬ ПРИ РІЗНОМУ ОСВІТЛЕННІ

2.1. Експериментальна установка

Схема дослідної установки для досліджень поверхневих пошкоджень металевих зразків показана на рис. 5. Для вимірювання освітленості використовували камеру, всередині якої встановлювали досліджуваний зразок та давач освітленості.

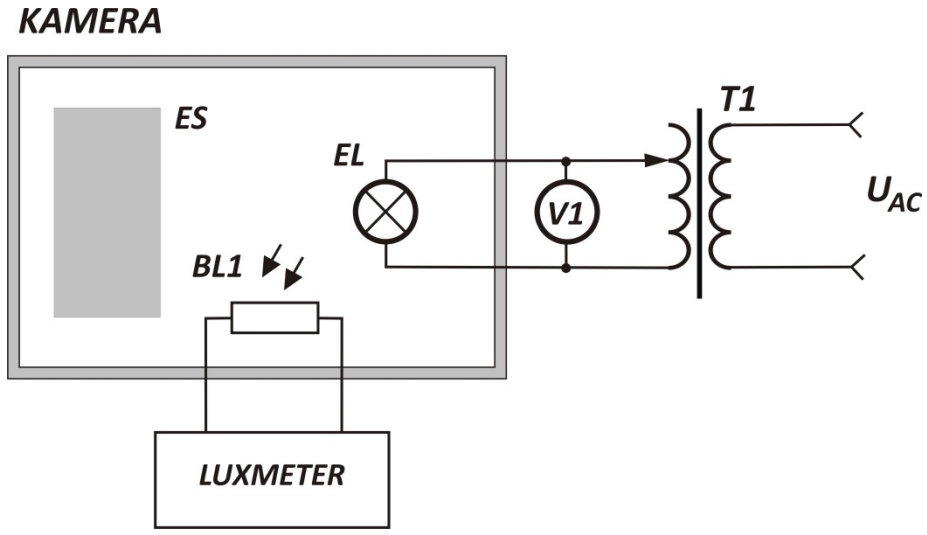


Рис. 5. Блок-схема отримання зображень експериментальних зразків. T1 – трансформатор, V1 – вольтметр змінного струму, BL1 – датчик люксметра, EL – джерело світла

Фото експериментальної установки показано на рис. 6. Вона має вид камери із світлонепроникними стінками та знімною кришкою, яка у закритому стані щільно прилягає до стінок. Дослідний зразок розміщують на дні короба. Лампа з регульованою освітленістю прикріплена до внутрішньої сторони кришки. У верхній частині камери змонтовано скляну ділянку для під'єднання об'єктиву фотокамери, що забезпечує можливість фотофіксації поверхні дослідного зразка для подальшого аналізу. Регулятор яскравості лампи виведено

назовні короба. За його допомогою на лампу подавали різну напругу, для зміни сили світла від джерела.

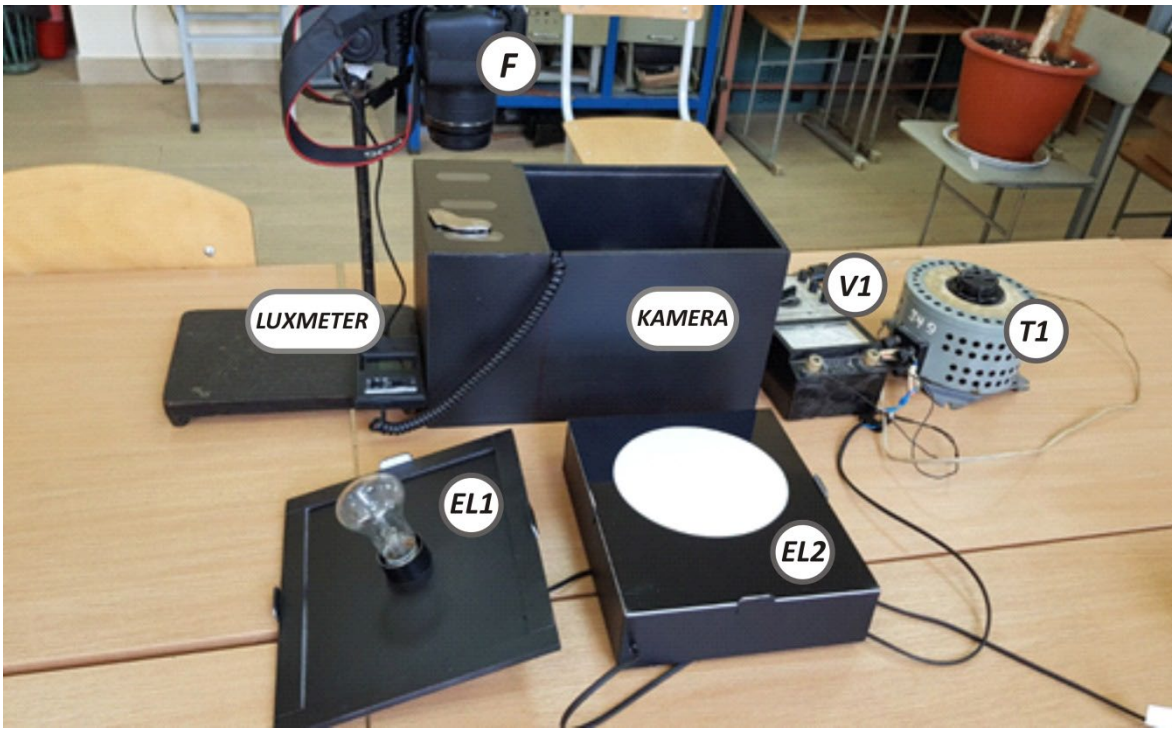


Рис. 6. Загальний вигляд експериментальної установки. T1 – трансформатор, V1 – вольтметр змінного струму, EL1 – модуль світла з лампою розжарювання, EL2 – світлодіодний модуль світла, F – фотоапарат для фотофіксації результатів досліджень

Давач освітленості прикріплено до дна поблизу місця для дослідного зразка. Рівень освітлення фіксували за допомогою люксметра, з'єднаного з давачем.

Описана конструкція установки дозволяє отримувати фотозображення поверхні дослідного зразка при змінному освітлені від використаного джерела, усуваючи при цьому вплив зовнішнього світла.

Для освітлення використано джерело світла, побудоване на основі світлодіодних модулів, що містять світлодіоди SMD 2835 з колірною температурою 3000 К, загальною потужністю 7 Вт та розміром 20×20 см. Для рівномірнішого освітлення використано світлорозсіюючий пластик.

Зразки ES, що досліджувалися, розміщувалися на дні камери. Напруга змінного струму на виході трансформатора T1 змінювалася із кроком 10В в межах від 150В до 230В, її величина контролювалася вольтметром V1. До вихідної обмотки трансформатора під'єднали джерело світла EL (джерело з рівномірною освітленістю, побудоване на основі світлодіодних модулів). Джерела світла виконані у вигляді змінних модулів і розташовувалися у верхній частині камери. Чутливим елементом BL1 люксметра LUXMETER вимірювалися поточні значення освітленості E. Частина верхньої кришки камери виконана прозорою для можливості фотофіксації результатів освітлення зразка за допомогою фотоапарата.

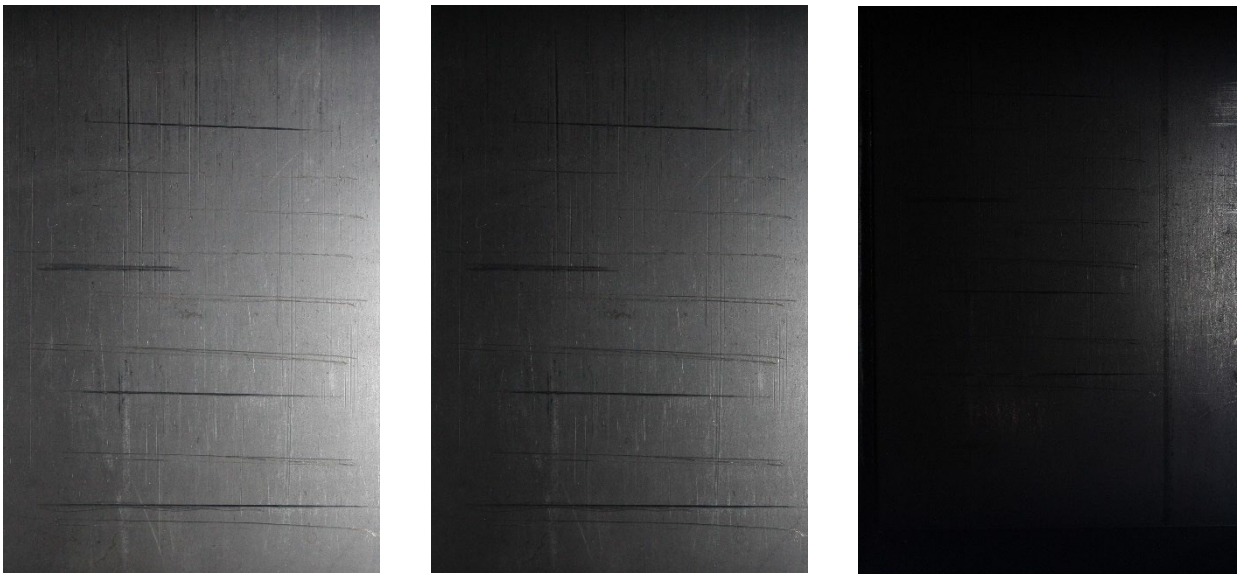
На схемі (рис. 6) T1 – трансформатор, V1 – вольтметр змінного струму, BL1 – датчик люксметра, EL1 – модуль світла з лампою розжарювання, EL2 – світлодіодний модуль світла, F – фотоапарат для фотофіксації результатів досліджень.

Таблиця 1. Умови отримання фотозображень поверхневих дефектів

№	Напруга на джерелі світла, В	Освітленість в зоні зразка, Лк
1.	13,0	330
2.	12,5	295
3.	12,0	265
4.	11,5	222
5.	11,0	192
6.	10,5	165
7.	10,0	139
8.	9,5	98
9.	9,0	78
10.	8,5	45
11.	8,0	21

Для більш точного вимірювання освітленості стінки камери пофарбовані в чорний колір. Чутливий елемент BL1 люксметра, ізольований від зовнішнього світла, розташовувався в середині камери. Вимірювання характеристик зразка проводили при кімнатній температурі, через 5 хв після подання напруги на освітлювальний пристрій для встановлення його робочого режиму. Для отримання цифрових зображень використовували камеру Nikon D5600.

Умови, при яких було отримано зображення досліджуваної стальної поверхні з пошкодженнями, приведено у таблиці 1. На рис. 7 показано приклади зображень дослідного зразка, отримані при різному освітленні (330, 139 та 45 Лк).



а б в

Рис. 7. Цифрові зображення стальної поверхні з пошкодженнями типу "подряпина", отримані при освітленні відповідно 330, 139 та 45 Лк

Для отриманих зображень є характерною нерівномірна освітленість, пов'язана з тим, що фотокамера і зразок розташовані збоку від джерела світла. Окремі дефекти (подряпини) дуже добре видимі при яскравому освітленні. При слабкому вони стають менш помітними і часто зливаються з фоном.

2.2. Кількісні параметри пошкоджень

Для об'єктивного кількісного оцінювання знайдених пошкоджень необхідно мати числові показники, які їх описують.

Одні з найчастіше вживаних кількісних параметрів, які використовують для оцінювання геометрії об'єкта (у нашому випадку об'єктом виступає відокремлений фрагмент пошкодження) – це параметри еквівалентного кола або еліпса. Щоб розрахувати їх, необхідно мати центр мас об'єкта. Нехай $f(x, y) \in F$ – множина пікселів зображення, з яких складається об'єкт. Тоді центр мас такого об'єкта має координати:

$$x_c = \frac{\sum_F x_i}{len(F)}, \qquad y_c = \frac{\sum_F y_i}{len(F)} \tag{2.1}$$

Площа пошкодження розраховується як загально кількість пікселів у множині F:

$$S = len(F). \tag{2.2}$$

В загальному випадку розпізнане пошкодження може містити "порожнини", які розпізнані як ділянки, що не є пошкодженими. Це можливо у випадку достатньо великих за площею дефектів (наприклад, множинні подряпини), які нерівномірно розподілені і оточують непошкоджену ділянку. Тому площу можна розраховувати двома способами: без врахування внутрішніх "порожнин" (як у формулі 2.2) і з їх врахуванням (тоді до площі S слід додати ще площу "порожнин").

2.2.1. Еквівалентне коло

Діаметр еквівалентного кола розраховується як діаметр круга, площа якого дорівнює площі пошкодження:

$$d_{eq} = \sqrt{\frac{4S}{\pi}}. \tag{2.3}$$

2.2.2. Еквівалентний еліпс

Параметри еквівалентного еліпса ілюструє рис. 7. Велика вісь еліпса довжиною $2a$ проходить через центр мас об'єкта і має нахил θ. Орієнтація може змінюватися у межах $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ і зростає за годинниковою стрілкою, оскільки вісь y зображення направлена вниз. Ексцентриситет еквівалентного еліпса розраховується як відношення між відстанню від його центру до фокусів та довжиною великої півосі:

$$\varepsilon_{eq} = \frac{c}{2a}. \tag{2.4}$$

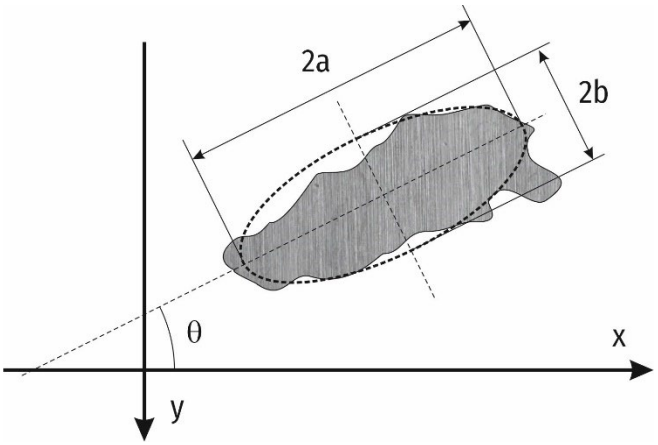


Рис. 7. Параметри еквівалентного еліпса

Параметри еквівалентного еліпса в сукупності дозволяють оцінити як видовженість об'єкта, так і його нахил щодо осей зображення.

Периметр об'єкта пошкодження розраховували як кількість пікселів, які мають хоча б один сусідній піксель, що належить не об'єкту, а фону.

РОЗДІЛ 3. РОЗПІЗНАВАННЯ ПОШКОДЖЕНЬ ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Для розпізнавання на отриманих цифрових зображеннях пошкоджень, а також для розрахунку їх кількісних параметрів, розроблено Python- застосунок. Його призначення- отримання в лабораторних умовах множини кількісних параметрів пошкоджень та аналіз їх варіативності (для різних рівнів освітлення).

Застосунок розроблено як скрипт, який запускається з командного рядка, та призначений для використання у лабораторних умовах. Він дозволяє автоматизувати процес обробки пакету зображень (розпізнати їх за допомогою нейромережевої моделі, розрахувати параметри дефектів, побудувати діаграми їх розподілу, та зберегти результати для подальшого використання).

3.1. Архітектура застосунку для розпізнавання пошкоджень

Функціонал застосунку реалізовано у класах SegmentationClient, DefectDetector, RegPropAnalyzer, ThresholdPropAnalyzer та DefectAnalyzer.

Клас SegmentationClient використовує навчену нейронну мережу для розпізнавання зображень. Він використовує методи бібліотеки Keras [22] щоб розпізнати пошкодження на заданому зображенні. Далі за допомогою методів бібліотеки scikit-learn [23] для кожної відокремленої ділянки розпізнаних пошкоджень розраховуються кількісні параметри. Результати записуються у csv-файл. Клас DefectDetector є обгорткою над SegmentationClient і дозволяє пакетно обробити всі зображення у заданій папці. Клас RegPropAnalyzer читає csv-файл з кількісними параметрами пошкоджень, сформований за допомогою класу SegmentationClient, і будує діаграми статистичного розподілу заданих параметрів для різних рівнів освітленості. При цьому використано діаграми виду "box-plot". Клас DefectAnalyzer використовує функціонал попередніх класів і будує діаграми залежностей для заданої множини рівнів освітленості.

UML- діаграму класів, які використано у застосунку, зображено на рис. 8.

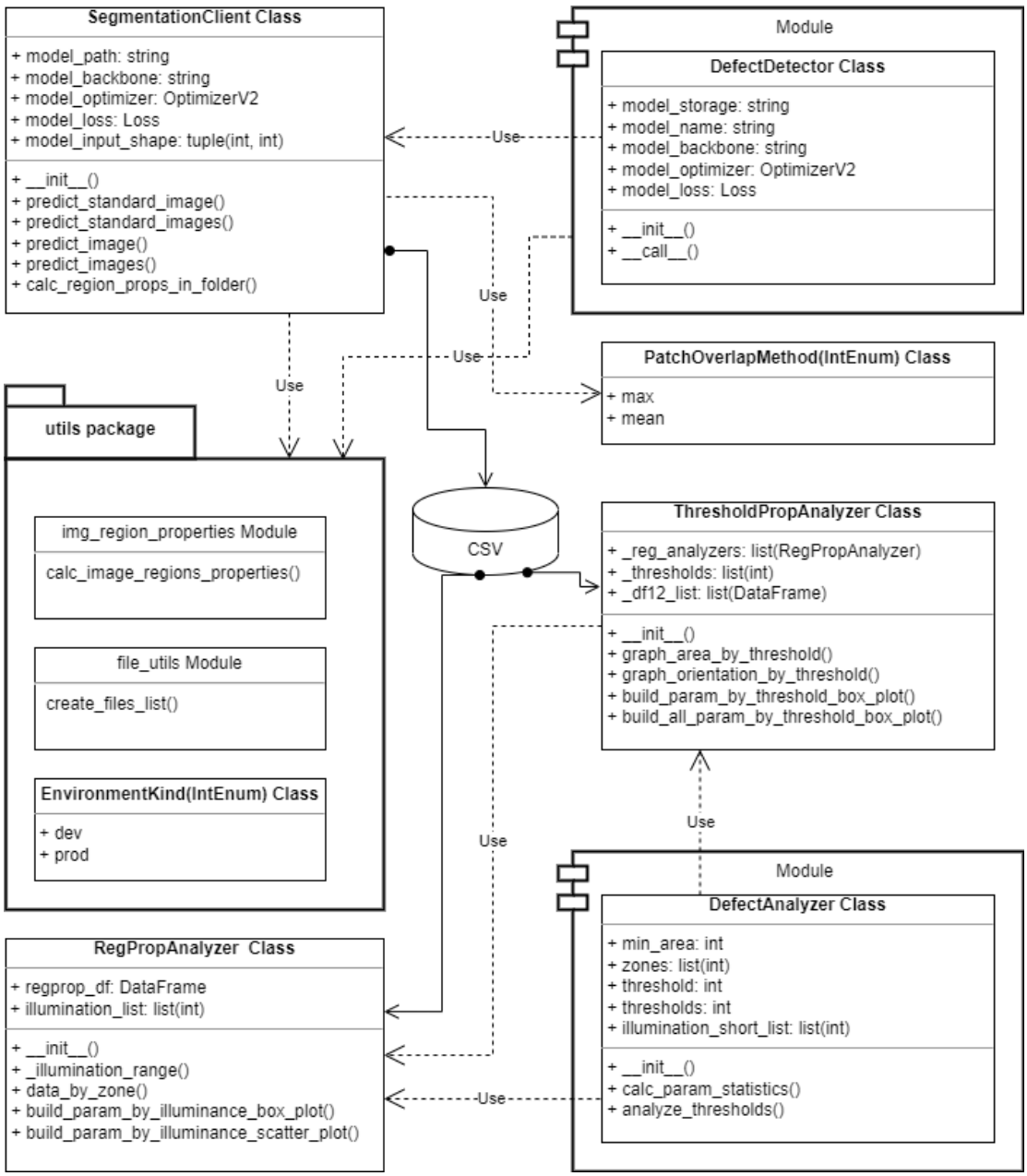


Рис. 8. UML-діаграма класів застосунку для розпізнавання пошкоджень на зображеннях

3.2. Результати розпізнавання

За допомогою розробленого застосунку було опрацьовано серію зображень, отриманих у результаті експерименту при різних рівнях освітлення зразка з поверхневими пошкодженнями.

На рис. 9 показано початкові зображення поверхневих пошкоджень при освітленні 330, 139, 45 та 21 лк (а) та результати їх розпізнавання за допомогою нейронної мережі (б).

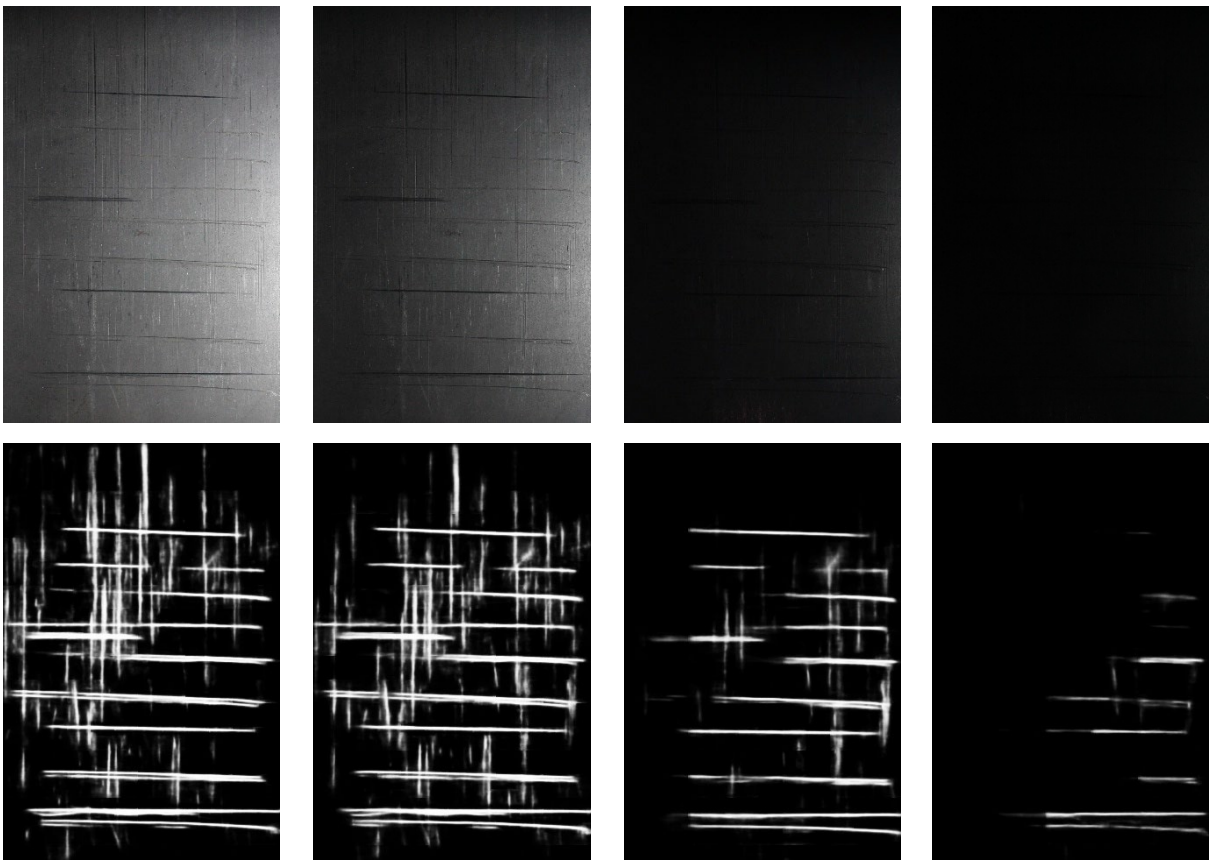


Рис. 9. Цифрові зображення стальної поверхні з пошкодженнями типу "подряпина", отримані при освітленні відповідно 330, 139, 45 та 21 лк

З отриманих результатів видно, що із збільшенням кількості світла, яке потрапляє на досліджувану поверхню, площа ділянок, розпізнаних як пошкодження, збільшується. Причиною цього є те, що дрібні подряпини (та інші морфологічні елементи поверхні) стають краще видимими. Особливо помітним є перехід від низького рівня освітлення (21 лк) до вищого (45 лк). Якщо у першому випадку вертикальні подряпини на зображенні не видимі взагалі, то при 45 лк частина з них вже розпізнається. При освітленні 139 лк нейромережа розпізнає більшість з них.

Зростання рівня освітленості від 139 до 330 лк майже нічого не додає для виявлення нових подряпин (хіба що дуже дрібних), але разом з цим робить об'єкти пошкоджень дещо більшими. Тому в цілому загальна площа пошкоджень на зображенні із збільшенням освітленості зростає.

Зауважимо, що нейронна мережа дуже чутлива навіть при слабкому освітленні: найбільш виражені пошкодження добре розпізнаються при 45 лк, незважаючи на те, що фотозображення є суттєво затемненим і візуально пошкодження практично невидимі (рис. 9,а, третє зображення). Проте при освітленні 21 лк (рис. 9,а, останнє зображення) пошкодження вже не розпізнаються. У цьому випадку на початковому фотозображенні інтенсивність пікселів у зоні з нерозпізнаними пошкодженнями коливається в межах від 1 до 5. Іншими словами, можна стверджувати, що при освітленні 21 лк отримане вхідне зображення є недостатньо інформативним, щоб робити висновок про наявність на ньому шуканих об’єктів. Таким чином, на всіх зображеннях, де між пікселями пошкоджень та фону зберігається навіть невеликий градієнт, нейромережева модель добре розпізнає шукані об’єкти.

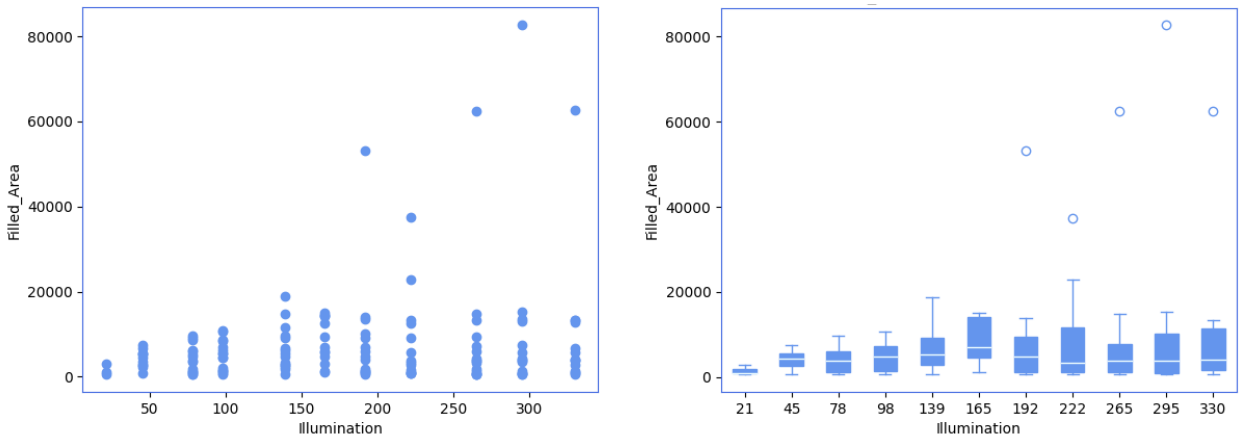
Зображення з низьким рівнем освітлення мають ще одну особливість: на них стають невидимі дрібні об’єкти. Тому, якщо потрібно розпізнати найбільш виражені пошкодження, доцільно використовувати невисокий рівень освітлення. Це забезпечить значно менший рівень шуму на зображенні, викликаного зростанням деталізації (при високому освітленні) поверхневих артефактів, які можуть і не бути пошкодженнями.

Одним з найважливіших та найочевидніших параметрів, які характеризують поверхню, є площа пошкоджень. На рис. 10 приведено діаграму зміни загальної площі всіх елементів зображення, розпізнаних як пошкодження.

Рис. 10. Залежність площі, розпізнаної як пошкодження, від освітлення

При зміні освітлення в діапазоні від 20 до 140 лк загальна площа, розпізнана як пошкодження, помітно зростає від 5 тис. пікселів до 100 тис. пікселів. Подальше зростання освітлення не приводить до суттєвого зростання цього параметра (його значення коливаються в межах 100-125 тис. пікселів). Це вказує на те, що нейромережа виявила всі пошкодження, для яких вона навчена, і подальше зростання освітлення не додає до зображення нових деталей, які б могли бути ідентифіковані як пошкодження.

Розподіл площ розпізнаних на зображенні фрагментів пошкоджень приведено на рис. 11,а,б. На діаграмі розсіювання (рис. 11,а) кожен маркер відповідає окремому фрагменту пошкодження. З графіка можна зробити висновок, що більшість фрагментів має площі до 15 тис. пікселів, а площі більші за 20 тис. пікселів мають поодинокі фрагменти. Діаграма розмаху (рис. 11,б) містить процентилі, які характеризують розкид площ. Нижня межа прямокутника відповідає 25-тому процентилю, верхня – 75-ому процентилю, а лінія між ними – 50-ому процентилю (медіані). Процентиль показує, яка частка (кількість) всіх пошкоджень мають площу, меншу за вказану. Наприклад, 25-ий процентиль зі значенням площі 1 тис. пікселів означає, що 25% всіх фрагментів мають площу, меншу за 1 тис. пікселів.



а б

Рис. 11. Діаграми розподілу площ розпізнаних фрагментів пошкоджень

Лініями- викидами на такій діаграмі показано значення, що знаходяться у межах ±1.5 міжквартильного інтервалу $IRQ = a_{75} - a_{25}$ (де a_{75}, a_{25} – відповідно 75-ий та 25-ий процентилі).

З діаграм розкиду площ можна зробити висновок, що незалежно від освітлення половина (медіана) всіх розпізнаних фрагментів пошкоджень має площу до 7 тис. пікселів. Кількість дуже великих фрагментів зростає із збільшенням освітленості. Так, при освітленні від 192 лк на зображенні з'являються фрагменти пошкоджень площею 40 тис. пікселів ті більші. Причиною цього є те, що при зростанні освітлення розпізнається більше дрібних пошкоджень, і якщо вони розташовані близько чи перетинаються, то зливаються в один великий фрагмент.

Найменший розкид площ фрагментів спостерігається при освітленні 100 лк. У контексті дослідженого зразка це пояснюється тим, що при низькому освітленні розпізнаються тільки найбільш виражені горизонтальні пошкодження, які мають приблизно однакову форму, напрям і площу. При зростанні освітлення розкид площ більший і приблизно однаковий для всіх рівнів освітлення.

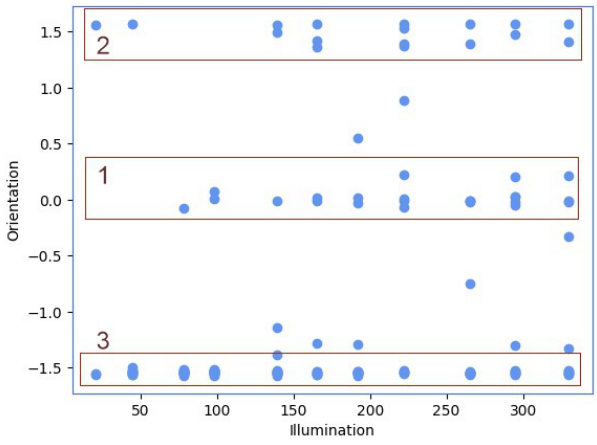


Рис. 12. Діаграма розподілу орієнтації розпізнаних фрагментів пошкоджень

Рис. 12 містить діаграму розсіювання орієнтації (кута нахилу) знайдених фрагментів пошкоджень. На діаграмі кожен маркер зображує розпізнаний фрагмент пошкодження. Можна виділити три групи пошкоджень, нахил яких

знаходиться біля 0 рад (зона 1), та $\pm\frac{\pi}{2}$ рад (зони 2 та 3). Це відповідає візуально впізнаваним напрямам подряпин на дослідному зразку (рис. 9), який містить переважно горизонтальні та вертикальні подряпини. Враховуючи, що пошкодження не представлені вектором напряму, кути $+\frac{\pi}{2}$ та $-\frac{\pi}{2}$ відповідають одній і тій же орієнтації.

Для оцінювання загального впливу освітленості на результат виявлення дефектів, розраховані результати порівняли з розміченими експертом даними. Як метрику використали коефіцієнт подібності Дайса (1.1).

На рис. 13 показано залежність коефіцієнта DSC від рівня освітленості дослідної поверхні. Найкращого результату ($DSC = 0.94$) отримано для освітленості ~160 лк. Менше значення DSC при менших рівнях освітленості означає, що частина пошкоджень ще не розпізнається. Спадання DSC при високому освітленні відбувається головним чином через те, що поверхневі утвори стають краще видимими, а розпізнані фрагменти – ширшими. Крім цього, при високому рівні освітлення з'являються дрібні фрагменти, які хибно ідентифікуються як пошкодження.

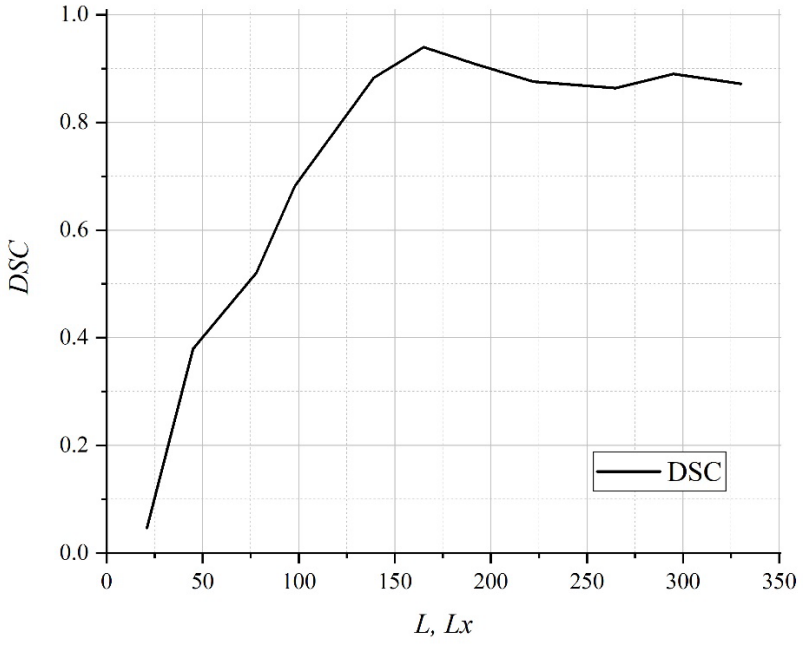


Рис. 13. Залежність коефіцієнта DSC від освітлення дослідної поверхні

ВИСНОВКИ

У роботі розглянуто перспективне технічно-прикладне завдання виявлення дефектів на металевих поверхнях при змінному освітленні. Для розпізнавання дефектів використано глибоку згорткову нейронну мережу семантичної сегментації зображень. Побудовано нейромережеву модель, яка має архітектуру U-net. Як енкодер моделі використано модель ResNet152. Спроектовану модель навчено на зображеннях металевих поверхонь з різних відкритих джерел.

При навчанні використали оптимізатор SGD (Stochastic gradient descent) з моментом Нестерова, який дозволяє прискорити навчання і згладжує коливання функції втрат. Під час навчання моделі застосовували функцію фокальних втрат. Вона зменшує вагу добре класифікованих прикладів і приділяє більше уваги зразкам, які представлені менше. Як метрики якості при навчанні використано DSC, $Precision$, $Recall$ та $F1-score$. Навчання нейромережі припиняли, коли швидкість навчання ставала меншою за 0.0001, або функція втрат не покращувала своє значення протягом 20 епох. Для навченої моделі тестова метрика $DSC@0.55 = 0.93$.

Щоб кількісно оцінити результати розпізнавання пошкоджень, використовували ряд параметрів (площа розпізнаних фрагментів, їх орієнтація тощо). Досліджено вплив освітлення поверхні на результат, який видає нейронна мережа. Вибірку зображень для дослідження сформували через фотографування металевої поверхні зразка з пошкодженнями при різних рівнях освітлення.

Виявлено, що найсильніше впливають на виявлення дефектів малі рівні освітлення (до 140 лк). Загалом вище освітлення приводить до кращої деталізації, і дефекти розпізнаються краще. Але при освітленні більше 200 лк зростає кількість дрібних фрагментів, які хибно розпізнаються як пошкодження. Тому найбільші пошкодження краще розпізнавати при середньому освітленні. Найкращого результату ($DSC = 0.94$) отримано для освітленості ~165 лк.

ПОСИЛАННЯ

1. Liu, Y., Zhang, C. & Dong, X. A survey of real-time surface defect inspection methods based on deep learning. Artif Intell Rev 56, 12131–12170 (2023). https://doi.org/10.1007/s10462-023-10475-7
2. Dan Li, Shiquan Ge, Kai Zhao, Xing Cheng, A Shallow Neural Network for Recognition of Strip Steel Surface Defects Based on Attention Mechanism, ISIJ International, 10.2355/isijinternational.ISIJINT-2022-201, 63, 3, (525-533), (2023).
3. R. Usamentiaga, D. G. Lema, O. D. Pedrayes and D. F. Garcia, "Automated Surface Defect Detection in Metals: A Comparative Review of Object Detection and Semantic Segmentation Using Deep Learning," in IEEE Transactions on Industry Applications, vol. 58, no. 3, pp. 4203-4213, May-June 2022, doi: 10.1109/TIA.2022.3151560.
4. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual learning for image recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, 770-778, doi: 10.1109/CVPR.2016.90.
5. Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks. –2018 – arXiv:1608.06993v5 [cs.CV].
6. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going Deeper with Convolutions. – 2014 – arXiv:1409.4842v1 [cs.CV].
7. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. – 2015. – arXiv:1512.00567v3 [cs.CV].
8. Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. – 2016 – arXiv:1602.07261v2 [cs.CV].
9. Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. – 2019 – arXiv:1905.11946v5 [cs.LG].

10. Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation // Springer International Publishing Switzerland. MICCAI 2015, Part III, LNCS 9351, pp. 234–241, 2015.
11. H. Huang et al., "UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020, pp. 1055-1059, doi: 10.1109/ICASSP40776.2020.9053405.
12. N. Enshaei, S. Ahmad and F. Naderkhani, "Automated detection of textured-surface defects using UNet-based semantic segmentation network," 2020 IEEE International Conference on Prognostics and Health Management (ICPHM), 2020, pp. 1-5, doi: 10.1109/ICPHM49022.2020.9187023.
13. H. Üzen, M. Türkoğlu and D. Hanbay, "Surface Defect Detection Using Deep U-Net Network Architectures," 2021 29th Signal Processing and Communications Applications Conference (SIU), 2021, pp. 1-4, doi: 10.1109/SIU53274.2021.9477790.
14. Sungbin Choi. Traffic map prediction using UNet based deep convolutional neural network, 2019. arXiv:1912.05288v1 [cs.LG].
15. Kaggle Severstal: Steel Defect Detection. Can You Detect and Classify Defects in Steel? 2019. Available online: https://www.kaggle.com/c/severstal-steel-defect-detection (accessed on 23 September 2023).
16. Kaggle: SD-saliency-900. Saliency detection for strip steel surface defects., 2020. Available online: https://www.kaggle.com/datasets/alex000kim/sdsaliency900 (accessed on 23 September 2023).
17. Song, G., Song, K., & Yan, Y. (2020). Saliency detection for strip steel surface defects using multiple constraints and improved texture features. Optics and Lasers in Engineering, 128, 106000. doi:10.1016/j.optlaseng.2019.106000
18. Yiting Li, Haisong Huang, Qingsheng Xie, Liguo Yao and Qipeng Chen. Research on a Surface Defect Detection Algorithm Based on MobileNet-SSD // Applied Sciences. – 2018 – 8, 1678; doi:10.3390/app8091678.

19. Xu Liyun, Li Boyu, Mi Hong, Lu Xingzhong. Improved Faster R-CNN algorithm for defect detection in powertrain assembly line // Procedia CIRP, Vol. 93. – 2020, pp. 479-484; https://doi.org/10.1016/j.procir.2020.04.031.
20. Ferguson MK, Ronay A, Lee YT, Law KH. Detection and Segmentation of Manufacturing Defects with Convolutional Neural Networks and Transfer Learning. Smart Sustain Manuf Syst. – 2018; 2:10.1520/SSMS20180033. doi:10.1520/SSMS20180033.
21. Focal loss for dense object detection / T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar // Proc. of the IEEE International Conference on Computer Vision (22-29 Oct. 2017). – 2017, 10.1109/ICCV.2017.324.
22. Keras: Deep Learning for humans. Available online: https://github.com/keras-team/keras (accessed on 23 September 2023).
23. scikit-learn. Machine Learning in Python. Available online: https://scikit-learn.org/stable/ (accessed on 23 September 2023).

АНОТАЦІЯ

У роботі розглянуто перспективне технічно-прикладне завдання виявлення дефектів на металевих поверхнях при змінному освітленні. Для розпізнавання дефектів використано глибоку згорткову нейронну мережу семантичної сегментації зображень. Побудовано нейромережеву модель, яка має архітектуру U-net з енкодером ResNet152. Спроектовану модель навчено на зображеннях металевих поверхонь з відкритих джерел.

Щоб кількісно оцінити результати розпізнавання пошкоджень нейронною мережею, використовували ряд параметрів (площа розпізнаних фрагментів, їх орієнтація тощо). Досліджено вплив освітлення поверхні на результат, який видає нейронна мережа.

Виявлено діапазони освітлення стального зразка, при якому пошкодження розпізнаються краще. Найбільші пошкодження краще розпізнавати при середньому освітленні. При освітленні більше 200 лк зростає кількість дрібних фрагментів, які хибно розпізнаються як пошкодження.

Робота містить вступ, три розділи, висновки та додаток. У вступі показано актуальність розробки та дослідження нейромережі для розпізнавання пошкоджень на зображеннях. У розділі 1 описано загальну архітектуру згорткової нейронної мережі для розпізнавання дефектів та процес її навчання. Розділ 2 містить методику дослідження результатів розпізнавання при змінному освітленні дослідного зразка. У розділі 3 приведено результати експериментів розпізнавання при змінному освітленні та їх аналіз. Додаток містить код Python-застосунку, який використано для розпізнавання пошкоджень та обчислення кількісних характеристик розпізнаних об’єктів.

Пояснювальна записка містить 28 сторінок (без титульного аркуша, посилань та додатку), 13 рисунків, 1 таблицю, перелік посилань з 23 пунктів та додаток.

Ключові слова: згорткова нейронна мережа, сегментація зображень, розпізнавання дефектів, освітленість, дефектоскопія.

ДОДАТОК

Додаток містить код Python-застосунку для розпізнавання пошкоджень за допомогою нейронної мережі.

Код класу SegmentationClient

```
import os
from enum import IntEnum, auto

import cv2
import numpy as np
import pandas as pd
from segmentation_models import get_preprocessing
from tensorflow.keras.models import load_model

from constants import MODEL_INPUT_SHAPE, PREDICTED_FOLDER
from utils.file_utils import create_files_list
from utils.img_region_properties import calc_image_regions_properties


class PatchOverlapMethod(IntEnum):
    """
    Methods of calculation of pixels for overlapped parts of patches
    """
    max = auto()
    mean = auto()


class SegmentationClient:

    def __init__(self,
                 model_path,
                 model_backbone,
                 model_optimizer,
                 model_loss,
                 model_input_shape=MODEL_INPUT_SHAPE):

        self.model_input = model_input_shape
        self.preprocess = get_preprocessing(model_backbone)

        optimizer = model_optimizer
        loss = model_loss
        self.model = load_model(model_path, compile=False)
        self.model.compile(optimizer=optimizer, loss=loss)

    def predict_standard_image(self, file_path):
        """
        Predict image that has 'standard' size defined by MODEL_INPUT_SHAPE
constant.
        :param file_path: str, Path to image
        :return: numpy array of predicted image
        """
        img = self.preprocess(cv2.imread(file_path))
        img = np.expand_dims(img, axis=0)
        res = self.model.predict(img)
        return res[0] * 255

    def predict_standard_images(self, folder_path):
        """
        Predict images that have 'standard' size (defined by MODEL_INPUT_SHAPE
```

```
constant)
        in the given folder. All predicted images are saved into separate
PREDICTED_FOLDER
        subfolder.
        :param folder_path: str, Path to folder that contains images.
        """
        if not folder_path.endswith("/"):
            folder_path = folder_path + "/"

        images_to_predict = create_files_list(folder_path, False)

        if not os.path.exists(f"{folder_path}{PREDICTED_FOLDER}/"):
            os.makedirs(f"{folder_path}{PREDICTED_FOLDER}/")

        for file_name in images_to_predict:
            cv2.imwrite(
                f"{folder_path}{PREDICTED_FOLDER}/" + file_name,
                self.predict_standard_image(folder_path + file_name)
            )

    def predict_image(self, file_path, overlap_method=PatchOverlapMethod.max):
        """
        Predict image with any (non-standard size) size
        :param file_path: str, Path to image
        :param overlap_method: PatchOverlapMethod, method of calculation of
pixels
            for overlapped parts of patches
        :return:
        """
        img = self.preprocess(cv2.imread(file_path))
        (image_height, image_width, _) = img.shape
        predicted = np.zeros((image_height, image_width, 1), np.uint8)

        (input_h, input_w) = MODEL_INPUT_SHAPE
        (shift_h, shift_w) = (input_h // 3 * 2, input_w // 3 * 2)

        x0 = 0
        pass_x = True
        while pass_x:
            x1 = x0 + input_w
            if x1 > image_width:
                x1 = image_width
                x0 = x1 - input_w
            if x1 == image_width:
                pass_x = False

            y0 = 0
            pass_y = True
            while pass_y:
                y1 = y0 + input_h
                if y1 > image_height:
                    y1 = image_height
                    y0 = y1 - input_h
                if y1 == image_height:
                    pass_y = False

                crop = np.expand_dims(img[y0:y1, x0:x1], axis=0)
                res = self.model.predict(crop)
                predicted_patch = res[0] * 255
                if overlap_method == PatchOverlapMethod.max:
                    predicted[y0:y1, x0:x1] = np.maximum(predicted[y0:y1,
x0:x1], predicted_patch)
                elif overlap_method == PatchOverlapMethod.mean:
```

```
                    predicted[y0:y1, x0:x1] = np.mean([predicted[y0:y1, x0:x1],
predicted_patch], axis=0)

                y0 = y0 + shift_h

            x0 = x0 + shift_w

        return predicted

    def predict_images(self, folder_path):
        """
        Predict images that have 'non-standard' size (defined by
MODEL_INPUT_SHAPE constant)
        in the given folder. All predicted images are saved into separate
PREDICTED_FOLDER
        subfolder.
        :param folder_path: str, Path to folder that contains images.
        """
        if not folder_path.endswith("/"):
            folder_path = folder_path + "/"

        images_to_predict = create_files_list(folder_path, False)

        if not os.path.exists(f"{folder_path}{PREDICTED_FOLDER}/"):
            os.makedirs(f"{folder_path}{PREDICTED_FOLDER}/")

        for file_name in images_to_predict:
            cv2.imwrite(
                f"{folder_path}{PREDICTED_FOLDER}/" + file_name,
                self.predict_image(folder_path + file_name)
            )

    @staticmethod
    def calc_region_props_in_folder(folder_path, binary_threshold,
result_csv_path, save_segmented_images=False):
        """
        Calculate regions properties in segmented images and save them
        into CSV file.
        :param binary_threshold: int, Threshold of Gray->Binary image transform
        :param folder_path: str, Path to folder with segmented images
        :param result_csv_path: str, Results CSV file full path
        :param save_segmented_images: bool, Save segmented images or not
        :return: Pandas DataFrame with regions properties
        """
        files = create_files_list(folder_path, include_path=False)

        df = None
        for file in files:
            file_name, file_extension = os.path.splitext(file)
            img_segmented = cv2.imread(os.path.join(folder_path, file))
            img_segmented = cv2.cvtColor(img_segmented, cv2.COLOR_BGR2GRAY)
            img_segmented[img_segmented < binary_threshold] = 0
            img_segmented[img_segmented >= binary_threshold] = 255

            if save_segmented_images:
                cv2.imwrite(os.path.join(folder_path, "thresholded",
f"{file_name}-thr{binary_threshold:03}.png"), img_segmented)

            dfi = calc_image_regions_properties(cv2.transpose(img_segmented),
file)
            df = dfi if df is None else pd.concat([df, dfi],
ignore_index=True).reset_index(drop=True)
```

```
        df.to_csv(result_csv_path, index=False)
        return df
```

Код класу DefectDetector

```
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.optimizers import SGD

from constants import *
from segmentation.segmentation_client import SegmentationClient


class DefectDetector:

    def __init__(self, model_storage_path, model_file_name, model_backbone,
model_optimizer, model_loss):
        self.model_storage = model_storage_path
        self.model_name = model_file_name
        self.model_backbone = model_backbone
        self.model_optimizer = model_optimizer
        self.model_loss = model_loss

    def __call__(self, *args, **kwargs):
        segmentation_client = SegmentationClient(
            model_path=os.path.join(self.model_storage, self.model_name),
            model_backbone=self.model_backbone,
            model_optimizer=self.model_optimizer,
            model_loss=self.model_loss
        )

        segmentation_client.predict_images(os.path.join(images_path))

        segmentation_client.calc_region_props_in_folder(
            folder_path=os.path.join(images_path, "predicted"),
            binary_threshold=BINARY_THRESHOLD,
            result_csv_path=os.path.join(images_path, "predicted", f"regions-
thr{BINARY_THRESHOLD}.csv")
        )


if __name__ == '__main__':

    print("Defects detection started")

    defect_detector = DefectDetector(
        model_storage_path=model_storage,
        model_file_name=model_name,
        model_backbone="resnet152",
        model_optimizer=SGD(),
        model_loss=binary_crossentropy
    )
    defect_detector()

    print("Defects detection finished successfully")
```

Код класу DefectAnalyzer

```
import os.path

from constants import *
from regions_analysis.threshold_prop_analyzer import ThresholdPropAnalyzer
from regions_analysis.reg_prop_analyzer import RegPropAnalyzer


class DefectAnalyzer:

    def __init__(self,
                 min_area,
                 zones,
                 threshold,
                 thresholds,
                 illumination_short_list):
        self.min_area = min_area
        self.zones = zones
        self.threshold = threshold
        self.thresholds = thresholds
        self.illumination_short_list = illumination_short_list

    def calc_param_statistics(self):
        reg_prop_analyzer = RegPropAnalyzer(
            regprop_parameters=os.path.join(regprop_path, f"regions-
thr{self.threshold}.csv"),
            min_area=self.min_area
        )
        df12 = reg_prop_analyzer.data_by_zone(self.zones)

        for param in ANALYZED_PARAMS:
            reg_prop_analyzer.build_param_by_illuminance_box_plot(
                df=df12,
                param=param,
                save_plot_path=os.path.join(ROOT_DIR, "plots")
            )
            reg_prop_analyzer.build_param_by_illuminance_scatter_plot(
                df=df12,
                param=param,
                save_plot_path=os.path.join(ROOT_DIR, "plots")
            )

    def analyze_thresholds(self):
        threshold_analyzer = ThresholdPropAnalyzer(
            files_path=regprop_path,
            files_list=[f"regions-thr{threshold:03}.csv" for threshold in
self.thresholds],
            min_area=600
        )
        for illumination in self.illumination_short_list:
            threshold_analyzer.graph_area_by_threshold(
                illumination=illumination,
                show_plot=True,
                save_plot_path=os.path.join(ROOT_DIR, "plots")
            )

            threshold_analyzer.graph_orientation_by_threshold(
                illumination=illumination,
                show_plot=True,
                save_plot_path=os.path.join(ROOT_DIR, "plots")
            )
```

```
        threshold_analyzer.build_all_param_by_threshold_box_plot(
            save_plot_path=os.path.join(ROOT_DIR, "plots")
        )


if __name__ == '__main__':

    print("Images analysis started")

    defect_analyzer = DefectAnalyzer(
        min_area=600,
        zones=[1, 2],
        threshold=200,
        thresholds=[51, 102, 128, 175, 200],
        illumination_short_list=[2, 100, 500, 800]
    )
    defect_analyzer.calc_param_statistics()
    defect_analyzer.analyze_thresholds()

    print("Images analysis finished")
```

Код класу RegPropAnalyzer

```
import os.path

import matplotlib.pyplot as plt
import pandas as pd

from constants import *


class RegPropAnalyzer:

    def __init__(self, regprop_parameters, min_area=0):
        self.regprop_df = pd.read_csv(regprop_parameters, delimiter=",")

        if min_area > 0:
            self.regprop_df = self.regprop_df[self.regprop_df[field_filled_area]
> min_area]

        self.regprop_df[field_zone] =
self.regprop_df[field_image_name].str.slice(8, 9).astype(int)

        self.regprop_df[field_illumination_str] =
self.regprop_df[field_image_name].str.slice(2, 5)
        self.regprop_df[field_illumination] = self.regprop_df.apply(lambda row:
self._illumination_range(row), axis=1)

        self.illumination_list =
sorted(set(self.regprop_df[field_illumination].to_list()))

    def _illumination_range(self, row):
        illum = int(row[field_illumination_str])
        if illum < 10:
            illumination = 2
        elif illum < 70:
            illumination = round(illum, -1)
        else:
            illumination = round(illum, -2)

        return illumination
```

```
    def data_by_zone(self, zones):
        return self.regprop_df[self.regprop_df[field_zone].isin(zones)]

    def build_param_by_illuminance_box_plot(self, df, param, show_plot=True,
save_plot_path=""):
        illuminations = sorted(set(df[field_illumination].to_list()))
        params = [df[df[field_illumination] == ill][param].to_list() for ill in
illuminations]

        plt.rc('axes', edgecolor=color_frame)
        fig1, ax1 = plt.subplots()
        ax1.set_title(f"{param} by Illumination Statistics")
        ax1.set_xlabel(field_illumination)
        ax1.set_ylabel(param)
        boxpl = ax1.boxplot(
            params,
            vert=True,
            patch_artist=True,
            labels=illuminations
        )

        for item in ['boxes', 'whiskers', 'fliers', 'caps']:
            plt.setp(boxpl[str(item)], color=color_bar)
        plt.setp(boxpl["medians"], color=color_median)
        plt.setp(boxpl["boxes"], facecolor=color_bar)
        plt.setp(boxpl["fliers"], markeredgecolor=color_bar)

        if show_plot:
            plt.show()
        if save_plot_path:
            fig1.savefig(os.path.join(save_plot_path, f"{param}-by-Illum-
Box.png"))

    def build_param_by_illuminance_scatter_plot(self, df, param, show_plot=True,
save_plot_path=""):
        plt.rc('axes', edgecolor=color_frame)
        fig1, ax1 = plt.subplots()
        ax1.set_title(f"{param} Values")
        ax1.set_xlabel(field_illumination)
        ax1.set_ylabel(param)
        sctpl = ax1.scatter(
            x=df[field_illumination],
            y=df[param],
            c=color_bar,
        )

        if show_plot:
            plt.show()
        if save_plot_path:
            fig1.savefig(os.path.join(save_plot_path, f"{param}-by-Illum-
Scatter.png"), bbox_inches='tight')

    def graph_area_by_threshold(self, illumination, show_plot=True,
save_plot_path=""):
        areas = [df[df[field_illumination] ==
illumination][field_filled_area].sum() / 1000 for df in self._df12_list]

        fig, ax = plt.subplots()
        ax.plot(self._thresholds, areas)

        ax.set(
            xlabel="threshold",
```

```
            ylabel="Filled Area, x1000 pix",
            title=f"Filled Area by Threshold (Illumination {illumination})"
        )
        ax.grid()

        if show_plot:
            plt.show()
        if save_plot_path:
            fig.savefig(os.path.join(
                save_plot_path,
                f"Filled-Area-by-Threshold-illum-{illumination:03}.png")
            )
```

Код класу ThresholdPropAnalyzer

```
import math
import os.path

import matplotlib.pyplot as plt

from constants import *
from regions_analysis.reg_prop_analyzer import RegPropAnalyzer


class ThresholdPropAnalyzer:

    def __init__(self, files_path, files_list, min_area=0):

        self._reg_analyzers = [
            RegPropAnalyzer(
                regprop_parameters=os.path.join(files_path, file_name),
                min_area=min_area
            )
            for file_name in files_list
        ]
        self._thresholds = sorted([int(file_name[11:14]) for file_name in
files_list])

        self._df12_list = [analyzer.data_by_zone([1, 2]) for analyzer in
self._reg_analyzers]

    def graph_area_by_threshold(self, illumination, show_plot=True,
save_plot_path=""):
        areas = [df[df[field_illumination] ==
illumination][field_filled_area].sum() / 1000 for df in self._df12_list]

        fig, ax = plt.subplots()
        ax.plot(self._thresholds, areas)

        ax.set(
            xlabel="threshold",
            ylabel="Filled Area, x1000 pix",
            title=f"Filled Area by Threshold (Illumination {illumination})"
        )
        ax.grid()

        if show_plot:
            plt.show()
        if save_plot_path:
            fig.savefig(os.path.join(
                save_plot_path,
```

```
                f"Filled-Area-by-Threshold-illum-{illumination:03}.png")
            )

    def graph_orientation_by_threshold(self, illumination, show_plot=True,
save_plot_path=""):
        orientations = [df[df[field_illumination] ==
illumination][field_orientation].mean() for df in self._df12_list]

        fig, ax = plt.subplots()
        ax.plot(self._thresholds, orientations)

        ax.set(
            xlabel="threshold",
            ylabel="Orientation, rad",
            title=f"Mean Orientation by Threshold (Illumination {illumination})"
        )
        ax.grid()
        ax.set_ylim(-math.pi / 2, math.pi / 2)

        if show_plot:
            plt.show()
        if save_plot_path:
            fig.savefig(os.path.join(
                save_plot_path,
                f"Orientation-by-Threshold-illum-{illumination:03}.png")
            )

    def build_param_by_threshold_box_plot(self, illumination, param,
show_plot=True, save_plot_path=""):
        params = [df[df[field_illumination] == illumination][param].to_list()
for df in self._df12_list]

        plt.rc('axes', edgecolor=color_frame)
        fig1, ax1 = plt.subplots()
        ax1.set_title(f"{param} by Threshold Statistics (Illumination
{illumination})")
        ax1.set_xlabel("Threshold")
        ax1.set_ylabel(param)
        boxpl = ax1.boxplot(
            params,
            vert=True,
            patch_artist=True,
            labels=self._thresholds
        )

        for item in ['boxes', 'whiskers', 'fliers', 'caps']:
            plt.setp(boxpl[str(item)], color=color_bar)
        plt.setp(boxpl["medians"], color=color_median)
        plt.setp(boxpl["boxes"], facecolor=color_bar)
        plt.setp(boxpl["fliers"], markeredgecolor=color_bar)

        if show_plot:
            plt.show()
        if save_plot_path:
            fig1.savefig(os.path.join(save_plot_path, f"{param}-by-Thresh-Illum-
{illumination:03}-Box.png"))

    def build_all_param_by_threshold_box_plot(self, save_plot_path):
        illuminations =
sorted(set(self._df12_list[0][field_illumination].to_list()))

        for param in ANALYZED_PARAMS:
            for illumination in illuminations:
```

```
self.build_param_by_threshold_box_plot(
    illumination=illumination,
    param=param,
    show_plot=True,
    save_plot_path=save_plot_path
)
```