Кафедра *програмного забезпечення*

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської кваліфікаційної роботи на тему:

Виявлення дефектів на поверхнях матеріалів за розподіленими та інваріантними ознаками інтенсивності зображень (Detection of surface defects in materials by distributed and invariant features of image intensity).

Студент групи ПЗМ-21, Віпшовський Ю.А.

Керівник проекту _____ (*Мельник Р.А.*)
(підпис)

Консультанти _____ (_____)
(підпис)

_____ (_____)
(підпис)

_____ (_____)

_____ (_____)

_____

Завідувач кафедри *Федасюк Д.В.*
(підпис)

“ ____ ” _____ 20 *25* р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут Комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення
Спеціальність F2 «Інженерія програмного забезпечення»

"ЗАТВЕРДЖУЮ"
Завідувач
кафедри _____
Федасюк Д.В.
«____» _____ 202__р.

ЗАВДАННЯ

на кваліфікаційну роботу (проект) студента групи ПЗМ-21 ОКР магістр
Віпшовського Юрія Андрійовича
(прізвище, ім'я, по-батькові)

1. Тема роботи (проекту) Виявлення дефектів на поверхнях матеріалів за розподіленими та інваріантними ознаками інтенсивності зображень.
(у разі виконання комплексної роботи в дужках вказується "комплексна робота(проект)")

затверджена наказом по університету від « 04 » квітня 2025 р. № 1264-4-06

2. Термін подання студентом закінченої роботи (проекту) 16 травня 2025р.

3. Вхідні дані для роботи (проекту) специфікація вимог, огляд і аналіз наукових статей, інтернет ресурси з зразками матеріалів та прикладами вирішення задачі програмними засобами, можливості відомих нейронних мереж для застосування при класифікації, огляд існуючих методів і алгоритмів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити) аналіз наявних рішень, постановка завдання, розробка алгоритму, специфікація вимог, реалізація програмного прототипу, тестування, дослідження і оцінка ефективності та швидкодії системи, висновки та перспективи розвитку

5. Перелік графічного матеріалу зображення дефектів металів, зображення кумулятивних гістограм, зображення кумулятивних відображень, знімки екрану із роботою програмного застосунку

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) середовище програмування Visual Studio, C#, Windows Forms, редактор коду Visual Studio Code, Python, Photoshop

7. Консультування роботи (проекту), із зазначенням розділів роботи

Розділ	Консультант	Завдання видав		Завдання прийняв	
		підпис	дата	підпис	дата

8. Дата, коли видано завдання 15.11.2023р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1.	Пошук наукових джерел із схожою тематикою	15.11.2023	Виконано
2.	Аналіз та дослідження вибраних джерел	30.11.2023	Виконано
3.	Написання аналітичного розділу роботи	31.12.2023	Виконано
4.	Дослідження теоретичних основ роботи алгоритму	28.02.2024	Виконано
5.	Пошук, обґрунтування та виведення формул	30.04.2024	Виконано
6.	Написання теоретичного розділу роботи	31.05.2024	Виконано
7.	Розробка алгоритму виявлення дефектів	30.07.2024	Виконано
8.	Розробка програмного забезпечення	31.08.2024	Виконано
9.	Дослідження швидкодії роботи, методів розпаралелення та пришвидшення алгоритму	30.11.2024	Виконано
10.	Дослідження точності роботи алгоритму	30.03.2024	Виконано
11.	Написання 3 та 4 розділів дипломної роботи	30.04.2024	Виконано
12.	Написання висновків та завершення дипломної роботи	16.05.2024	Виконано

Студент _____
(підпис)

Керівник _____
(підпис)

Видавництво Львівської політехніки. Наклад 5000. Зам. 180288. 2018

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена актуальному завданню автоматизованого виявлення дефектів на поверхнях металевих матеріалів. У роботі проведено аналіз існуючих підходів та методів комп’ютерного зору для виявлення дефектів, визначено їх переваги та недоліки. Запропоновано новий підхід, що ґрунтується на статистичних методах аналізу зображень, а саме використанні кумулятивних гістограм.

Розроблено прототип програмної системи на мові програмування C# з використанням технології Windows Forms, яка дозволяє ефективно реалізувати запропонований метод та здійснювати його експериментальну перевірку. Проведено детальне дослідження ефективності роботи запропонованого алгоритму на зображеннях різних розмірів та складності, виявлено оптимальні параметри його роботи.

Результати тестування показали високу точність виявлення дефектів (до 98% на зображеннях малого розміру та до 86% на великих зображеннях), а також хорошу здатність алгоритму до масштабування завдяки ефективному розпаралеленню. Отримані результати підтверджують перспективність застосування розробленого підходу у промисловості, що дозволить підвищити якість контролю матеріалів та знизити витрати, пов’язані з людським фактором.

Ключові слова: виявлення дефектів, аналіз поверхні матеріалів, комп'ютерний зір, кумулятивні гістограми, статистичний аналіз, автоматизований контроль якості.

ABSTRACT

The master's thesis is dedicated to the pressing issue of automated defect detection on metal surfaces. The work undertakes a comprehensive analysis of existing approaches and methods of computer vision for defect detection. It identifies the advantages and disadvantages of these methods and provides a detailed evaluation of their effectiveness. A novel approach is proposed, which is based on statistical image analysis methods, specifically the utilization of cumulative histograms.

A prototype of a software system in the C# programming language using Windows Forms technology has been developed, which allows for the effective implementation of the proposed method and its experimental verification. A comprehensive investigation was conducted into the efficacy of the proposed algorithm on images of varying dimensions and intricacy, leading to the identification of its optimal operational parameters.

The experimental findings demonstrated the efficacy of the defect detection algorithm, with a high accuracy of up to 98% for small images and up to 86% for large images. The scalability of the algorithm was also examined, and it was found to be efficient due to its parallelization. The outcomes substantiate the viability of implementing the developed approach in industrial settings, which is poised to enhance the efficacy of material inspection and curtail expenditures stemming from the human factor.

Keywords: defect detection, material surface analysis, computer vision, cumulative histograms, statistical analysis, automated quality control.

ЗМІСТ

ВСТУП

Виявлення дефектів на поверхні металевих матеріалів є одним із ключових аспектів процесу забезпечення якості продукції в сучасній промисловості. Дефекти, такі як тріщини, подряпини, порожнини та інші нерівності, можуть негативно впливати на експлуатаційні властивості виробів, що є причиною зниження їхньої міцності, надійності та довговічності. Наявність таких дефектів у матеріалах, які використовуються в критичних умовах, може призвести передчасних поломок техніки, що спричиняє не лише економічні втрати, а й потенційні загрози для безпеки експлуатації.

У зв'язку з цим контроль якості матеріалів на різних етапах виробництва є обов'язковою вимогою, особливо у високотехнологічних галузях, де надійність продукції є вирішальним фактором. Автоматизація цього процесу набуває особливого значення в умовах великих обсягів виробництва, коли ручний контроль стає неефективним і схильним до помилок. Автоматизовані системи виявлення дефектів дозволяють здійснювати швидкий та об'єктивний аналіз стану матеріалів, знижуючи залежність від людського фактору, а також забезпечують більш високу точність і повторюваність результатів.

Сучасні методи виявлення дефектів базуються на використанні різних технологій, серед яких, після стрімкого розвитку штучного інтелекту, значне місце займають методи, що ґрунтуються на обробці зображень. Технології комп'ютерного зору дають можливість аналізувати поверхню матеріалів з високою роздільною здатністю, що дозволяє виявляти навіть мікроскопічні дефекти, які можуть бути непомітними для людського ока. Завдяки таким підходам можна не лише своєчасно виявити дефекти, але й класифікувати їх за типом, розміром та локалізацією, що сприяє оптимізації процесів виробництва та подальшої обробки матеріалів.

Незважаючи на активне впровадження штучного інтелекту в процес виявлення дефектів, більшість існуючих методів стикаються з проблемами

необхідності великої кількості навчальних даних, високими вимогами до обчислювальних ресурсів та складністю адаптації до нових типів дефектів або умов виробництва. Це зумовлює актуальність розробки альтернативних методів, які є менш залежними від об'єму навчальної вибірки та більш гнучкими у застосуванні на виробництві.

Саме тому важливим є пошук і впровадження статистичних підходів до аналізу зображень, що можуть працювати без тривалого попереднього навчання і водночас забезпечувати високу точність і стабільність результатів. Одним із перспективних напрямів у цьому контексті є використання кумулятивних гістограм, які дозволяють ефективно аналізувати зображення навіть за умов обмежених обчислювальних ресурсів, забезпечуючи високу швидкодію та можливість масштабування.

Отже, автоматизовані системи контролю якості з використанням комп'ютерного зору є важливим інструментом для підвищення ефективності виробництва та забезпечення відповідності продукції високим стандартам якості. Використання цих технологій дозволяє мінімізувати ризики появи дефектної продукції на ринку, забезпечуючи тим самим як економічну вигоду, так і підвищення рівня безпеки кінцевих виробів.

РОЗДІЛ 1. ВИЗНАЧЕННЯ ДЕФЕКТІВ НА МАТЕРІАЛАХ ТА ТЕКСТУРАХ

1.1 Опис предметної області визначення дефектів на матеріалах та текстурах

В сучасному світі, де високотехнологічні матеріали використовуються в різних галузях промисловості, пошук та аналіз дефектів на матеріалах та текстурах є питанням критичної важливості. Виробничі процеси піддаються впливу різноманітних умов і навантажень, що може призводити до появи дефектів у структурах матеріалів.

Пошук і ефективний аналіз цих дефектів має вирішальне значення для забезпечення якості та безпеки кінцевих виробів. Історія розвитку матеріалознавства та інженерії свідчить про те, що наявність навіть найменших дефектів може унеможливити надійне функціонування конструкцій та обладнання. Виявлення дефектів у матеріалах допомагає виявити причини їх виникнення, проаналізувати процес виробництва та внести необхідні зміни, оптимізувати його. Процес виявлення дефектів можна оптимізувати із метою покращення ефективності виявлення дефектів [1].

У сучасній матеріалознавчій дисципліні термін "дефект" використовується для опису різних несправностей, аномалій та недоліків, які можуть виникати у внутрішній структурі або на поверхні матеріалів. Дефекти можуть бути наслідком виробничих процесів, взаємодії з навколишнім середовищем або внаслідок несприятливих умов експлуатації.

Дефекти можна класифікувати як внутрішні, що виникають всередині матеріалу, та поверхневі, що розташовані на його зовнішній стороні. Внутрішні дефекти можуть включати в себе порожнистість, включення, дислокації та інші аномалії у кристалічній структурі, тоді як поверхневі дефекти можуть бути представлені тріщинками, подряпинами, корозійними плямами тощо.

Визначення дефектів важливе, адже їхня наявність впливає на експлуатаційну безпеку та довговічність матеріалів. Дефекти можуть служити початковою точкою для початку руйнування матеріалів під впливом навантажень, що може негативно вплинути на безпеку конструкцій та пристроїв. Тому, виявлення та аналіз дефектів стає важливим завданням для забезпечення якості та надійності виготовлених продуктів.

Також наявність дефектів на кінцевому продукті, зменшує його вартість, а якщо дефекти бури присутні на матеріалі то вони перемістять і на кінцевий продукт, тому важливо використовувати якісні матеріали. Зовнішній вигляд критично впливає на торговельну вартість продуктів і купівельну поведінку споживачів. Дефекти, які зазвичай зустрічаються, включають подряпини, ямки та відшарування фарби. Це часто призводить до повернення продажів та/або скарг споживачів, що призводить до великих економічних втрат і шкоди корпоративній репутації. Тому виявлення граничних дефектів стало важливим етапом обробки у виробництві та контролі якості [2,3].

Існує кілька способів визначення наявності дефектів різноманітних матеріалів, ці способи залежать від типу матеріалу (деревина, пластик, метал) та його властивостей. Наприклад якість металу можна перевірити такими методами [4]:

1. Візуальний контроль – візуальна перевірка зовнішнього вигляду металу на наявність тріщин, сколів, вм’ятин та інших.

2. Візуально-оптичний метод – метод перевірки з допомогою різноманітних оптичних систем у тому числі камер, для візуального контролю.

3. Магнітно-порошковий контроль – застосування магнітного поля для виявлення тріщин у металах.

4. Ультразвуковий контроль – застосування ультразвукових хвиль для виявлення дефектів, таких як тріщини та вміст порожнин.

5. Рентгенівський контроль – використання рентгенівських променів для виявлення внутрішніх дефектів, таких як тріщини та порожнини.

6. Електромагнітний контроль – використання електромагнітних полів для виявлення тріщин.

7. Контроль твердості – вимірювання твердості поверхні для виявлення змін в структурі та можливих дефектів.

8. Хімічний аналіз – використання хімічних тестів для визначення складу матеріалу та виявлення можливих дефектів, таких як корозія та окислення.

9. Теплові методи, такі як термографія, що використовують інфрачервоне випромінювання для виявлення дефектів та аномалій в температурі [5].

Деякі із цих методів підходять для аналізу не тільки металів а й інших матеріалів. Найпростіший у реалізації, та достатньо ефективний метод виявлення дефектів це візуально-оптичний. Цей метод універсальний для усіх матеріалів та дозволяє виявити наявність дефектів на поверхні, та відбракувати матеріал. Візуально-оптичний контроль може проводитись як людиною особисто, та і з допомогою інтелектуальних систем, які отримують зображення із попередньо налаштованих камер та інших датчиків, та аналізують його з допомогою різноманітних алгоритмів.

1.2 Існуючі підходи виявлення дефектів матеріалів

Для перевірки наявності дефектів на матеріалі з допомогою візуально-оптичного методу необхідно виконати такі основні етапи:

1. Отримати якісне зображення матеріалу
2. Виявити чи присутні на матеріалі дефекти чи ні

Зображення матеріалу можна отримати з допомогою камери, або декількох камер, у залежності від розміру матеріалу.

У [6] запропоновано використовувати 2 камери для аналізу поверхні шаф розміром 0.6м х 1.6м, та систему освітлення з двох смугових джерел

світла. Після отримання фотографія відправляється на комп’ютер для подальшої обробки та пошуку дефектів. Система використовує фільтр згладжування Гауса та оператор градієнта Соболя, щоб пом’якшити вплив нерівномірного освітлення. Для відділення країв від фону використано алгоритм порогової сегментації К3М. Так як зображення шафи є дуже великим його розбивають на велику кількість малих блоків та аналізують їх окремо.

Ефективність даного методу находиться на рівні 88%, у той час як інші методи які були перевірені у дослідженні на аналогічних умовах показали точність у діапазоні від 46% до 58%.

Дана робота є дуже корисною для нашої праці, адже тут описані та порівняні різні методи статистичної оброки зображень при пошуку дефектів. Із недоліків можна відзначити значний акцент на межах а не на цілому зображенні.

У [7] описано різноманітні методи статистичного аналізу, а саме сегментації зображень для ідентифікації біологічних мікрооб’єктів. Дані методи доволі швидко працюють і не потребують навчальних даних, що значно спрощує розробку системи.

Однак дані методи розраховані на пошук великих кластерів, вони добре справлятимуться із великими дефектами на матеріалі, але якщо буде багато невеликих дефектів, то сегментація не дасть потрібного результату і може сприйняти матеріал за хороший.

У [8] проаналізовано декілька методів кластеризації зображення, для швидкого пошуку даних. Описані алгоритми достатньо швидко розбивають зображення на велику кількість кластерів та показують хорошу точність.

Недоліком даної системи є її великий час роботи при розбитті зображення на 2 кластери – 16 секунд, що унеможливлює її використання для пошуку дефектів на зображеннях.

В усіх інших роботах на схожі теми було використано та модифіковано різні нейронні мережі.

У [9] запропоновано нову модель згорткової нейронної мережі для пошуку дефектів на зображеннях кремнієвих кристалів. Система працює із зображеннями малого розширення 28x28, та запропоновано використовувати GAP (Global Average Pooling) що значно зменшує кількість параметрів мережі, і в свою чергу полегшує її навчання.

Дана система працює із малими кристалами чіпів, розміри яких всього декілька міліметрів, тому розширення зображення 28x28 цілком достатньо для аналізу кремнієвих кристалів, але при аналізі великих площ матеріалів працювати із таким малим розширенням буде неможливо. Також розбивати зображення на такі малі картинки буде теж доволі проблематично, тому хоч і система доволі проста та швидка, але для аналізу великих площ матеріалів не підходить.

У [10] описана система для вирішення проблеми пошуку дефектів на поверхні металевого прокату. Було використано нейронну мережу U-Net для виділення контурів дефектів на зображенні. Точність виділення досягає 60-70 процентів, але разом із цим якщо на зображенні присутній дефект, то система із ймовірністю близькою до 99% виявить наявність дефекту на предметі. Точність визначення положення дефекту не є суттєва при аналізі матеріалу, адже якщо на матеріалі присутні якісь зовнішні дефекти, то можуть бути і внутрішні, отже матеріал не варто використовувати.

Недоліками даної системи як і усіх наступних на базі різних нейронних мереж є необхідність великої кількості навчальних зображень із усіма можливими типами дефектів, адже мережі необхідно під час навчання отримати характеристики кожного типу дефекту, щоб правильно класифікувати його. Неправильна класифікація дефекту може призвести до того що зображення із дефектом класифікується як хороше, що неприпустимо.

У [11] запропоновано використовувати двох потокову згорткову нейронну мережу для аналізу кольорового та градієнтного зображення

алюмінієвих профілів для пошуку на них дефектів. Система перетворює зображення до розширення 227x227, та аналізує його.

Дана система у результаті експериментів показала точність у приблизно 95% при класифікації різних дефектів, але при визначенні наявності дефекту на зображенні працює дуже добре. Недоліками даної системи є необхідність працювати із квадратними зображеннями з розширенням 227x227, що є незручно особливо при роботі із профілями, адже їх форма дуже витягнута, а отже необхідно зробити багато фотографій одного профіля для аналізу. Також не сказано у роботі скільки часу тривало навчання цієї мережі, та який час аналізу одного зображення. Враховуючи те що мережа працює із доволі великими зображеннями та містить у собі 2 нейронні мережі, одна із яких аналізує кольорові зображення, можна зробити висновок що кількість параметрів у мережі дуже велика а отже і час навчання та опрацювання також є достатньо значною.

У [12] створений прототип для перевірки наявності дефектів на металевих пластинах. У системі використовується достатньо проста нейронна мережа, натренована на малій кількості зображень.

Точність визначення наявності дефектів зазначено на рівні 99%, що є дуже хорошим результатом, який досягається завдяки добре спроектованій системі освітлення, яка рівномірно підсвічує досліджуваний зразок. Дана система не була перевірена на великій кількості реальних прикладів, але очікуються також хороші результати.

У загальному цю систему можна використовувати як хороший приклад для порівняння, але досліджувані зразки були малого розміру, тому освітили їх рівномірно було не складно. Для роботи із великими пластинами дана система не підходить, і її необхідно буде модифікувати.

У [13] розроблено класифікатор дефектів шкіри з допомогою трансформаторів зору. Дана система аналізує кольорові зображення з розширенням 224x224, які розбиваються на малі зображення з розширенням

14x14 та аналізуються окремо одне від одного, після цього модель збирає їх та фіксує глобальні залежності між зображеннями.

Дана система була натренована на 3600 зображеннях усього за 3 хвилини, та досягла точності у 93% при часі обробці одного зображення в 80 Мілі секунд, що дуже хороший результат враховуючи те що система працює із кольоровими зображеннями. Також сильною стороною системи є те що ніякої додаткової обробки зображення окрім зміни його розширення не потрібно.

У [14] запропоновано новий метод виявлення дефектів на поверхні сталі з допомогою AGCN – поєднання R-CNN та механізмів уваги і графових згорткових нейронних мереж. Система складається із 4 частин – мережа виділення характерних ознак (VGG16 з ядром 3x3 та п’ятьма згортковими шарами), модуля RPN (виділяє позицію із можливим положенням дефектів), модуля RoI (стандартизує розміри вхідного та вихідного зображення), та RCNN (встановлює наявність дефекту, та його тип). Потім з допомогою повно зв’язного графа встановлюється кореляція між дефектами, а для посилення зв’язка використовуються механізми уваги.

Для навчання мережі було використано 4000 зображень із дефектами. Мережа навчалась протягом 100 епох, та всього із 8 зображеннями на епоху, у результаті досягнута точність сегментації у 85.8%, що є достатньо хорошим результатом, адже точність виявлення чи дефект присутній чи його немає буде вищою.

У [15] розроблено систему для аналізу наявності дефектів на рідкокристалічному екрані. Розроблено систему освітлення, та отримання зображень, яка отримує зображення великого розширення 2330x1750 та з глибиною кольору 24 біта, на яких видно малі дефекти плівки.

Для виділення контурів плівки запропоновано власний метод, який спочатку використовує згладжуючий фільтр, а потім алгоритм автоматичного порогу Оцу. Для виявлення дефектів запропоновано спочатку збільшити контраст між дефектами та плівкою з допомогою

аналізу локалізованої енергії, а для точного визначення місця знаходження дефекту використано метод крос-проекції. Для класифікації дефектів їх поділяють на яскраві та темні та використовують детектор Кірша.

При проведенні експериментів було досягнуто рівня виявлення дефектів 100% для відбивачів і LGP і досягли рівня 98,7% для листів дифузора. Причиною нижчого рівня виявлення аркушів дифузора є наявність п’яти аркушів низької якості. Середній час виявлення дефекту системою 6 секунд, що у більш ніж 2 рази швидше ніж при ручному виявленні дефектів.

Дана система містить цікаві та ефективні рішення які можна запровадити при реалізації нашого проекту. Та показує хороші результати при аналізі плівок. Також розроблено хорошу систему освітлення, але її можна використовувати тільки для виявлення дефектів на плівках, так як тут є фонове підсвічення знизу плівки.

У [16] представлено загальний огляд сучасних підходів до виявлення дефектів поверхонь промислових компонентів з використанням методів комп’ютерного зору. Автори розглядають різноманітні алгоритми та мережі, акцентуючи увагу на їхній практичній реалізації та ефективності, що є дуже корисним для порівняння існуючих алгоритмів із запропонованим у нашій роботі.

У [17] автори запропонували вдосконалений підхід на основі архітектури YOLOv5 для виявлення дефектів на металевих поверхнях. Проведені експерименти показали високий рівень точності та швидкості виявлення дефектів. Цей підхід є цікавим завдяки своїй простоті налаштування та можливості ефективної роботи в реальному часі.

Робота [18] пропонує новий бенчмарк та спеціалізовану нейронну мережу для виявлення дефектів металевих поверхонь. Дослідники акцентують увагу на тому, що наявні набори даних недостатньо репрезентативні, тому створили власний бенчмарк, який забезпечує більш

точне оцінювання алгоритмів, їх результати та роботу доцільно використовувати для дослідження власного алгоритму.

У роботі [19] було запропоновано мережу FOHR Net, спрямовану на виявлення дефектів металевих поверхонь у реальному часі з високою точністю. Запропонована мережева архітектура включає модуль багаторівневого вирівнювання ознак (MFAM) і модуль двох гілкової реорганізації ознак (DFRM), які сприяють якісному представленню складних дефектів навіть за умов їх перекриття або значної схожості.

Автори дослідження підкреслюють перевагу запропонованого підходу в ефективності та точності, що дозволяє застосовувати мережу безпосередньо на виробництві у режимі реального часу, не втрачаючи високої точності результатів.

До недоліків можна віднести необхідність ретельного налаштування та тренування мережі, що вимагає значних обчислювальних ресурсів.

У статті [20] автори вдосконалили популярну мережу YOLOv5 для виявлення дефектів на сталевих поверхнях. Основними інноваціями є інтеграція деформованих згорткових шарів, механізму уваги CBAM для виділення важливих ознак, а також заміна функції втрат на Focal EIOU для точнішої локалізації дефектів. Додатково, автори застосували алгоритм кластеризації K-means для адаптивного визначення параметрів.

Проведені експерименти на стандартному наборі даних NEU-DET показали суттєве підвищення точності (до 78.8%), що на 4.3% перевищує базову реалізацію YOLOv5. Цей підхід демонструє ефективність у складних сценаріях та значні переваги при роботі з дефектами складної форми та різних розмірів.

Проте, як і більшість нейронних мереж, цей метод вимагає значних ресурсів для тренування і точного налаштування параметрів моделі.

1.3 Мета, задача та прогнозована новизна результатів дослідження.

На основі проведеного аналізу можна зробити висновок, що мета роботи полягає в спрощенні процесу створення систем для аналізу наявності дефектів, збільшення швидкості роботи з допомогою використання методів статистичного аналізу та обробки зображень.

Для досягнення мети необхідно вирішити такі задачі:

1. Дослідити існуючі методи та алгоритми пошуку та класифікації дефектів на матеріалах та текстурах.
2. Створити систему із статистичних методів для обробки зображення, та розробити ефективну модель для виявлення дефектів із обробленого зображення.
3. Розробити програму для визначення наявності дефектів на зображеннях.

Об’єкт дослідження: процес візуально-оптичного виявлення дефектів матеріалів та текстур.

Предмет дослідження: методи та алгоритми автоматизованого виявлення дефектів матеріалів та текстур з використанням статистичного аналізу зображень.

Наукова новизна: поєднання методів статистичного аналізу зображень із допомогою кумулятивних гістограм та алгоритмів визначення із бінарних зображень дефектів матеріалів та текстур.

1.4 Висновки

У цьому розділі показано актуальність дослідження статистичних методів для визначення дефектів на поверхнях матеріалів. Процес визначення дефектів містить ряд проблем які необхідно вирішити, зокрема:

- Створення системи рівномірного освітлення елементів
- Вибір алгоритмів для перед обробки зображень, щоб зменшити вплив нерівномірного освітлення

- Розробка та дослідження алгоритму пошуку дефектів з допомогою кумулятивних гістограм
- Розробка застосунку для аналізу зображень поверхонь матеріалів та текстур на наявність дефектів

Провівши аналіз літератури, та розроблених підходів для вирішення цих проблем було виявлено низку недоліків:

- Необхідно багато навчальних даних для нейронних мереж
- Виявлення дефектів матеріалів на краях елементів дуже проблематичне при використанні нейронних мереж
- Використання нейронних мереж при аналізі пікселів оригінального зображення дає різноманітні похибки, при виникненні дефектів, яких не було у навчальному наборі

Ці недоліки показують, що проблема, яка розглядається у цій роботі, є актуальною і потребує вирішення. Розробка статистичних методів аналізу зображення та подальша обробка не вхідного зображення, а зображення отриманого після аналізу кумулятивними гістограмами, має вирішити дані проблеми, та полегшити процес виявлення дефектів на матеріалах та текстурах.

РОЗДІЛ 2. ТЕОРИТИЧНА ОСНОВА ДЛЯ ВИЯВЛЕННЯ ДЕФЕКТІВ НА МАТЕРІАЛАХ

2.1 Створення відображень кумулятивних гістограм

Для комп’ютера зображення — це набір байтів, які він інтерпретує у пікселі, що у подальшому засвічують із певною інтенсивністю окремі точки на екрані монітора. У результаті цього комп’ютер сприймає зображення не як цілісний об'єкт, а лише як послідовність окремих пікселів. Кожен піксель опрацьовується окремо, без врахування контексту або сусідніх елементів зображення. Таким чином, при обробці зображень комп’ютеру вкрай складно уловити різноманітні взаємозв’язки та структури, які характеризують зображення у цілому. Адже навіть якщо ретельно аналізувати кожен окремий піксель, неможливо визначити, чи належить він до дефектної ділянки матеріалу, чи, навпаки, є частиною якісної, неушкодженої поверхні.

Враховуючи це, основним завданням будь-якого методу виявлення дефектів за допомогою аналізу зображень є встановлення та аналіз взаємозв’язків між пікселями, що дозволяють розглядати зображення як єдину структуру, а не окремі незалежні точки. Для розв'язання цієї задачі ми пропонуємо використати статистичні методи обробки зображень, які дозволяють ефективно знаходити та оцінювати такі взаємозв’язки.

Основна ідея запропонованого алгоритму полягає в аналізі зв’язків між пікселями кожного окремого стовпця або рядка зображення. Для цього виконується побудова та подальший аналіз кумулятивних гістограм, які дозволяють встановити кількісні співвідношення між пікселями різних рівнів інтенсивності.

Гістограма — це графік, який показує кількість пікселів кожної окремої інтенсивності на зображенні (рис. 2.1, зелений графік).

Кумулятивна гістограма - це графік, що відображає співвідношення накопиченої кількості пікселів усіх інтенсивностей, менших або рівних заданій, до загальної кількості пікселів усього зображення (рис. 2.1, синій

графік). Використання таких гістограм дозволяє оцінювати інтенсивність пікселів не окремо, а у контексті всього зображення або його значної частини, що значно покращує якість і точність аналізу дефектів.

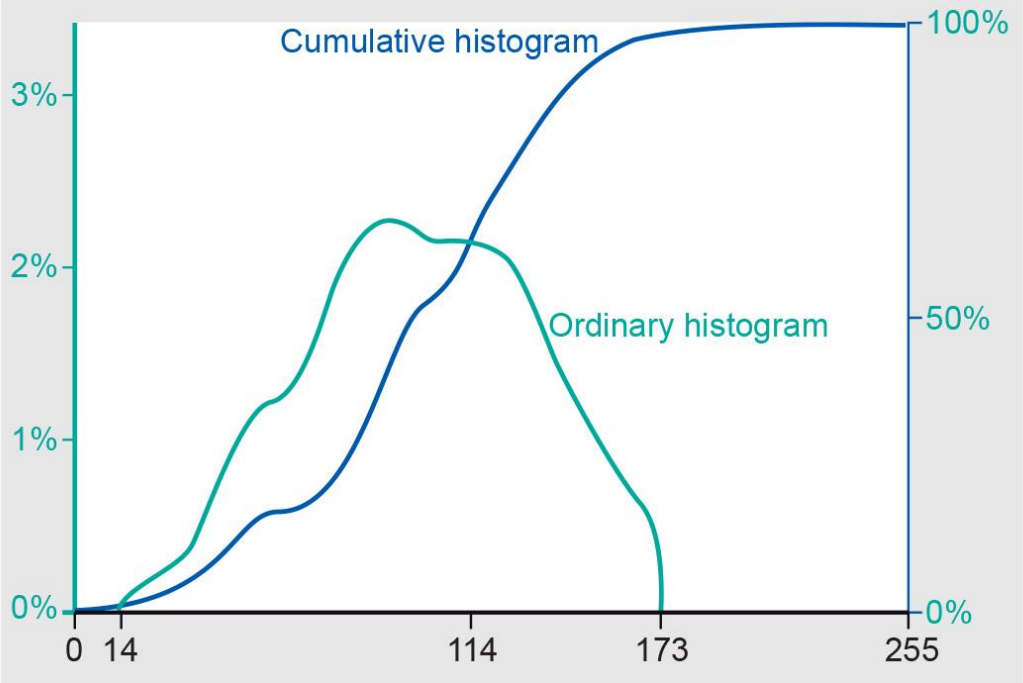

Рисунок 2.1 Приклад гістограми та кумулятивної гістограми зображення

Аналіз кумулятивних гістограм по стовбцях чи рядках покаже відношення кількості пікселів різних інтенсивностей у середині одного стовбця чи рядка, що дає можливість отримати дані по наявності та кількості різних пікселів уже не на усьому зображені, а тільки на одній частині. Ми також пропонуємо візуалізовувати гістограми у стовбці або рядки(в залежності від вибраного напрямку аналізу зображення), щоб мати можливість візуально оцінити, та обробити потім це відображення. У результаті ми отримаємо нове зображення, у якому кожен стовбець чи рядок відповідає за кумулятивне відображення цього ж самого стовбця чи рядка оригінального зображення.

Проаналізувавши це відображення ми отримаємо як інтенсивності пікселів змінюються в залежності від їх позиції на оригінальному

зображенні. Це інформація дозволяє визначити наявність дефекту на зображенні, та локалізувати його.

Також можна будувати зразу декілька гістограм на один стовбець чи рядок виділивши для кожної гістограми певний інтервал інтенсивностей який можна обчислити для певного стовбця чи рядка за формулою:

$$intRange = \lfloor (MaxInt - MinInt)/N \rfloor$$

$$intBeg_i = intEnd_{i-1}$$

$$intEnd_i = i * intRange + MinInt$$

де *MaxInt* та *MinInt* – верхня та нижня межа інтенсивностей що враховуються алгоритмом,

$intRange$ – довжина інтервалу

N - кількість гістограм на один рядок чи стовбець,

i - змінюється від 1 до *N*,

$intBeg_1 = MinInt$, - початкове значення інтервалу

$intEnd_N = MaxInt$ - кінцеве значення інтервалу

У такому випадку ми отримаємо для одного стовбця розподіл пікселів по інтервалах. Таких гістограм можна побудувати до N = 255 на один стовбець чи рядок, адже більше ніж 255 інтервалів інтенсивності виділити не можна, бо інтенсивність пікселя вимірюється від 0 до 255.

Для кожної із гістограм вираховується кількість пікселів кожної інтенсивності із стовбця чи рядка, що входять у інтервал інтенсивностей для цієї гістограми. Також обраховується загальна кількість пікселів з цього стовбця чи рядка, що мають інтенсивність із цього інтервалу. З допомогою цих даних обчислюється сама кумулятивна гістограма (формула 2.2).

$$histogram[i] = \frac{\sum_{j=0}^{H} 1,\ \ if(start \le intensity(column, j) \le i)}{count(histogram)} \qquad (2.2)$$

де для кожної інтенсивності *i* із інтервалу гістограми (start - end) обчислюється кількість пікселів (перевіряються усі пікселі j стовбця чи рядка від 0 до H – висота стовбця чи довжина рядка) з інтенсивністю менше

рівною за *i* та більше рівною за start, та ділиться на загальну кількість пікселів зображення із заданого інтервалу (start - end).

У результаті ми отримаємо масив із відносними значеннями від 0 до 1, кількості пікселів заданої інтенсивності *i* до загальної кількості пікселів із інтервалу інтенсивностей гістограми. Далі ми переводимо це значення до інтервалу від 0 до 255 помноживши значення з масиву на 255 та узявши цілу частину. Після цього ми візуалізуємо кумулятивну гістограму у стовбець висотою у 255 пікселів чи рядок довжиною 255 пікселів.

Якщо при створенні відображення використовувалась лише одна гістограма, значить вона покриває увесь інтервал інтенсивності від 0 до 255 (x = 0..255). Тоді для створення стовбця чи рядка достатньо просто узяти значення перетвореного інтервалу (від 0 до 255) для кожної інтенсивності із гістограми і у стовбці поставити на позицію x піксель із інтенсивністю що відповідає значенню із перетвореного інтервалу із позиції x (рис. 2.2). Аналогічний алгоритм і для побудови рядка.

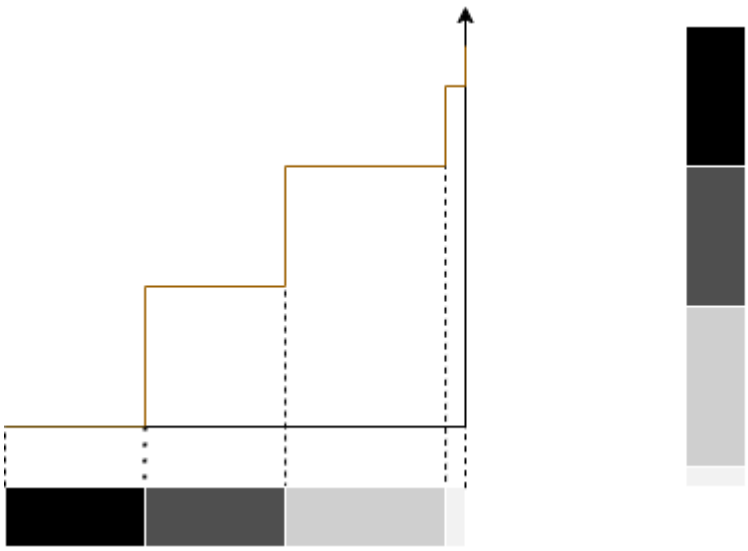

Рисунок 2.2 Перетворення кумулятивної гістограми у стовбець

Якщо ж гістограм для одного стовбця чи рядка було n, то для кожної гістограми будується аналогічний стовбець чи рядок і після цього вони об’єднуються у порядку інтервалів за які вони відповідають. Тобто якщо перша гістограма відповідає за інтервал інтенсивностей 0-9, а друга за

інтервал 10-19 то стовбець відображення другої гістограми будується під першим і так далі. У результаті вийде результуючий стовбець із вистою 255 пікселів. (рис. 2.3, приклад об’єднання 3 гістограм)

Рисунок 2.3 Відображення 3 кумулятивних гістограм у одному стовбці

2.2 Використання кумулятивних відображень для аналізу зображень

Для аналізу зображення його необхідно перетворити у зображення в відтінках сірого, для цього використаємо формулу 2.3, яка перетворює зображення у відтінки сірого із інтенсивністю, що відповідає інтенсивностям, які сприймаються людським оком [21]

$$pixel(x,y) = 0.299 \cdot Red(x,y) + 0.587 \cdot Green(x,y) + 0.114 \cdot Blue(x,y) \quad (2.3)$$

де інтенсивність результуючого пікселя з координатою (x,y) рівна сумі інтенсивностей усіх спектрів (Червоного Red, Зеленого Green, Синього Blue) цього пікселя помножених на відповідні вагові коефіцієнти.

Після цього усе зображення (нехай його розширення буде n стовбців на m рядків) розбивається по стовбцях чи рядках. Та для кожного стовбця чи рядка будується k гістограм. У нашому випадку варіант k=255 є

найцікавішим, так як ми отримаєм інформацію про наявність пікселів кожної інтенсивності у кожному рядку чи стовбці.

Для прикладу проаналізуємо зображення скачане із ресурсу freepik від автора wirestock (рис. 2.4). Його розширення 512x310 пікселів.

Рисунок 2.4 Зображення з інтернету для аналізу кумулятивними відображеннями

Для початку перетворимо за формулою 2.3 зображення у відтінки сірого (рис. 2.5)

Рисунок 2.5 Зображення у відтінках сірого

Ми бачимо що це зображення доволі темне. Дуже яскравих областей у ньому немає. Трохи є яскравого неба у середині зображення.

Перетворимо дане зображення у кумулятивні відображення по стовбцях (рис. 2.6) та по рядках (рис. 2.7).

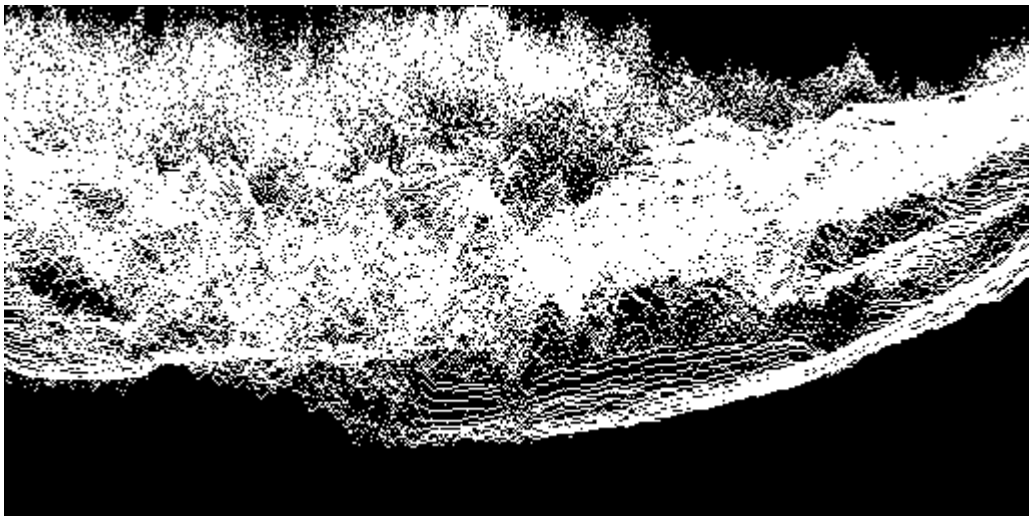

Рисунок 2.6 Кумулятивне відображення по стовбцях зображення із інтернету

Відображення по стовбцях має ширину рівну ширині вхідного зображення (512 пікселів) а висота його рівна 255 пікселям. Пікселі на ординаті 0 відповідають за наявність пікселів з інтенсивністю 0 у відповідних стовбцях зображення, відлік ординат іде зверху зображення. Кожен стовбець зображення відображає наявність пікселів різних інтенсивностей відповідного стовбця вхідного зображення.

З аналізу по стовбцях видно що у всіх стовбцях присутні темні пікселі та пікселі середніх тонів, але майже відсутні яскраві пікселі, та починаючи з певної інтенсивності взагалі відсутні пікселі, адже низ відображення повністю чорний, що означає яскравих пікселів не було у жодному стовбці.

Також видно що середина зображення містить найяскравіші пікселі, адже там відображення знаходиться найнижче, і на оригінальному зображенні це також можна помітити – у цій ділянці відбувається жовтий захід сонця, що створює багато світла по відношенню до усього зображенння.

Рисунок 2.7 Кумулятивне відображення по стовбцях зображення із інтернету

Відображення по рядках має висоту рівну висоті вхідного зображення (310 пікселів) а ширина його рівна 255 пікселя. Пікселі на абсцисі 0 відповідають за наявність пікселів з інтенсивністю 0 у відповідних рядках зображення, відлік абсцис іде зліва зображення. Кожен рядок зображення відображає наявність пікселів різних інтенсивностей відповідного рядка вхідного зображення.

Аналіз по рядках показує що верхня частина зображення не містить дуже темних пікселів, та майже відсутні яскраві пікселі. Потім ми бачимо що приблизно у середніх рядках зображення досить різко піднімається яскравість пікселів і майже усі рядки мають тільки пікселі із інтенсивністю більшою за 128 (середина інтервалу). Нижня ж частина зображення теж має різкий перехід зразу до темних тонів і там навпаки майже відсутні рядки із пікселями із інтенсивністю більшою за 128.

Проаналізувавши ці два відображення ми можемо сказати що на зображенні є світла ділянка, серед доволі темного усього зображення. Із аналізу по рядках ми можемо чітко сказати у яких рядках вона починається а у яких закінчується, а з аналізу по стовбцях ми можемо аналогічно сказати

у яких стовбцях ця ділянка починається і у яких закінчується. З допомогою даного аналізу можна виявляти прямокутні області на зображенні які мають дефекти.

2.3 Виявлення наявності дефектів матеріалів з допомогою кумулятивних відображень

З допомогою кумулятивних відображень, ми можемо легко аналізувати інтенсивності на зображеннях у двох напрямках (по рядках і по стовбцях), що має значно полегшити виявлення наявності дефектів на матеріалі.

Оглянемо декілька різних видів дефектів на поверхні металу:

1. Подряпина (рис. 2.8) – вузька полоса на поверхні металу, здебільшого пряма або у вигляді дуги, утворена внаслідок здирання металу якимось поступальним рухом вздовж його поверхні. Може бути як глибока так і легенька поверхнева. Характерні ознаки: товщина у декілька пікселів, темніша або світліша за поверхню метала, має зазвичай чіткі грані.

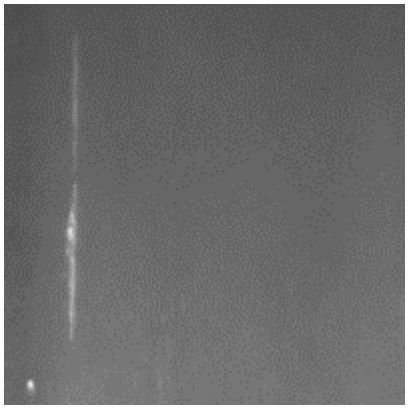

Рисунок 2.8 Приклад подряпин на металі

2. Вм’ятина (рис. 2.9) – незначна деформація поверхні металу, утворена внаслідок якогось удару. Розміри можуть бути різними як маленька так і велика, може бути вм’ятина усередину або назовні. Характерні ознаки: край вм’ятини темніший або

світліший у залежності від того чи вона усередину чи назовні, центр вм’ятини зазвичай такий же як і решта матеріалу.

Рисунок 2.9 Приклад вм'ятин на металі

3. Пробоїна (рис. 2.10) – дірка утворена внаслідок сильного удару. Розміри зазвичай невеликі, навколо пробоїни буде вм’ятина, адже метал пробував просто деформуватись, але не витримав навантаження. Або дуже рівний контур, якщо частину металу вирізали. Характерні ознаки: відсутність матеріалу, навколо пробоїни є вм’ятина.

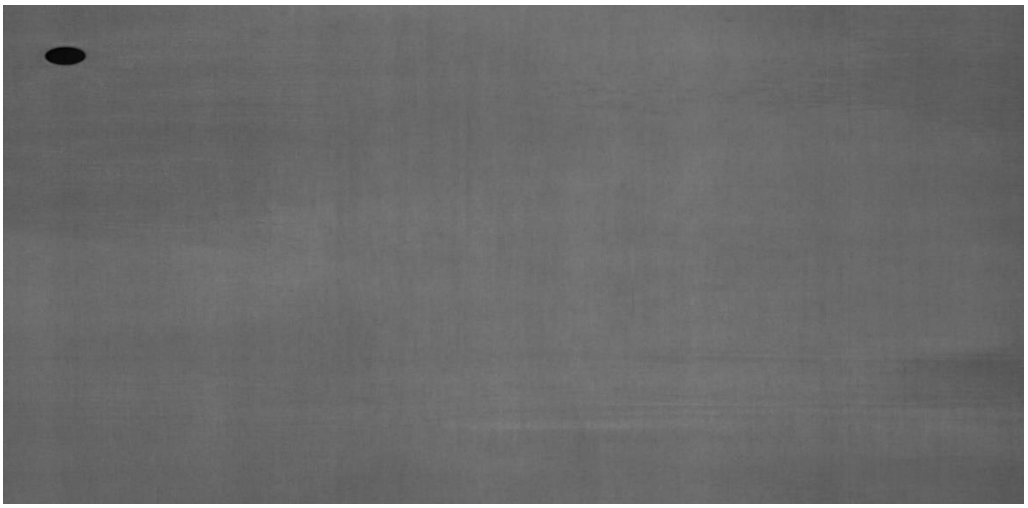

Рисунок 2.10 Приклад пробоїни на металі

4. Тріщина (рис. 2.11) – поверхневе, або наскрізне пошкодження металу доволі хаотичної форми зазвичай із гострими краями, що утворилось внаслідок критичного навантаження в наслідок деформацій, або вібрацій. Характерні ознаки: темніша за поверхню металу, невелика товщина(якщо поверхня повністю не зруйнувалась).

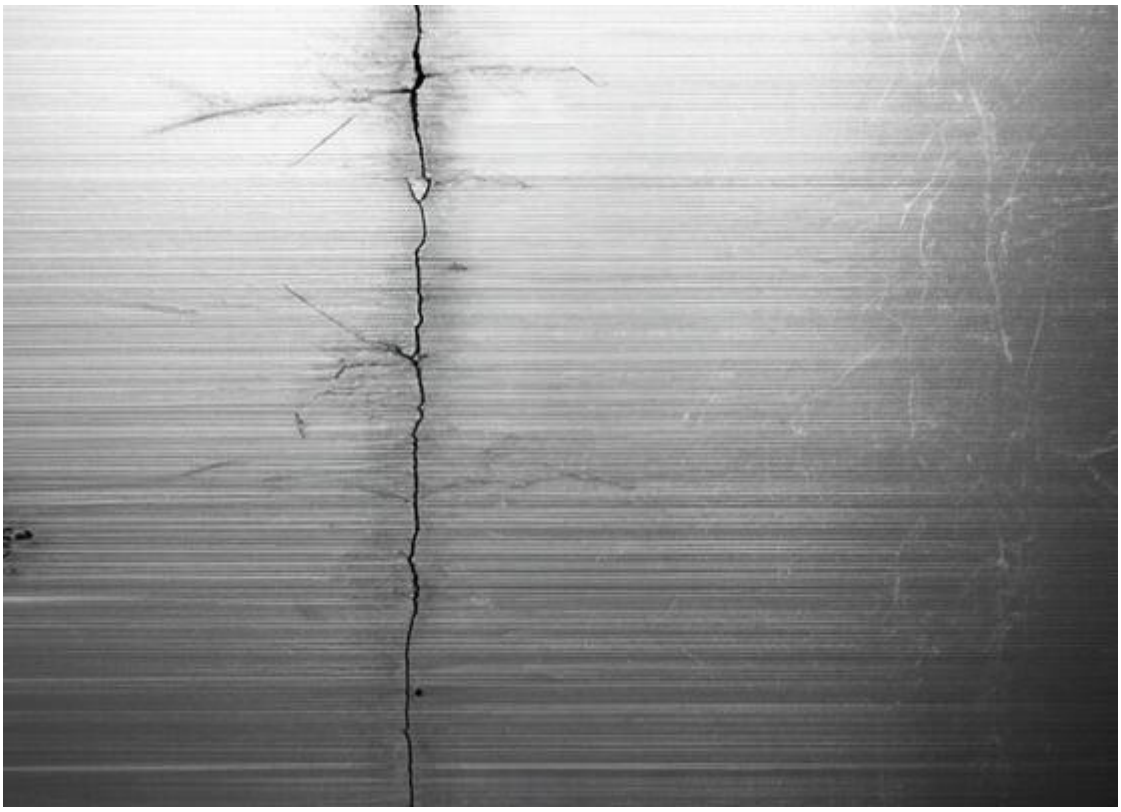

Рисунок 2.11 Приклад зображення тріщини

5. Інші дефекти виробництва – вкраплення, плями, сліди роликів та інші (рис. 2.12). Теж мають помітний перепад у інтенсивностях на зображенні, або у світлішу сторону або у темнішу, що можна легко визначити з допомогою запропонованого алгоритму виявлення дефектів на зображеннях.

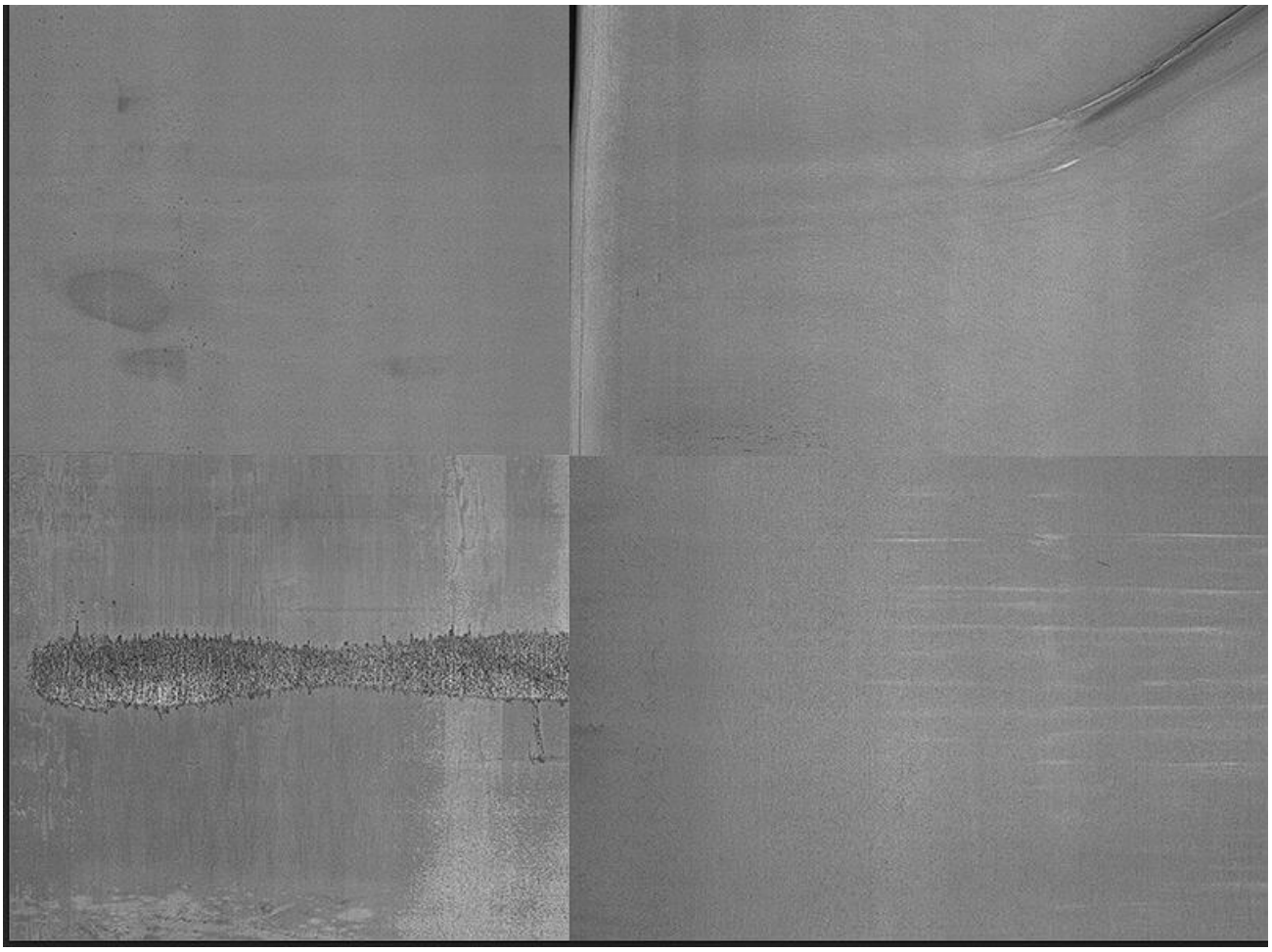

Рисунок 2.12 Різні дефекти виробництва металу

Усі ці дефекти мають іншу інтенсивність по відношенню до основної інтенсивності зображення поверхні, отже при аналізі зображення із будь-яким із дефектів на відображенні буде у стовбцях та рядках із дефектом помітно відхилення від нормальної інтенсивності.

Наявність цього дефекту можна виявити програмно. Для цього ми будуємо відображення або по стовбцях або по рядках, не має значення, адже дефект буде присутній в обох відображеннях. Після цього побудувати звичайну гістограму для усього зображення і з допомогою неї виявити основний колір фотографії металу (матеріалу) – найчастіша інтенсивність.

Перед початком дослідження зображення потрібно вибрати допустиму похибку у $\boldsymbol{p}$ процентів у межах якої відхилення на відображенні не рахуватимуться за дефекти. Після побудови відображення та виявлення

найчастішої інтенсивності на зображенні нам потрібно на відображенні перевірити чи є білі пікселі що знаходяться за межами допустимого прямокутника.

Допустимий прямокутник будується на відображенні. Якщо це відображення по стовбцях (рис. 2.13 червона лінія – найчастіша інтенсивність, оранжевий прямокутник – допустимий прямокутник) то прямокутник будується на усю ширину зображення, так щоб середня лінія була найчастішою інтенсивністю на зображенні.

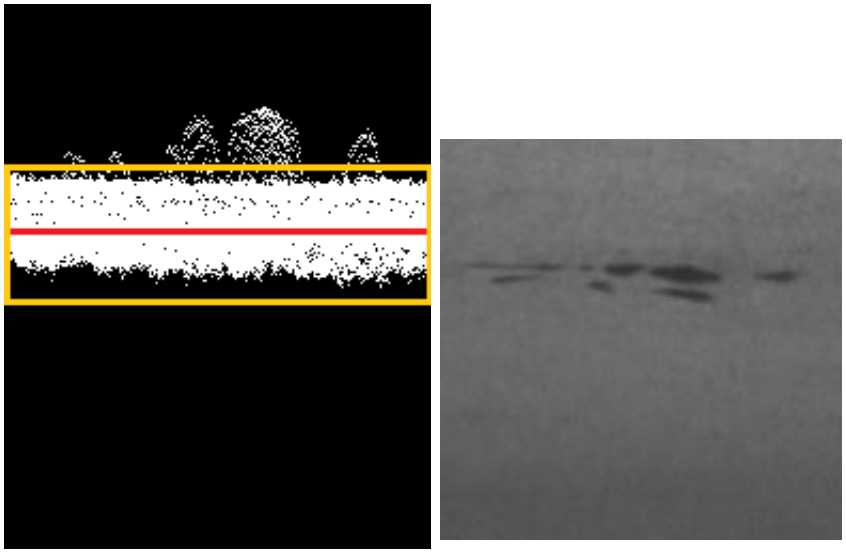

Рисунок 2.13 Зображення та його кумулятивне відображення з аналізом статичними методами

Для цього обчислюємо найчастішу інтенсивність на зображенні – ***m***, обчислюємо висоту прямокутника – ***h***, як добуток похибки на різницю між максимальним та мінімальним значеннями інтервалу побудови відображення за формулою:

$$h = p * (MaxInt - MinInt)$$

Далі на h/2 відкладаєм уверх і вниз від m і на усю ширину зображення у результаті отримуємо прямокутну область, що виражає допустимі для нас інтенсивності на зображенні, які ми не враховуватимемо як дефекти.

Далі можна або просто зафарбувати допустимий прямокутник у чорне і визначити чи залишились на зображенні білі пікселі. Якщо так значить дефекти присутні, якщо ні значить дефектів немає, при заданій похибці p.

Аналогічний аналіз можна провести розбиваючи зображення по рядках.

2.4 Класифікація дефектів на зображені

У попередньому розділі було описано, як за допомогою кумулятивних відображень можна ефективно виявити наявність дефектів виключно з використанням статистичних методів, без застосування складних моделей на базі нейронних мереж. Проте, важливим обмеженням цього підходу є неможливість класифікації виявлених дефектів за їх типом, оскільки кумулятивні гістограми втрачають деяку важливу інформацію про зображення - форму дефектів, їхню текстуру, контури та інші візуальні особливості, що є критичними для визначення конкретного типу дефекту.

Для вирішення даної проблеми та щоб розширити можливості запропонованого методу, ми пропонуємо проаналізувати зображення з допомогою двох кумулятивних відображень зображення по рядках, та по стовбцях. У результаті при наявності дефектів ми отримаємо аномалії на кумулятивному відображенні, та знаючи що при аналізі по рядках – рядок кумулятивного відображення відповідає рядку оригінального зображення, а по стовбцях – стовбці відповідають оригінальному зображенню, ми можемо отримати координати стовбців та рядків (рис. 2.14) у яких є дефекти. Завдяки такому аналізу метод кумулятивних відображень дозволяє легко виділити прямокутники із потенційно наявними дефектами - зробивши перетин усіх стовбців у яких виявлено дефекти з усіма рядками із дефектами.

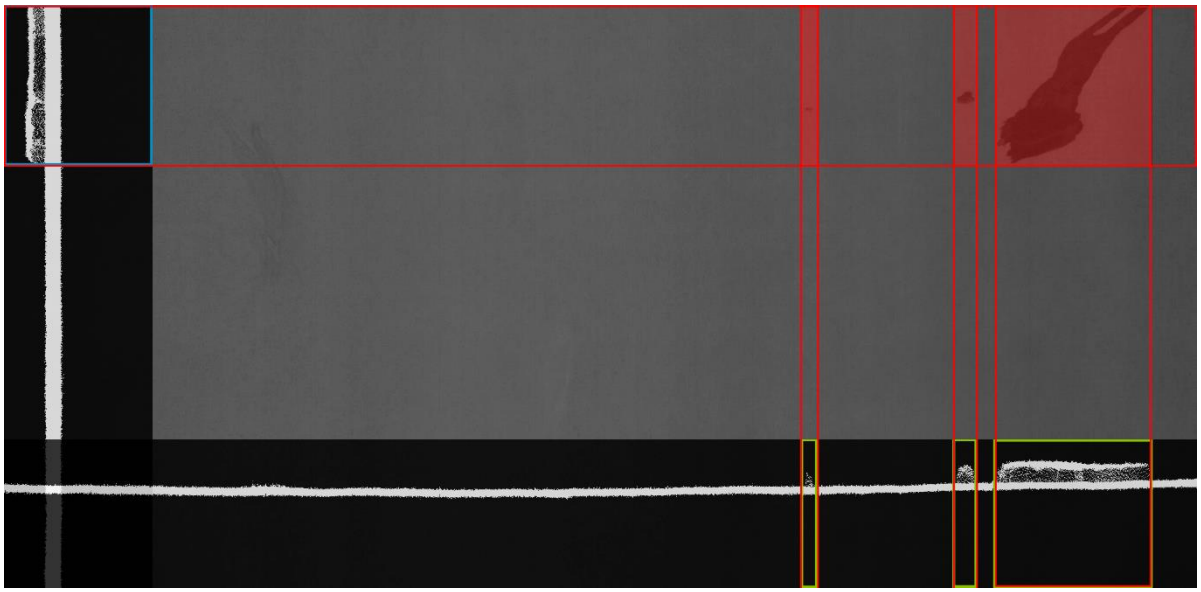

Рисунок 2.14 Проаналізоване зображення кумулятивними відображеннями з виділеними квадратами із дефектами

Виділивши прямокутники із дефектами, ми можемо обробляти їх для класифікації дефектів у оригінальній роздільній здатності, без стиснення та без поділу на величезну кількість під зображень оригінального зображення із подальшою обробкою кожного із них з допомогою нейронних мереж.

Уже виділені квадрати мають значно менше розширення у порівнянні з початковим зображенням, отже ми уже отримуємо значно менше зображення на вході нейронної мережі, а його подальше зменшення не зможе нівелювати дефект, адже ми не стискаємо велике зображення у 10 разів, а лише трохи зменшуємо, або навіть збільшуємо розширення щоб на зображення відповідало входу мережі, що дозволяє використовувати для класифікації не великі складні нейронні мережі що стараються врахувати усі можливі викривлення у результаті змін у зображенні, а малі та швидкі які на основі декількох шарів обробляють зображення та класифікують його відповідно до даних на яких навчались.

2.5 Висновки

Для написання роботи буде використано запропонований уперше метод побудови кумулятивних відображень, який перетворює вхідне зображення по рядках чи по стовбцях у n-ну кількість кумулятивних

гістограм, кожна із яких відповідає за свій інтервал інтенсивності. Побудовані гістограми відображаються у стовбці (або рядки) будуючи відображення одне під одним (або справа одне від одного). Для побудови відображення будується стовбець (або рядок) висотою у інтервал інтенсивності даної кумулятивної гістограми, після цього кожен піксель стовбця замальовується у колір, що відповідає значенню відповідної інтенсивності у кумулятивній гістограмі помноженої на 255. У результаті якщо для якоїсь інтенсивності x (від початкового значення інтервалу гістограми до кінцевого значення інтервалу) кумулятивної гістограми значення буде k є [0;1], то помноживши його на 255 отримаємо число із проміжку [0..255], яке перетворюється у інтенсивність (колір у відтінках сірого) пікселя з координатою x.

Побудувавши відображення ми можемо з допомогою звичайної гістограми для усього зображення виявити найчастішу інтенсивність, що зустрічається на зображені. Указавши допустиму похибку p для виявлення дефектів, ми можемо відсіяти найчастішу інтенсивність, тим самим залишиться тільки відображення інтенсивностей, що відрізняються більше ніж на p процентів від найчастішої інтенсивності. Ці відхилення ми і виявляємо як наявність дефектів.

Також ми можем визначити номери стовбців та рядків де ці відхилення були виявлені, перетнувши їх на оригінальному зображенні ми отримаємо прямокутники у середині яких будуть знаходитись дефекти. Ці прямокутники можна у подальшому обробити простими нейронними мережами для їх класифікації.

У результаті система зможе виявляти наявність дефектів з допомогою статистичного аналізу. У подальшому систему можна розвивати добавивши визначення положення дефектів, визначаючи координати стовбців та рядків із дефектами, та обрізавши прямокутник між цими стовбцями і рядками опрацьовувати дане зображення з допомогою простої нейронної мережі для його класифікації.

РОЗДІЛ 3. РОЗРОБКА ПРОТОТИПУ ПРОГРАМНОЇ СИСТЕМИ ВИЯВЛЕННЯ ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ МЕТАЛІВ

3.1. Обґрунтування вибору алгоритму аналізу кумулятивних гістограм

У процесі дослідження проблеми виявлення дефектів металів було розглянуто різні методи та алгоритми для цієї задачі. В попередньому розділі описаний метод аналізу зображення з допомогою кумулятивних відображень. У цьому розділі ми опишемо розробку прототипу програмної системи для дослідження запропонованого алгоритму. Перевагою даного алгоритму є його масштабованість, універсальність та простота реалізації.

1. Масштабованість - алгоритм побудови кумулятивних відображень легко піддається розпаралеленню. Кожну ділянку зображення можна аналізувати окремо, а об’єднання результатів проводити вже на завершальному етапі. Це дає змогу ефективно використовувати ресурси багатоядерних процесорів або швидких графічних прискорювачів.
2. Універсальність - кумулятивні гістограми дозволяють виявляти усі можливі типи дефектів на матеріалі незалежно від даних на яких алгоритм проектувався. Тому цей підхід може застосовуватися для виявлення різноманітних дефектів: тріщини, вибоїни, нерівності, слідів корозії тощо.
3. Простота реалізації. Порівняно з деякими іншими методами (наприклад, складними нейронними мережами), реалізація алгоритму побудови та аналізу кумулятивних гістограм доволі швидко програмується та не вимагає навчання на різноманітних датасетах, що дозволяє запустити систему готову до використання у короткі строки.

Об’єднання цих аспектів робить метод кумулятивних гістограм оптимальним вибором для швидкого й точного виявлення дефектів на зображеннях металів, особливо коли важливо дотриматися балансу між продуктивністю та ефективністю. Дослідження даного алгоритму з допомогою запропонованого прототипу програмного забезпечення дозволить визначити його точність, швидкодію та порівняти його із аналогами, та отримавши хороші результати покращувати його (для збільшення швидкодії або точності) та розробляти повноцінну систему для аналізу зображень на наявність дефектів та проводити її дослідження у реальних або наближених до реальних умовах.

3.2. Обґрунтування використання C# та Windows Forms

Для розробки прототипу системи для дослідження алгоритму вирішено розробити десктопний застосунок. Для цього було використано мову програмування C# у парі із технологію Windows Forms. Цей стек технологій за останні роки знов почав набирати популярності. Мова програмування C# уже багато років перебуває у топі серед мов програмування. Технологія Windows Forms хоч і є достатньо уже старою, але за останні роки знову почала набирати популярності, коли у ній з’явилась підтримка Blazor, що дозволяє створювати та стилізувати компоненти з допомогою Web верстки. Також багато застосунків було розроблено колись на Windows Forms, і за останні декілька років їх почали активно оновлювати, що дуже підняло популярність даної технології.

Такий вибір є оптимальним з огляду на декілька важливих аспектів, які безпосередньо вплинули на продуктивність, гнучкість розробки та масштабованість системи.

1. C# – це об’єктно-орієнтована мова, що вирізняється високим рівнем абстракції та зручним синтаксисом, успадкованим

частково від Java та C++. Завдяки механізмам керованого коду (managed code) та автоматичного збирання сміття (garbage collection) вона спрощує написання і налагодження складних програм, зокрема алгоритмів обробки зображень. Усі ці особливості роблять C# привабливою мовою для розробників, коли йдеться про швидку розробку різноманітних сервісів (а саме API та декстопних застосунків Windows) і забезпечення стабільної роботи великих проєктів.

2. Зручні інструменти для розробки асинхронного коду - у C# паралельне виконання коду реалізоване через низку високорівневих інструментів. Найбільш поширені з них – Task Parallel Library (TPL), що дозволяє ефективно розподіляти навантаження між ядрами процесора. У випадку обробки зображень, та побудови кумулятивних відображень до них, які можуть бути досить великими за розміром, така можливість особливо актуальна: обробку зображення можна розділи на сектори кожен з них опрацьовувати у окремому потоці, прискоривши виконання алгоритму в рази. При цьому C# забезпечує безпечну роботу з потоками, мінімізуючи ризики конфліктів і збоїв, що часто виникають у низькорівневому багато поточному середовищі.
3. Windows Forms – технологія у середовищі .NET, що значно спрощує створення графічного інтерфейсу користувача. Замість того, щоб писати багато коду для відтворення кнопок, списків, полів введення та елементів відображення зображень, розробник може використовувати зручний дизайнер форм. Це дозволяє швидко додавати або видаляти елементи керування, налаштовувати їхні властивості й додавати обробники події цих елементів (натискання кнопок, введення тексту тощо). Завдяки цьому можна швидко створити прототип застосунку, провести

користувацьке тестування та вносити зміни до застосунку без істотних зусиль. Окрім того, Windows Forms достатньо легка у підтримці, бо має просту структуру й не вимагає глибоких знань про векторну графіку чи верстку XAML.

4. Велика екосистема - .NET надає широкий вибір стандартних і сторонніх бібліотек. Це означає, що для багатьох поширених завдань (наприклад, для читання й запису файлів зображень різних форматів, роботи з базами даних, створення звітів, логування, тестування тощо) розробнику достатньо тільки підключити відповідну бібліотеку. Це дає можливість швидко інтегрувати у проєкт складні алгоритми обробки зображень та машинного навчання.
5. Зручне розгортання та підтримка - застосунок C# можна легко запустити на усіх пристроях із Windows, застосунок легко поширюється як єдиний exe файл або інсталятор із мінімальною кількістю залежностей. Оновлення може виконуватися заміною файлів або за допомогою стандартних інструментів розгортання, що суттєво скорочує витрати на супровід системи. Користувачі отримують зрозумілий та звичний для них інтерфейс.

C# у поєднанні з Windows Forms є дуже добре задокументованим. Існують численні приклади, форуми та розгорнута офіційна документація від Microsoft, що полегшує навчання і вирішення будь-яких проблем, які можуть виникнути під час розробки. Висока популярність платформи у корпоративному сегменті дозволяє досить просто та швидко отримати підтримку від спільноти.

Таким чином, поєднання можливості паралельного програмування, швидкого прототипування інтерфейсу, доступу до розвинених бібліотек та простоти розгортання робить C# у поєднанні із Windows Forms чудовим

вибором для розробки прототипу програмної системи. Така комбінація забезпечує гнучкість, масштабованість і стабільність, що є критично важливими чинниками в розробці системи виявлення дефектів на зображеннях металів.

3.3. Опис середовища розробки (Visual Studio)

Для реалізації застосунку було обрано інтегроване середовище розробки (IDE) Visual Studio. Це є потужний інструмент для створення додатків на платформі .NET. Дане середовище розроблене у Microsoft та користується високою популярністю серед програмістів.

Це середовище розробки має низку переваг, які значно покращують та спрощують процес розробки застосунків:

1. Інтегрований дизайн форм (Windows Forms Designer) - однією з ключових переваг Visual Studio для розробників, що працюють із Windows Forms, є вбудований візуальний дизайнер. З його допомогою елементи інтерфейсу такі як кнопки, панелі, списки, поля введення та інші можна розміщувати, та редагувати на формі в інтерактивному режимі. Це значно зменшує обсяг коду що необхідно написати, адже увесь код для дизайну генерується автоматично, що значно прискорює процес прототипування та дозволяє швидко змінювати зовнішній вигляд застосунку, або додавати чи оновлювати функціонал.
2. Вбудовані інструменти відлагодження (Debugger) - Visual Studio надає розвинений дебагер, що дозволяє покрокове виконання програми, перевіряти значення змінних, створювати брейкпоінти та навіть змінювати код під час виконання.
3. Інструменти профілювання (Performance Profiler, Diagnostic Tools) - допомагають проаналізувати алгоритми під час роботи, що особливо корисно при обробці зображень великого розміру

або під час інтенсивних паралельних обчислень, та дозволяє оптимізувати та пришвидшити код.

4. Управління залежностями (NuGet) - інтеграція з NuGet дозволяє швидко та ефективно знаходити й підключати сторонні бібліотеки, та пакети, що розширюють можливості застосунку. Це суттєво прискорює процес розробки, оскільки відпадає потреба в ручному пошуку, завантаженні та налаштуванні сторонніх рішень. Досить додати їх через менеджер пакетів — і Visual Studio автоматично здійснить налаштування.
5. Система контролю версій (Git) - вбудована підтримка системи Git забезпечує зручний спосіб зберігати історію змін у проєкті, працювати з гілками, робити злиття та відслідковувати різні версії коду. Це дозволяє повертатись до різних версій застосунку, та ітеративно проводити розробку добавляючи новий функціонал по мірі готовності у окремих гілках.
6. Підтримка декількох мов програмування і технологій - Visual Studio не обмежується лише мовою C# — у ньому можна писати код на F#, Visual Basic, C++, а також створювати веб-додатки (ASP.NET). Хоча цей прототип програмної система орієнтований на C# у комбінації із Windows Forms, майбутнє масштабування системи чи додавання нових функціональних модулів наприклад, веб-сервісів не становитиме проблеми.
7. IntelliSense та розумні модулі для аналізу коду - Visual Studio забезпечує покращену підсвітку синтаксису, автозавершення коду, контекстні підказки, а також швидку навігацію по проєкту. Це не тільки підвищує швидкість написання коду, але й допомагає запобігти типових помилок, адже середовище розробки аналізує синтаксис та структуру коду в режимі реального часу.
8. Зручність і універсальність для корпоративних рішень - Visual Studio часто використовується в корпоративних середовищах

завдяки масштабованості, стабільності і розвиненим інструментам Azure DevOps. Отже, якщо проєкт зростатиме і з’явиться потреба в складніших CICD-процесах чи його розгортанні у хмарному середовищі, то Visual Studio уже має вбудовані рішення для цих задач.

Таким чином, Visual Studio забезпечує ефективне та зручне середовище розробки, яке сприятливо впливає на швидкість, якість і надійність створення коду. Наявність потужних інструментів для відлагодження та візуального дизайнеру для Windows Forms робить його одним із найкращих IDE для розробки прототипу програмного забезпечення для виявлення дефектів на зображеннях металів, особливо коли потрібна ефективна взаємодія з Windows Forms і багато поточне виконання алгоритмів обробки зображень.

3.4. Проектування клієнтського застосунку

Основні вимоги до застосунку - це можливість збирати дані та аналізувати результати, а саме швидкодію та точність визначення наявності дефектів. Важливим для застосунку є можливість завантажити одне зображення, створити кумулятивне відображення, вивести його на екран та проаналізувати його як з допомогою алгоритму так і візуально з допомогою людського ока. Такий підхід дозволяє оцінити роботу алгоритму та результати отримані ним.

Основними елементами інтерфейсу є два поля де відображається завантажене зображення та будується зображення кумулятивного відображення. Інтерфейс зображений нижче на рисунку 3.1.

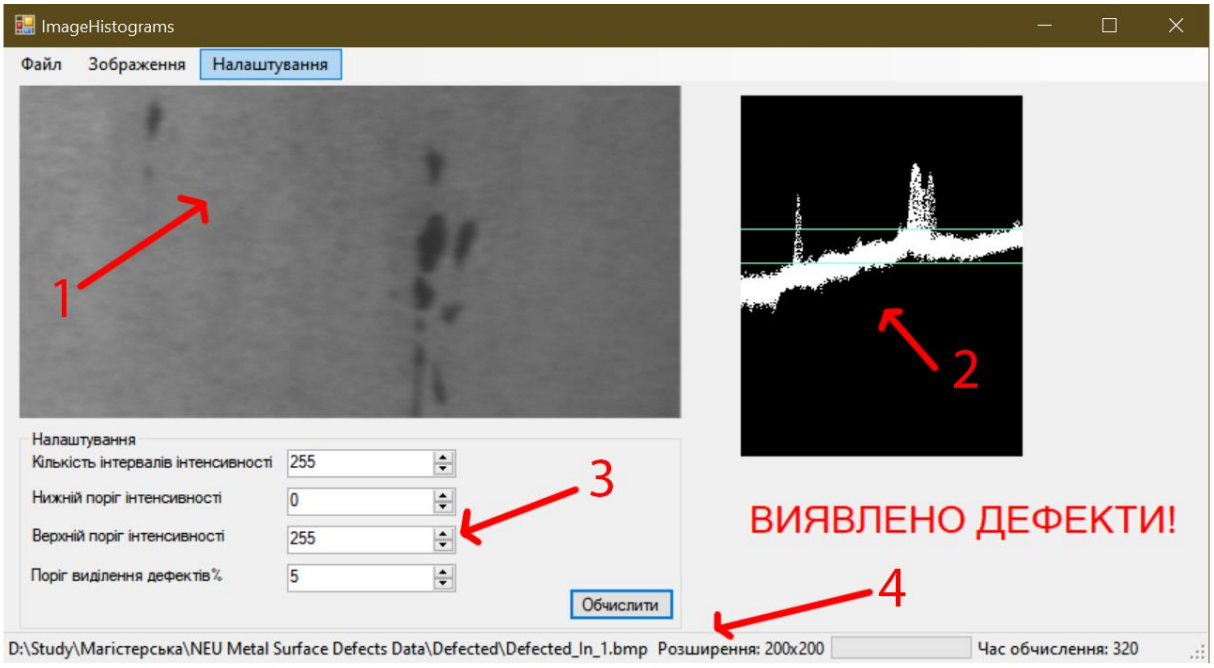

Рисунок 3.1 Знімок екрана із інтерфейсом застосунку

У інтерфейсі присутні наступні елементи:

1. Панель із завантаженим зображенням – відображає зображення із якого буде побудовано кумулятивне відображення, та яке буде проаналізовано алгоритмом на наявність дефектів.
2. Панель із побудованим кумулятивним відображенням – відображає побудоване зображення кумулятивного відображення
3. Налаштування параметрів алгоритму – налаштування кількості інтервалів інтенсивностей (відповідає за кількість гістограм якими аналізується зображення), нижній поріг інтенсивності (відповідає за мінімальне значення інтенсивності, яке береться до уваги алгоритмом, дозволяє повністю прибрати дуже темні ділянки зображення що не несуть інформації), верхній поріг інтенсивності (відповідає за максимальне значення інтенсивності, яке береться до уваги алгоритмом, дозволяє повністю прибрати дуже світлі ділянки зображення що не несуть інформації), поріг виділення дефектів (відповідає за похибку що вважатиметься нормою і алгоритмом не виявляється як наявність дефекту)

4. Інформаційний рядок – відображає шлях до зображення що відкрите програмою на даний момент, його роздільну здатність, прогрес обробки зображення алгоритмом та загальний час обробки зображення

У верхній частині робочого вікна є рядок меню, для взаємодії із програмою з наступними пунктами:

1. Файл (рис. 3.2) – у цьому меню знаходяться основні елементи управлінням програмою.
    a. Відкрити декілька – відкрити папку із зображеннями для їх аналізу, аналіз зображень відбувається автоматично і у результаті створюється файл із статистикою у форматі csv, де виводиться інформація про аналіз кожного зображення. Також обчислюється та зберігається у файлі процентне співвідношення між правильно класифікованими зображеннями та загальною кількістю зображень що аналізувались.
    b. Відкрити – відкриває одне зображення, показує його у панелі для вхідного зображення, та аналізує його із вибраними налаштуваннями при натисканні кнопки обчислити, будуючи кумулятивне відображення та відображаючи його у панелі для нього.
    c. Зберегти результат – зберігає побудоване кумулятивне відображення із панелі для кумулятивного відображення у файлі у вибраній директорії.
    d. Зберегти зображення – зберігає оброблене вхідне із панелі для вхідного зображення у файлі у вибраній директорії.
    e. Вихід – закриває програму.

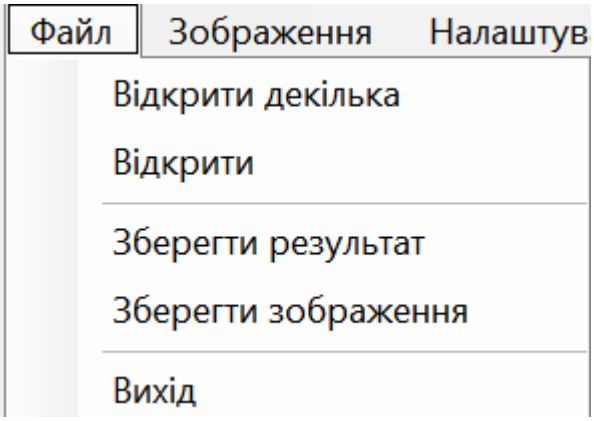

Рисунок 3.2 Меню Файл

2. Зображення (рис. 3.3) – у цьому меню знаходяться усі фільтри перед обробки зображення, для даного прототипу програмного забезпечення достатньо тільки перетворення до відтінків сірого. Ці фільтри накладаються на вхідне зображення при натисканні на відповідну кнопку а результат їхньої роботи відображається на панелі вхідного зображення

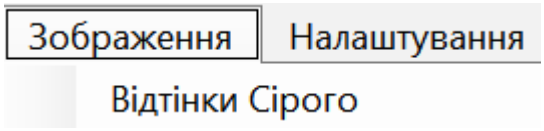

Рисунок 3.3 Меню Зображення

3. Налаштування (рис. 3.4) – у цьому меню вибирається напрям обходу зображення алгоритмом або вертикальний (по стовбцях) або горизонтальний (по рядках). Реалізовано у вигляді перемикача (при виборі Горизонтальний, Вертикальний вибір відміняється і навпаки), адже для виявлення дефектів достатньо здійснити обхід тільки в одному напрямку.

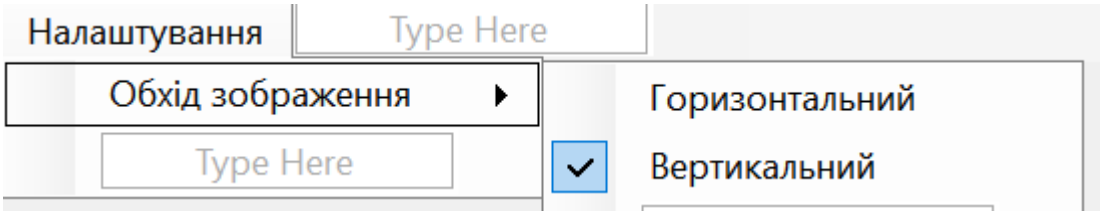

Рисунок 3.4 Меню Налаштування

Також на рисунку 1 показано використання системи для аналізу одного зображення, де видно у програмі завантажене зображення, вибрано параметри для його аналізу, побудоване та проаналізоване його кумулятивне відображення.

При аналізі багатьох зображень потрібно їх вибрати у вікні що відкрилось множинним вибором(з допомогою кнопки Shift на клавіатурі). Після цього натиснути кнопку відкрити та зразу почнеться аналіз, при завершенні якого ми отримаємо повідомлення у вікні, про завершення аналізу, а результати збережені у файл (рис. 3.5)

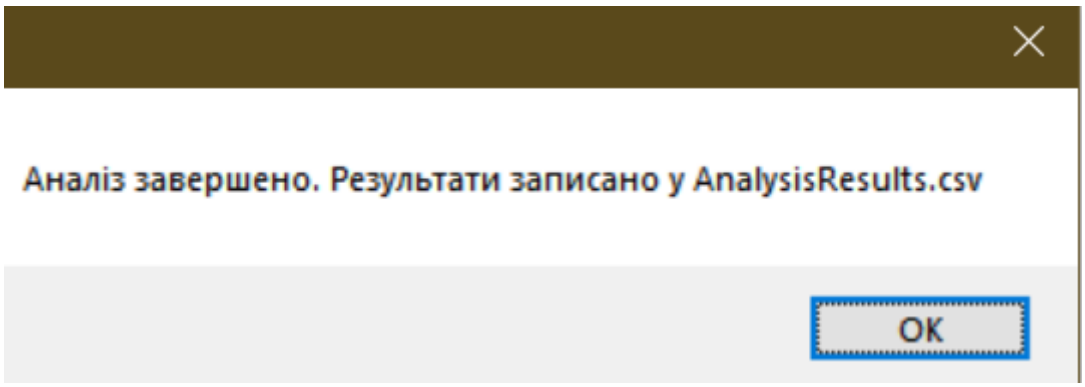

Рисунок 3.5 Повідомлення про завершення аналізу

У файлі ми зберігаєм наступну інформацію:

- FileName – назва файлу що аналізується
- HasDefectInName – чи у назві файлі написано що він містить дефект
- DetectedByAlgorithm – чи алгоритм виявив дефект на даному зображенні
- IsCorrect – чи збігається результат аналізу із реальною наявністю дефекту на зображені
- Accuracy – обчислюється та зберігається внизу після перерахування усіх проаналізованих зображень підрахована точність.

Структуру програми можна описати наступною UML діаграмою (рис. 3.6)

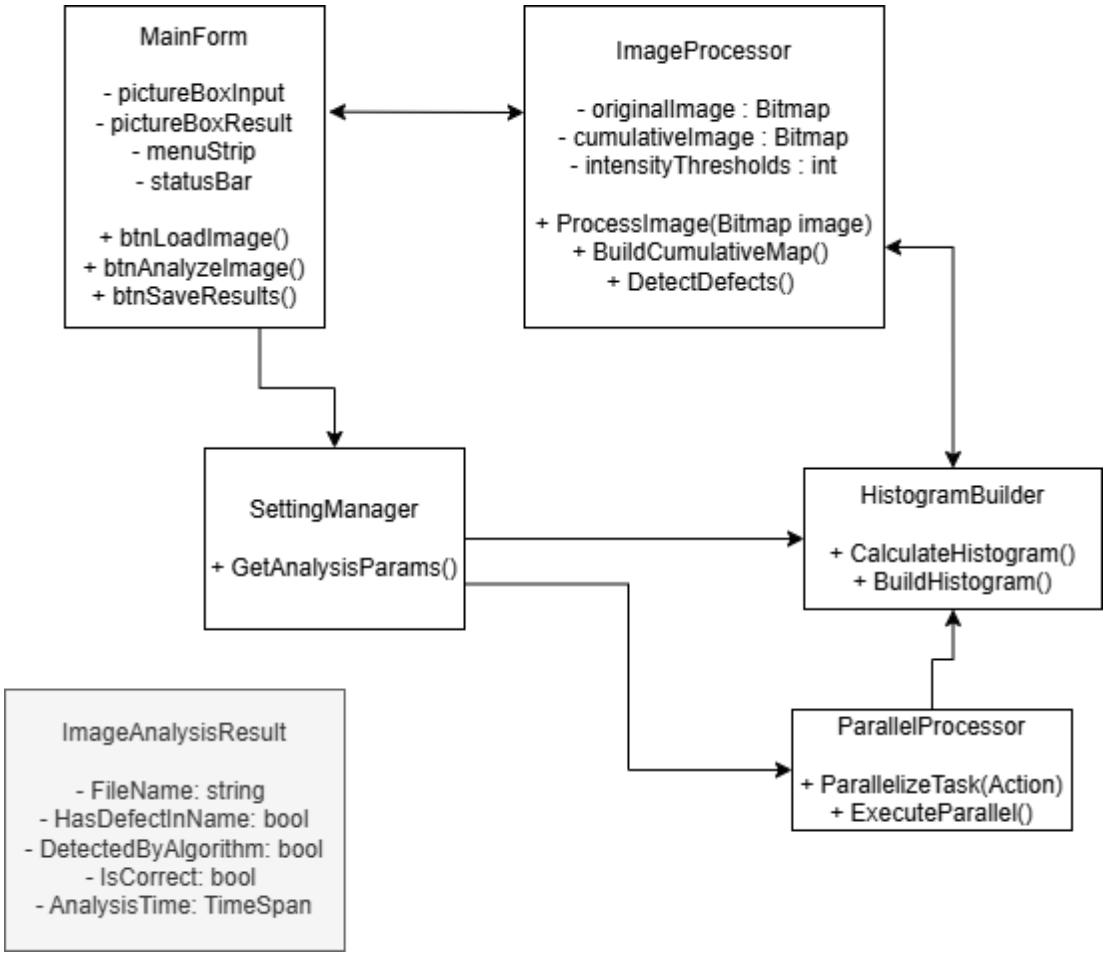

Рисунок 3.6 UML діаграма застосунку

На цій діаграмі зображені наступні елементи:

1. MainForm - головний клас, що відповідає за взаємодію користувача з застосунком. Основна форма що містить усі елементи управління, логіку запуску усіх функцій, та усі елементи інтерфейсу. У ньому реалізовані функції: завантаження зображення, запуску аналізу, налаштування параметрів та збереження результатів роботи.

   btnLoadImage – відкриває діалог вибору файлу зображення для завантаження його у програму.

   btnAnalyzeImage – запускає алгоритм побудови кумулятивного відображення та алгоритм виявлення наявності дефектів.

btnSaveResults – зберігає результуюче кумулятивне відображення у файл.

2. ImageProcessor - основний клас, який реалізує логіку обробки зображень та виявлення дефектів. Він здійснює обробку зображення, побудову кумулятивної гістограми і визначає наявність дефектів.

   ProcessImage(Bitmap image) – метод перед обробки зображення, а саме перетворення його до відтінків сірого. Даний метод використовує перетворення кожного пікселя до відтінків сірого за формулою, що відповідає інтенсивностям сприйняття кольорів людським оком (див. формула 2.3)

   BuildCumulativeMap() – побудова кумулятивного відображення із завантаженого зображення.

   1. Відсікає нижню та верхню межі інтенсивностей отримуючи їх із SettingsManager.
   2. Отримує з SettingsManager кількість гістограм для дослідження зображення.
   3. Отримує з SettingsManager напрям обходу зображення алгоритмом (вертикальний чи горизонтальний).
   4. В залежності від напряму обходу починає по рядку або по стовбцю обходити зображення та формувати кумулятивне відображення використовуючи алгоритм та формули описані у розділі 2.1.

   DetectDefects() – аналізує кумулятивне відображення на наявність дефектів.

   1. Визначається найчастіша інтенсивність на зображенні - вона береться за основу при виявленні дефектів, як бездефектний матеріал.
   2. Отримує з SettingsManager допустиму похибку оцінки наявності дефектів.

3. Аналізує з допомогою алгоритму та формул описаних у розділі 2.3 наявність дефектів на зображенні з допомогою кумулятивного відображення.

3. ImageAnalysisResult - клас, що зберігає результати аналізу окремого зображення. У собі містить: FileName - шлях до файлу, HasDefectInName - наявність у файлі дефекту (оприділяється за наявністю «defected_» у назві), DetectedByAlgorithm - факт виявлення дефекту алгоритмом, IsCorrect - правильність результату, AnalysisTime - час обробки даного зображення алгоритмом.
4. ParallelProcessor - допоміжний клас, що забезпечує розпаралелення задач та покращує швидкодію системи шляхом багато потокового виконання завдань. Він надає методи для запуску паралельних задач, особливо важливих при аналізі великих зображень або великої їх кількості.
   Розпаралелення відбувається шляхом поділу зображення на частини, які опрацьовуються паралельно, а після обробки об’єднуються в одне кумулятивне відображення, яке уже в подальшому аналізується окремо. Із SettingsManager отримується кількість потоків для розпаралелення, та напрям обходу. Після чого знаючи розширення зображення у даному напрямі обходу(якщо обхід по рядках, то отримуємо кількість рядків із зображення, якщо по стовбцях то кількість стовбців). Далі обчислюється кількість рядків або стовбців що будуть оброблятись одним потоком. Створюється задана кількість потоків і у кожен завантажуються відповідні рядки або стовбці для обробки. Якщо кількість рядків чи стовбців не ділиться націло на кількість потоків для обчислення, то у останній потік завантажується додатково частини що залишились. Усі потоки запускаються одночасно, та мають приблизно однакове навантаження, що у результаті дає приблизно однаковий час робити для кожного із потоків, а отже при обробці великих зображень

завантаженість потоків буде приблизно однакова, завантаженість системи буде максимальна, а час роботи із зображенням мінімальний.

5. SettingsManager – клас, що відповідає за отримання та передачу налаштувань роботи алгоритму (кількість гістограм для аналізу, мінімальна інтенсивність, максимальна інтенсивність, поріг виявлення, напрям обходу зображення), та налаштувань розпаралелення (кількість потоків для розпаралелення).

3.5. Опис датасету

У межах цього дослідження використовувалося два набори зображень узятих із відкритих джерел - NEU-DET[16] та GC10-DET[17]. Які були об’єднані в єдину базу.

Датасет NEU-DET містить 1800 квадратних зображень малого розширення(200x200) із 6 різними типами дефектів – Сітчасте розтріскування (Crazing), Включення (Inclusion), Плями (Patches), Ямковість (Pitted), Прокатані дефекти (Rolled), Подряпини (Scratches). На рис. 3.7 подано декілька зображень із даного датасету.

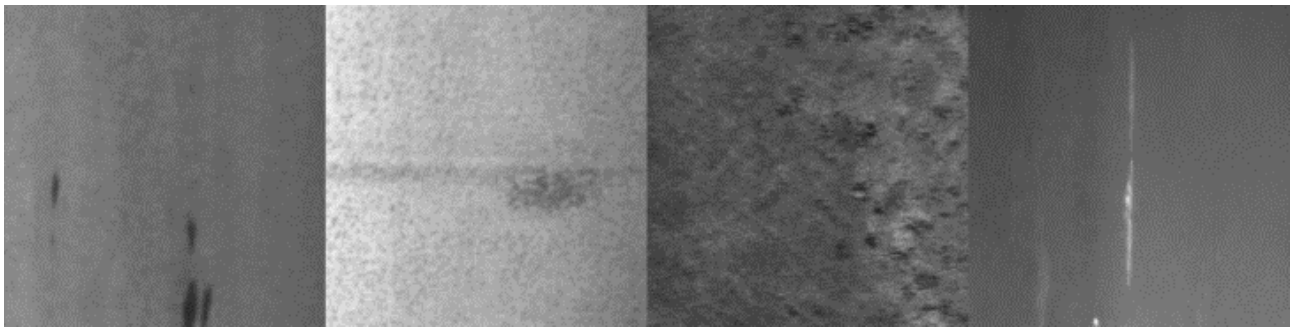

Рисунок 3.7: Зображення із дефектами із датасету NEU-DET

Датасет GC10-DET містить 8 566 прямокутних зображень великого розширення(2048x1000) з 10 різними типами дефектів – Згини (crease), півмісячний розрив (crescent gap), Включення (inclusion), Масляні плями (oil spot), Пробивний отвір (punching hole), Прокатана ямка (rolled pit), Шовковиста пляма (silk spot), Поздовжня складка (waist folding), Водяні

плями (water spot), Лінія зварювання (welding line). На рис. 3.8 подано декілька зображень із даного датасету.

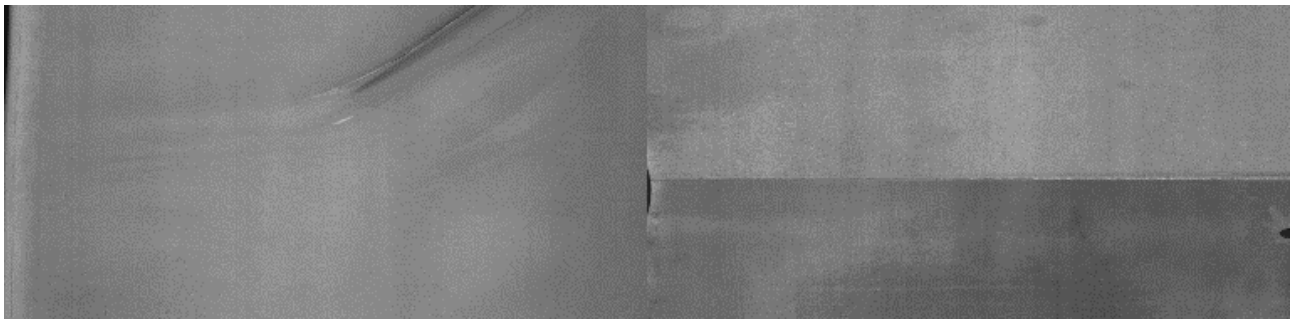

Рисунок 3.8: Зображення із дефектами із датасету GC10-DET

1. **Зображення з дефектами.** Файлам, що містять дефекти, було додано префікс «defected_». При дослідженні точності роботи алгоритму такий префікс дозволяє швидко визначити до якої категорії належить те чи інше зображення, та швидко порівняти із результатом що видав алгоритм при обробці даного зображення. Зображення що позначені як «defected_» містять різноманітні пошкодження поверхні: тріщини, вибоїни, плями корозії тощо.
2. **Зображення без дефектів.** Із високо роздільних знімків металевих виробів із датасету GC10-DET було вирізано фрагменти, де відсутні будь-які візуальні пошкодження. Такі фрагменти розглядаються як еталон для бездефектного стану металу.

Об’єднання цих двох датасетів, та групування їх за наявністю чи відсутністю дефектів дозволяє системі ефективно визначати наявність або відсутність дефектів, ефективно проводити дослідження із зображеннями різних розмірів, та швидко створювати звіти із різними метриками що досліджуються. При аналізі багатьох зображень система створює файл із результатами. У результаті також обчислюється процентне відношення правильно проаналізованих зображень – точність роботи алгоритму.

3.6. Висновки

У цьому розділі розглянуто основні технічні рішення, що лежать в основі розробки прототипу програмної системи для дослідження алгоритму виявлення дефектів з допомогою кумулятивних відображень на зображеннях металів.

Обґрунтовано вибір алгоритму аналізу кумулятивних гістограм, який забезпечує високу швидкість, універсальність, зручність та масштабування. Даний алгоритм є перспективним для дослідження, тому розробка прототипу програмного забезпечення є виправданою.

Вибір мови C# та платформи Windows Forms пояснюється потребою в розпаралеленні, та простоті створення інтерфейсу. Розвинена екосистема .NET дозволяє швидко та ефективно проводити розробку додатка, а її хороша документація дозволяє швидко знаходити рішення проблем що виникають.

У цьому розділі детально описано інтерфейс та можливості системи, з використанням знімків екрану та діаграм. Описана взаємодія частин програмного забезпечення між собою, та описано використання програми.

Також описаний процес збирання та підготовки датасету, що включає дві групи зображень із дефектами та без них.

Результатом цього розділу є розроблений прототип програмного забезпечення, що дозволяє досліджувати запропонований алгоритм виявлення дефектів з різними вхідними даними, та з різними параметрами алгоритму. Також застосунок створює базову аналітику, яка дозволяє швидко отримати оцінку роботи алгоритму.

РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО АЛГОРИТМУ

4.1. Вибір метрик

Для оцінки ефективності роботи алгоритму виявлення дефектів застосовуються наступні метрики:

1. **Precision (Точність)** – яка частка передбачених дефектів справді є дефектами.
2. **Recall (Повнота)** – яка частка реальних дефектів була виявлена.
3. **True Negative Rate (TNR)** – відображає, скільки дефектних зображень правильно класифіковано.
4. **True Positive Rate (TPR) -** відображає, скільки бездефектних зображень правильно класифіковано як такі.
5. **False Positive Rate (FPR)** – показує, скільки помилкових спрацьовувань допускає алгоритм (коли бездефектне зображення зараховано до дефектних).
6. **False Negative Rate (FNR)** – вказує, скільки реальних дефектів алгоритм не помітив.

Важливим є також вимірювання швидкодії та стабільності результатів при зростанні обсягу зображень. Оскільки мета – отримати не лише точну, а й швидку систему, тому усі метрики варто розглядати комплексно.

4.2. Тестування швидкодії роботи алгоритму

Першим етапом тестування є оцінка швидкодії обраного алгоритму на різних обсягах даних. Основна увага приділяється часу обробки одного зображення та дослідженню часу послідовної роботи алгоритму.

Тестова система ноутбук з 6-ядерним, 6-поточним Ryzen 5 4500U та 8 гігабайтами оперативної пам’яті, та вбудованим графічним ядром Vega 6.

Вимірювання проводились на зображеннях із двох датасетів окремо, щоб визначити швидкодію на зображеннях з різними розширеннями.

Зображення із датасету NEU-DET з розширенням 200x200 було проаналізовано з допомогою алгоритму. Для дослідження було проведено 10 експериментів з різними зображеннями та по 3 експерименти на кожне зображення а потім результати усереднюються для точнішої оцінки та зведення до мінімуму зовнішніх впливів. Під час роботи максимальне навантаження на оперативну пам’ять що було зафіксовано з допомогою інструментів Visual Studio - 76 мб, що є дуже хорошим результатом та означає що алгоритм не використовує багато пам’яті для роботи. Але час роботи алгоритму при аналізі інтенсивностей від 0 до 255 на зображені 200x200 у середньому вийшов 723мс що дуже багато, та не витримує конкуренції з альтернативними методами.

Аналогічно проаналізовані зображення із датасету GC10-DET з розширенням 2048x1000. Під час роботи максимальне навантаження на оперативну пам’ять що було зафіксовано з допомогою інструментів Visual Studio - 78 мб, що є дуже хорошим результатом та означає що алгоритм не використовує багато пам’яті для обробки навіть великих зображень. Але час роботи алгоритму при аналізі інтенсивностей від 0 до 255 на зображені 2048x1000 у середньому вийшов 25147мс що дуже багато, та не витримує конкуренції з альтернативними методами, але вони не пропонують можливості аналізу таких великих зображень.

4.3. Тестування розпаралелення та порівняння з іншими алгоритмами

Оскільки алгоритм кумулятивних гістограм, а також інші компоненти системи можуть бути доволі просто розподілені по кількох потоках,

важливо перевірити, наскільки ефективно використовується багато ядерність процесора.

Для цього у програмі виставили роботу у 6 потоках та провели дослідження швидкодії, аналогічно як і в попередньому розділі експерименти проводились із двома датасетами окремо із 10 зображеннями з кожного датасету і по 3 експерименти на зображення.

При аналізі швидкодії на датасеті NEU-DET отримано середню швидкість обробки зображення 137.8мс що приблизно у 6 разів швидше, та уже дозволяє конкурувати із деякими алгоритмами. Також було проведено дослідження із 2 потоками та із 4 потоками. Отримано час обробки при роботі з 2 потоками – 385мс та при роботі із 4 потоками – 184мс. Ці результати показують що алгоритм дуже добре розпаралелюється і при наявності більшої кількісті ядр процесора можна добитись ще більшого майже лінійного пришвидшення. Обмеження по розпалаленню є кількість рядків чи стовбців у зображенні, тому обчислення можна дуже зручно розпаралелювати на відео мапі.

При аналізі швидкодії на датасеті GC10-DET при обробці у 6 потоках отримано середню швидкість обробки зображення 4972.8мс, що досить багато, але уже значно менше. Враховуючи то що інші алгоритми або розбивають зображення на менші, або його просто стискають то мати можливість обробити його повністю є корисною функцією, адже стиснути чи розділити на менші можна завжди.

Після отримання результатів їх порівняно із даними із робіт [10,11,12,18] та зведено у таблиці 4.1.

Таблиця 4.1

Порівняння швидкодії алгоритмів

	SSD	Faster-RCNN	YOLO-V2	YOLO-V3	Досліджуваний алгоритм
NEU-DET	29ms	37ms	7.91ms	15.75ms	137.8ms
GC10-DET	29ms	43ms	78.01ms	86.80ms	4972.8ms

Також з [16,18] було порівняно швидкість різних методів, але тільки на наборі даних NEU-DET, та розміщено всі результати в таблицю 4.2.

Таблиця 4.2

Порівняння швидкодії алгоритмів

Назва метода	Час роботи (ms)
ViT-01[7]	104.5
ViT-02[7]	79.5
ResNet-50[7]	108.2
Inception-V3[7]	107.5
EfficientNet-B0[7]	63.2
Faster R-CNN[8]	62.1
SegNet[8]	46.4
PSPNet[8]	42.7
RefineNet[8]	53.4
AGCN[8]	36.4
Досліджуваний алгоритм	137.8

Швидкодія досліджуваного алгоритму при розпаралеленні на 6 потоків нижча ніж у інших методів, але якщо врахувати що аналоги запускались на значно потужніших системах, то результат є дуже хорошим.

4.4. Тестування точності виявлення дефектів

Для дослідження точності роботи алгоритму проводиться серія експериментів. Ці експерименти мають на меті обчислити точність роботи алгоритму виявлення дефектів.

Методика проведення експериментів наступна:

1. Сформувати тестову вибірку із двох категорій зображень:
   - З дефектами (з префіксом у назві файлу «defected_»), що включає зображення різних типів дефектів (тріщини, плями корозії, вибоїни тощо).
   - Без дефектів, отримані шляхом вирізання частин зображень без дефектів із вихідних великих знімків.
2. Проведення аналізу:
   1. Завантажити підготовлені зображення в систему.
   2. Запустити автоматичний аналіз у програмі з різними параметрами алгоритму (порогові значення інтенсивності, кількість інтервалів тощо).
   3. Зафіксувати результати аналізу у файлі CSV, який містить такі дані для кожного зображення:
   4. Назва файлу (Filename)
   5. Реальний статус наявності дефекту (HasDefectInName)
   6. Передбачений статус алгоритмом (DetectedByAlgorithm)
   7. Коректність результату (IsCorrect)
3. Проведення серії експериментів - кожен експеримент повторити мінімум 3 рази для уникнення випадкових коливань після цього усереднити отримані значення метрик. Для детального аналізу роботи алгоритму використовуються такі метрики:

Precision (точність): визначає, яку частку зображень, класифікованих як дефектні, справді мають дефекти (формула 4.1).

$$Precision = \frac{TP}{TP+FP} \quad (4.1)$$

Recall (повнота): визначає, яку частку реальних дефектних зображень правильно визначено алгоритмом (формула 4.2).

$$Recall = \frac{TP}{TP+FN} \quad (4.2)$$

F1-score (гармонічне середнє між Precision та Recall): загальна оцінка якості класифікації (формула 4.3).

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} \quad (4.3)$$

Accuracy (загальна точність): загальна кількість правильно класифікованих зображень по відношенню до всіх зображень (формула 4.4).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (4.4)$$

, де:

TP (True Positive) — дефект правильно виявлений;

TN (True Negative) — відсутність дефекту правильно визначена;

FP (False Positive) — дефект помилково виявлений;

FN (False Negative) — дефект не виявлений алгоритмом.

4. План експериментів:

   Планується виконати щонайменше 3 експериментальні серії:

   - Тестування на датасеті малих зображень (200x200 пікселів, NEU-DET):

     Обсяг вибірки: 300 зображень (250 з дефектами та 50 без дефектів, адже із малих зображень складно вирізати бездефектні частини). Кількість повторень експерименту: 3 (для усереднення результатів).

   - Тестування на датасеті великих зображень (2048x1000 GC10-DET):

     Обсяг вибірки: 100 зображень (70 з дефектами та 30 без дефектів).

   - Змішана вибірка - зображення з різними розмірами, різними типами дефектів та різними умовами зйомки.

Обсяг вибірки: 100 зображень (50 дефектних, 50 без дефектів).

Після проведення кожного етапу експериментів результати узагальнюються у таблицях, а також проводиться порівняння точності роботи запропонованого алгоритму з відомими аналогами.

5. Документування результатів - усі результати тестування зберігаються у структурованому вигляді:

Файл формату CSV із детальною статистикою по кожному зображенню.

Таблиці з підрахованими метриками (Precision, Recall, F1-score, Accuracy).

Графіки залежності точності від розміру зображень, відносного розміру дефекту, параметрів алгоритму.

Результати будуть використані для подальшого аналізу ефективності алгоритму та його вдосконалення.

6. На основі отриманих результатів планується:
   - Оптимізувати параметри алгоритму (порогові значення).
   - Впровадити додаткові методи для локалізації дефектів.
   - Провести порівняння ефективності розробленого рішення з іншими методами, зокрема з використанням глибинного навчання.

4.4.1 Тестування на малих зображеннях

Було вибрано із зображень з датасету NEU-DET 50 зображень, або вирізано ділянку зображення що можна класифікувати як бездефектні, адже видимих дефектів на них немає, або вони дуже слабко виражені.

Проведено по 3 експерименти з різними порогами виявлення дефектів, які змінювались у межах від 3% до 25% з перемінним кроком (табл. 4.3). У результаті експериментів виявлено, що у всіх 3 експериментах з однаковими параметрами алгоритму були ідентичні результати, алгоритм

однаково опрацьовує одне зображення, тому у таблиці наведено результати по 3 експериментах у одному рядку.

Таблиця 4.3.

Дослідження точності на зображеннях малих розширень

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1-3	3%	250	50	250	0	50	0
4-6	5%	261	39	250	11	39	0
7-9	8%	284	16	250	32	16	0
10-12	12%	279	21	229	50	0	21
13-15	15%	199	101	149	50	0	101
16-18	18%	155	145	105	50	0	145
19-21	20%	126	274	76	50	0	174
22-24	25%	86	214	36	50	0	214

У результаті цих експериментів можна замітити що при збільшенні порогу виявлення дефектів точність зростає від 3% до 12%, адже ми правильніше класифікуємо бездефектні зображення. Після цього вона спадає досить різко, так як усі бездефектні зображення уже правильно визначаються, а збільшуючи поріг ми допускаємо наявність більш серйозних дефектів і значно починає падати кількість правильно виявлених дефектів на зображення із дефектами.

Для подальшого дослідження вирішено провести ще серію експериментів між погром 5% та 12% із кроком в 1%, щоб визначити якої максимальної точності можливо досягнути на даній вибірці. Результати цих досліджень зведено у таблицю 4.4.

Таблиця 4.4.

Детальніше дослідження точності на зображеннях малих розширень

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1	3%	250	50	250	0	50	0
2	4%	252	48	250	2	48	0
3	5%	261	39	250	11	39	0
4	6%	270	30	220	20	30	0
5	7%	281	19	250	31	19	0
6	8%	284	16	250	34	16	0
7	9%	288	12	250	38	12	0
8	10%	294	6	249	45	5	1
9	11%	292	8	243	49	1	7
10	12%	279	21	229	50	0	21

Із експериментів видно що найкращий результат досягнуто при порогу виявлення у 10% - 294 правильно класифікованих зображення. Для цього експерименту вирахуємо усі метрики:

$$Precision = 249/(249 + 5) = 0.9803 = 98.03\%$$

$$Recall = 249/(249 + 1) = 0.996 = 99.6\%$$

$$F1 - score = 2 * (0.9803 * 0.996)/(0.9803 + 0.996) = 0.9881 = 98.81\%$$

$$Accuracy = (249 + 45)/300 = 0.98 = 98\%$$

Результати дуже хороші, але із експериментів видно, що для кінцевої системи необхідно підібрати поріг, який буде вважатись адекватним для даного виробництва, але менше 5% використовувати не рекомендується, адже у такому випадку відсіються дуже багато достатньо чистих зображень.

4.4.2 Тестування на великих зображеннях

Було вибрано із зображень з датасету GC10-DET 30 зображень що можна класифікувати як бездефектні, або вирізано частину зображення де візуально дефект не спостерігається і збережено як окреме бездефектне зображення.

Проведено з різними порогами виявлення дефектів, які змінювались у межах від 5% до 20% з перемінним кроком. Верхня та нижня границя порогу виявлення дефектів трохи зміщені, адже з попередніх експериментів виявлено, що при 3% жодного зображення не було класифіковано як бездефектне, а при 25% дуже велика кількість дефектних зображень класифікується як бездефектне. Результати зведені у таблицю 4.5.

Таблиця 4.5.

Дослідження точності на зображеннях великих розширень

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1	5%	70	30	70	0	30	0
2	8%	73	27	70	3	27	0
3	12%	78	22	70	8	22	0
4	15%	86	14	69	17	13	1
5	18%	76	24	57	19	11	13
6	20%	69	31	48	21	9	22

У результаті цих експериментів видно що кількість правильно виявлених зростає до 15% а потім починає спадати, адже дефектні зображення починають вважатись бездефектними. Ці зображення з великим розширенням, та на них більше перепад від освітлення, тому й поріг для цих зображень більший.

Для подальшого дослідження вирішено провести детальніші експерименти із кроком в 1% від 12% до 18%, щоб визначити якої максимальної точності можна досягнути на даній вибірці. Результати дослідження зведено у таблицю 4.6

Таблиця 4.6.

Детальніше дослідження точності на зображеннях великих розширень

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1	12%	78	22	70	8	22	0
2	13%	82	18	70	12	18	0
3	14%	85	15	70	15	15	0
4	15%	86	14	69	17	13	1
5	16%	81	19	64	17	13	6
6	17%	79	21	61	18	12	9
7	18%	76	24	57	19	11	13

Із експериментів видно що найкращий результат досягнуто при порогу виявлення у 15% - 86 правильно класифікованих зображення. Для цього експерименту вирахуємо усі метрики:

$$Precision = 69/(69 + 13) = 0{,}8415 = 84.15\%$$

$$Recall = 69/(69 + 1) = 0{,}9857 = 98.57\%$$

$$F1 - score = 2 * (0{,}9857 * 0{,}8414)/(0{,}9857 + 0{,}8414) = 0{,}9078 = 90.78\%$$

$$Accuracy = (69 + 17)/100 = 0.86 = 86\%$$

Результати все ще хороші, але видно що на великих зображеннях точність зменшилась, та оптимальний поріг для виявлення дефектів більший на 5%, це можна пояснити тим що зображення більше, область матеріалу що фотографується більша, а отже і перепади у яскравості більші.

Для кращих результатів потрібно удосконалювати освітлення, щоб велика поверхня матеріалу освітлювалась рівномірно.

4.4.3 Тестування на змішаній вибірці

Із двох тестувальних вибірок вибрано 50 дефектних та 50 бездефектних зображення навмання. Верхню та нижню межу порогу виявлення дефектів та крок залишили таким самим, Результати зведено в таблицю 4.7.

Таблиця 4.7.

Дослідження точності на різних зображеннях

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1	5%	56	44	50	6	44	0
2	8%	69	31	50	19	31	0
3	12%	80	20	50	30	20	0
4	15%	83	17	45	38	12	5
5	18%	75	25	35	40	10	15
6	20%	74	26	32	42	8	18

У результаті цих експериментів видно що кількість правильно виявлених зростає до 15% а потім починає спадати. Аналогічно як із іншими вибірками дефектні зображення починають вважатись бездефектними, а бездефектні зображення уже всі правильно класифіковані.

Для подальшого дослідження вирішено провести детальніші експерименти із кроком в 1% від 12% до 18%, щоб визначити якої максимальної точності можна досягнути на даній вибірці. Результати дослідження зведено у таблицю 4.8.

Таблиця 4.8.

Детальніше дослідження точності на різних зображеннях

№	Поріг виявлення	Кількість правильно виявлених	Кількість неправильно виявлених	TP	TN	FP	FN
1	12%	80	20	50	30	20	0
2	13%	83	17	49	34	16	1
3	14%	82	18	46	36	14	4
4	15%	83	17	45	38	12	5
5	16%	77	23	39	38	12	11
6	17%	77	23	38	39	11	12
7	18%	75	25	35	40	10	15

У результаті експерименту видно що при 13% порозі виявлення та при 15% порозі кількість правильно визначених однакова і рівна 83 зображенням. Різниця є тільки у тому що при 13% більше дефектних зображень виявлено, але також і більше бездефектних оприділено як дефектні. Підрахуємо для обох випадків метрики.

13%:

$$Precision = 49/(49+16) = 0{,}7538 = 75.38\%$$

$$Recall = 49/(49+1) = 0{,}98 = 98\%$$

$$F1-score = 2*(0{,}7538*0{,}98)/(0{,}7538+0{,}98) = 0{,}8521 = 85.21\%$$

$$Accuracy = (49+34)/100 = 0.83 = 83\%$$

15%:

$$Precision = 45/(45+12) = 0{,}7894 = 78.94\%$$

$$Recall = 45/(45+5) = 0{,}9 = 90\%$$

$$F1-score = 2*(0{,}9*0{,}7894)/(0{,}9+0{,}7894) = 0{,}8411 = 84.11\%$$

$$Accuracy = (45+38)/100 = 0.83 = 83\%$$

З 13% порогом виявлення метрики кращі, тому як результат візьмемо його.

4.5 Висновки

У цьому розділі ми дослідили та порівняли швидкодію роботи запропонованого алгоритму, та дослідили точність роботи при виявленні різноманітних дефектів на зображеннях поверхонь матеріалів.

У результаті порівнянь ми виявили, що швидкодія при роботі на 6 ядерному процесорі достатня для обробки малих зображень до 200x200 пікселів, але для обробки великих зображень, необхідно більшу потужність комп'ютерного забезпечення. Час обробки малих зображень близький до часу обробки нейронними мережами, що робить алгоритм конкуренто спроможним. Але при обробці великих зображень більшість алгоритмів нейронних мереж працює значно швидше, адже використовує метод зменшення розширення, що впливає на якість обробки зображення, у той час як запропонований алгоритм працює із зображенням в оригінальній якості та не втрачає деталей при аналізі.

Також у результаті дослідження розпаралелення ми дійшли висновку, що даний алгоритм дуже добре піддається розпаралеленню та для забезпечення хорошої швидкодії можна спробувати розпаралелити обчислювання на графічному прискорювачі, адже усі обчислення не вимагають складних операцій. Таке удосконалення теоретично дозволить наблизити час обробки зображень до рівня нейронних мереж, які також для обчислень використовують графічні прискорювачі.

У результаті дослідження точності роботи алгоритму при виявленні наявності дефектів, ми отримали хороші результати при правильному налаштуванні алгоритму під систему освітлення матеріалів. Точність досягнула 98% у найкращому випадку, при обробці зображень малого розширення, що є чудовим результатом. Але при роботі з зображеннями великої роздільної здатності, ми замітили що через нерівномірність освітлення на великій площі матеріалу ускладняється робота алгоритму та

зменшується точність до 86% процентів, що досі є хорошим результатом. Аналогічна проблема коли ми тестували на вибірці з двох датасетів у яких різні системи освітлення, але навіть у таких умовах удалось добитись точності у 83% що ми вважаємо дуже хорошим результатом.

Дослідження показали, що при добре розробленій системі освітлення та подальшому розпаралеленні, алгоритм дозволяє отримувати точні результати доволі швидко.

ВИСНОВКИ

У ході виконання роботи було виконано такі пункти:

1. Проведено аналіз існуючих підходів та технологій виявлення дефектів, що дозволило виявити переваги та недоліки різних існуючих методів, та підтвердило актуальність використання статистичних методів обробки зображень.
2. Уперше запропоновано та теоретично обґрунтовано алгоритм виявлення дефектів поверхонь металевих матеріалів з використанням методу розподіленої візуалізації кумулятивних гістограм інтенсивності зображення.
3. Розроблено прототип програмної системи, який реалізує алгоритм аналізу кумулятивних гістограм на мові програмування C# та технології Windows Forms, що забезпечило зручну інтеграцію різних компонентів, високу швидкість розробки різних прототипів та можливість розпаралелення обчислень. Використання середовища розробки Visual Studio забезпечило ефективну розробку, відлагодження та масштабування програми.
4. Досліджено та проведено розпаралелення запропонованого алгоритму, у результаті якого було виявлено, що досліджуваний алгоритм добре масштабується за рахунок ефективного використання багатьох потоків, що значно зменшує загальний час обробки зображення. Для досягнення кращих результатів запропоновано провести додаткові дослідження із використанням графічних прискорювачів. Базуючись на отриманих результатах збільшення швидкодії та теоретичних основах роботи алгоритму є усі підстави вважати, що при використанні багатоядерних процесорів або графічних прискорювачів можна суттєво покращити швидкодію, що робить запропонований підхід перспективним для використання в умовах реального виробництва.

5. Під час тестування алгоритму на двох різних наборах зображень було досягнуто високої точності виявлення дефектів. А саме: до 98% точності виявлення наявності дефектів на невеликих зображеннях з розширенням 200x200 пікселів, до 86% точності виявлення дефектів на зображеннях великих ділянок металу з великим розширенням, та до 93% точності при використанні змішаної вибірки із різними зображеннями. Також було встановлено, що точність залежить від налаштувань параметрів алгоритму, а також від рівномірності освітлення поверхні матеріалу на зображенні.

У подальшому дослідженні даного метода необхідно буде, дослідити покращення паралельної обробки зображення алгоритмом, реалізувати та дослідити метод виділення дефектів із зображення та дослідити і вибрати найкращий варіант для подальшої класифікації виділених дефектів.

Отже, розроблений алгоритм та програмна реалізація підтвердили свою ефективність у вирішенні завдання автоматизованого контролю якості поверхонь металевих матеріалів, демонструючи добру точність та потенціал для подальшого дослідження, вдосконалення та впровадження у виробництво.

СПИСОК ЛІТЕРАТУРИ

1. Сторожик Д. ТЕХНОЛОГІЇ ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ КОМПЛЕКСУВАННЯ ДАНИХ (Огляд) [Електронний ресурс] / Д. Сторожик, А. Протасов // Технічна діагностика та неруйнівний контроль. – 2022. – Т. 2022, № 4. – С. 17–26. – Режим доступу: https://patonpublishinghouse.com/tdnk/pdf/2022/tdnk202204all.pdf#page=17 (дата звернення: 17.12.2023). – Назва з екрана.

2. Qiu L. A high-efficiency fully convolutional networks for pixel-wise surface defect detection [Electronic resource] / Lingteng Qiu, Xiaojun Wu, Zhiyang Yu // IEEE access. – 2019. – Vol. 7. – P. 15884–15893. – Mode of access: https://doi.org/10.1109/access.2019.2894420 (date of access: 17.12.2023). – Title from screen.

3. Steel surface defect recognition: a survey [Electronic resource] / Xin Wen [et al.] // Coatings. – 2022. – Vol. 13, no. 1. – P. 17. – Mode of access: https://doi.org/10.3390/coatings13010017 (date of access: 17.12.2023). – Title from screen.

4. A review of non-destructive testing (NDT) techniques for defect detection: application to fusion welding and future wire arc additive manufacturing processes [Electronic resource] / Masoud Shaloo [et al.] // Materials. – 2022. – Vol. 15, no. 10. – P. 3697. – Mode of access: https://doi.org/10.3390/ma15103697 (date of access: 17.12.2023). – Title from screen.

5. Кучеренко О. Ахроматизація та атермалізація об’єктивів інфрачервоної техніки [Електронний ресурс] / О. Кучеренко, О. Муравйов, В. Тягур // Наукові вісті НТУУ "КПІ". – 2012. – С. 115–117. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/36927/1/2012-5-15.pdf. – Назва з екрана.

6. Automated product boundary defect detection based on image moment feature anomaly [Electronic resource] / Yeping Peng [et al.] // IEEE access. – 2019. – Vol. 7. – P. 52731–52742. – Mode of access:

https://doi.org/10.1109/access.2019.2911358 (date of access: 17.12.2023). – Title from screen.

7. Баган Я. А. Алгоритми текстурної сегментації на основі статистичних ознак [Електронний ресурс] / Ярослав Андрійович Баган. – 2020. – Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/40538/1/Баган_МР_2020.pdf. – Назва з екрана.

8. Чмутов Ю. Розроблення методу класифікації зображень із використанням кластерних структур для швидкого пошуку даних [Електронний ресурс] / Ю. Чмутов. – 2021. – Режим доступу: https://openarchive.nure.ua/items/d24e02b3-9e57-42e5-b055-f873bdaf6139. – Назва з екрана.

9. A light-weighted CNN model for wafer structural defect detection [Electronic resource] / Xiaoyan Chen [et al.] // IEEE access. – 2020. – Vol. 8. – P. 24006–24018. – Mode of access: https://doi.org/10.1109/access.2020.2970461 (date of access: 17.12.2023). – Title from screen.

10. Стешенко Я. В. Удосконалення алгоритмів сегментації зображень поверхневих дефектів металевих виробів [Електронний ресурс] : Магістерська дисертація : 151 / Стешенко Ярослав Віталійович. – Київ, 2023. – 111 с. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/56273/1/Steshenko_magistr.pdf. – Назва з екрана.

11. Duan C. Two-Stream convolutional neural network based on gradient image for aluminum profile surface defects classification and recognition [Electronic resource] / Chunmei Duan, Taochuan Zhang // IEEE access. – 2020. – Vol. 8. – P. 172152–172165. – Mode of access: https://doi.org/10.1109/access.2020.3025165 (date of access: 17.12.2023). – Title from screen.

12. Simões Hoffmann L. F. Detection of liner surface defects in solid rocket motors using multilayer perceptron neural networks [Electronic resource] / Luiz

Felipe Simões Hoffmann, Francisco Carlos Parquet Bizarria, José Walter Parquet Bizarria // Polymer testing. – 2020. – Vol. 88. – P. 106559. – Mode of access: https://doi.org/10.1016/j.polymertesting.2020.106559 (date of access: 17.12.2023). – Title from screen.

13. Smith A. D. Vision transformers for anomaly detection and localisation in leather surface defect classification based on low-resolution images and a small dataset [Electronic resource] / Antony Douglas Smith, Shengzhi Du, Anish Kurien // Applied sciences. – 2023. – Vol. 13, no. 15. – P. 8716. – Mode of access: https://doi.org/10.3390/app13158716 (date of access: 17.12.2023). – Title from screen.

14. Zhang C. Multilayer feature extraction of AGCN on surface defect detection of steel plates [Electronic resource] / Chi Zhang, Jian Cui, Wei Liu // Computational intelligence and neuroscience. – 2022. – Vol. 2022. – P. 1–13. – Mode of access: https://doi.org/10.1155/2022/2549683 (date of access: 17.12.2023). – Title from screen.

15. Novel framework for optical film defect detection and classification [Electronic resource] / Ngoc Tuyen Le [et al.] // IEEE access. – 2020. – Vol. 8. – P. 60964–60978. – Mode of access: https://doi.org/10.1109/access.2020.2982250 (date of access: 17.12.2023). – Title from screen.

16. Surface defect detection of industrial components based on vision [Electronic resource] / Chen Z. [et al.] // Scientific Reports. 2023. – Vol. 13 - Mode of access: https://doi.org/10.1038/s41598-023-49359-9 - Title from screen.

17. Metal surface defect detection based on improved YOLOv5 [Electronic resource] / Zhou C. [et al.] // Scientific Reports - 2023. – Vo. 13, no. 15 - Mode of access: https://doi.org/10.1038/s41598-023-47716-2 - Title from screen.

18. Deep Metallic Surface Defect Detection: The New Benchmark and Detection Network [Electronic resource] / Lv X. [et al.] // Sensors 2020. – Vol. 20, no. 6. – P. 1562 - Mode of access: https://doi.org/10.3390/s20061562 - Title from screen.

19. Feature optimization-guided high-precision and real-time metal surface defect detection network. [Electronic resource] / Chan, S. [et al.] / Sci Rep 2024, Vo. 14 - Mode of access: https://doi.org/10.1038/s41598-024-83430-3

20. Improved YOLOv5 Network for Steel Surface Defect Detection. [Electronic resource] / Bo H. [et al.] / Metals 2023, Vo. 13, no. 8, - P.1439 - Mode of access: https://doi.org/10.3390/met13081439

21. Pratt W (2007) Digital image processing. Wiley-Interscience.

ДОДАТКИ

Додаток А

Побудова кумулятивного відображення

```
            var test = Stopwatch.StartNew();
            int width = _image.Width;
            int height = _image.Height;
            int intensityIntervals = (int)numericUpDown1.Value;
            int minimumIntensity = (int)numericUpDown2.Value;
            int maximumIntensity = (int)numericUpDown3.Value;
            int intensityRange = (maximumIntensity-minimumIntensity) /
intensityIntervals;

            if (isVerticalScan)
            {
                _histogramImage = new Bitmap(width, (maximumIntensity -
minimumIntensity));


                    Color[,] Results;
                    Results = calcPositionInterval(0, width, _image);

                    for (int x = 0; x < Results.GetLength(0); x++)
                    {
                        for (int y = 0; y < Results.GetLength(1) - 1; y++)
                        {
                            _histogramImage.SetPixel(x, y, Results[x, y]);
                        }
                    }

                    pictureBox2.Image = _histogramImage;
                    test.Stop();
                    toolStripStatusLabel3.Text = "Час обчислення: " +
test.ElapsedMilliseconds;

                    return;
                    for (int x = 0; x < width; x++)
                    {
                        for (int i = 0; i < intensityIntervals; i++)
                        {
                            int[] columnHistogram = new int[intensityRange];
                            double[] columnCumulativeHistogram = new
double[intensityRange];
                            int pixelCount = 0;
                            // Перебираємо рядки
                            for (int y = 0; y < height; y++)
                            {
                                // Отримуємо кольор пікселя на позиції (x, y)
                                Color pixelColor = _image.GetPixel(x, y);

                                // Обчислюємо яскравість пікселя (середнє значення
кольорів)
                                int brightness = (int)(pixelColor.R + pixelColor.G
+ pixelColor.B) / 3;

                                if (brightness >= minimumIntensity && brightness <=
maximumIntensity
                                    && brightness >= i * intensityRange &&
brightness < (i + 1) * intensityRange)
                                {
                                    columnHistogram[brightness % intensityRange]++;
                                    pixelCount++;
```

```
                            }
                            // Оновлюємо гістограму для відповідної яскравості

                        }
                        double accumulator = 0;
                        for (int brightness = 0; brightness < intensityRange;
brightness++)
                        {
                            if (pixelCount == 0)
                            {
                                _histogramImage.SetPixel(x, brightness + i *
intensityRange, Color.FromArgb(0, 0, 0));
                                continue;
                            }
                            accumulator += columnHistogram[brightness];
                            columnCumulativeHistogram[brightness] =
(accumulator) / pixelCount;
                            int colorValue =
(int)(columnCumulativeHistogram[brightness] * 255);
                            _histogramImage.SetPixel(x, brightness + i *
intensityRange, Color.FromArgb(colorValue, colorValue, colorValue));
                        }
                    }

                    toolStripProgressBar1.Value = (int)((double)x / width) *
100;
                }
            }
        }
        else
        {
            _histogramImage = new Bitmap((maximumIntensity - minimumIntensity),
height);

            for (int y = 0; y < height; y++)
            {
                for (int i = 0; i < intensityIntervals; i++)
                {
                    int[] columnHistogram = new int[intensityRange];
                    double[] columnCumulativeHistogram = new
double[intensityRange];
                    int pixelCount = 0;
                    // Перебираємо рядки
                    for (int x = 0; x < width; x++)
                    {
                        // Отримуємо колір пікселя на позиції (x, y)
                        Color pixelColor = _image.GetPixel(x, y);

                        // Обчислюємо яскравість пікселя (середнє значення
кольорів)
                        int brightness = (int)(pixelColor.R + pixelColor.G +
pixelColor.B) / 3;

                        if (brightness >= minimumIntensity && brightness <=
maximumIntensity
                            && brightness >= i * intensityRange && brightness <
(i + 1) * intensityRange)
                        {
                            columnHistogram[brightness % intensityRange]++;
                            pixelCount++;
                        }
                        // Оновлюємо гістограму для відповідної яскравості

                    }
                    double accumulator = 0;
```

```
                    for (int brightness = 0; brightness < intensityRange;
brightness++)
                    {
                        if (pixelCount == 0)
                        {
                            _histogramImage.SetPixel(brightness + i *
intensityRange, y, Color.FromArgb(0, 0, 0));
                            continue;
                        }
                        accumulator += columnHistogram[brightness];
                        columnCumulativeHistogram[brightness] = (accumulator) /
pixelCount;
                        int colorValue =
(int)(columnCumulativeHistogram[brightness] * 255);
                        _histogramImage.SetPixel(brightness + i *
intensityRange, y, Color.FromArgb(colorValue, colorValue, colorValue));
                    }
                }

                toolStripProgressBar1.Value = (int)((double)y / height) * 100;
            }
        }

        pictureBox2.Image = _histogramImage;
        test.Stop();
        toolStripStatusLabel3.Text = "Час обчислення:
"+test.ElapsedMilliseconds;

    }
```

Додаток Б

Розпаралелення обчислень

```
      private Color[,] calcPositionInterval(int startPos, int endPos, Bitmap
imageCopy )
      {
          int width = imageCopy.Width;
          int height = imageCopy.Height;
          int intensityIntervals = (int)numericUpDown1.Value;
          int minimumIntensity = (int)numericUpDown2.Value;
          int maximumIntensity = (int)numericUpDown3.Value;
          int intensityRange = (maximumIntensity - minimumIntensity) /
intensityIntervals;
          Color[,] res = new Color[endPos-startPos,256];
          for(int x = 0; x < endPos - startPos; ++x)
          {
              for (int y=0; y < 256; ++y)
              {
                  res[x, y] = Color.Black;
              }
          }
          for (int x = startPos; x < endPos; x++)
          {
              for (int y = 0; y < height; ++y)
              {
                  Color pixelColor = imageCopy.GetPixel(x, y);
                  res[x - startPos, (int)(pixelColor.R + pixelColor.G +
pixelColor.B) / 3] = Color.White;

              }
          }
          return res;
      }

Color[][,] paralelResults = new Color[threadCount][,];
      Thread[] threads = new Thread[threadCount];
      int elemsForThread = width / threadCount;
      for (int i = 0; i < threadCount; ++i)
      {
          int index = i;
          Bitmap imgClone = (Bitmap)_image.Clone();
          if (i == threadCount - 1)
          {
              threads[i] = new Thread(() =>
              {
                  paralelResults[index] = calcPositionInterval(index *
elemsForThread, width, imgClone);
              });
          }
          else
          {
              threads[i] = new Thread(() =>
              {
                  paralelResults[index] = calcPositionInterval(index *
elemsForThread, (index + 1) * elemsForThread, imgClone);
              });
          }
          threads[i].Start();
      }
      foreach (Thread thread in threads)
      {
          thread.Join();
      }
```

```
        test.Stop();
        toolStripStatusLabel3.Text = "Час обчислення: " + test.ElapsedMilliseconds;
        for (int i = 0; i < threadCount; i++)
        {
            for(int x = 0; x < paralelResults[i].GetLength(0); x++)
            {
                for(int y= 0; y< paralelResults[i].GetLength(1)-1; y++)
                {
                    _histogramImage.SetPixel(i* elemsForThread + x, y,
paralelResults[i][x, y]);
                }
            }
        }
```