Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДІВ АНАЛІЗУ ТЕКСТУРНИХ ЗОБРАЖЕНЬ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-22-3
Ляхов П.Л.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Яковлева О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ляхову Павлу Леонідовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів аналізу текстурних зображень для вирішення задачі класифікації
затверджена наказом по університету від 3 листопада 2023 року № 1280См
2. Термін подання студентом роботи до екзаменаційної комісії 30 грудня 2023 р.
3. Вихідні дані до роботи Математичні моделі аналізу текстурних зображень для вирішення задачі класифікації, Python, Google Drive, Google Colab, scitic-image, теоретичні відомості про методи аналізу текстурних зображень.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд сучасного стану питання аналізу текстурних зображень.
2. Текстурні зображення та задачі, пов'язані з аналізом текстурних зображень.
3. Методи опису текстури.
4. Задача пошуку та класифікації текстурних зображень.
5. Сучасні програмні засоби для вирішення задач комп'ютерного зору.
6. Математична модель методу лінійних бінарних шаблонів.
7. Математична модель методу, на основі матриці збігів.
8. Класіфікація на основі нейроних мереж, донавчання моделі
9. Оцінка класифікатора за текстурними ознаками.
10. Алгоритм пошуку текстурних зображень.
11. Створення датасету текстур.
12.Реалізація методу бінарних шаблонів.
13.Реалізація методу, на основі матриці збігів.

14. Оцінка у порівняльному аспекті якості та швидкодії пошуку текстур за розглянутими методами.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми пошуку текстурних зображень, постановка задачі, створення датасету текстур, математична модель методу лінійних бінарних шаблонів, математична модель методу, на основі матриці збігів, критерій схожості зображень, метрики для оцінки якості пошуку за текстурними ознаками, реалізація методу бінарних шаблонів, реалізація методу, на основі матриці збігів, оцінка у порівняльному аспекті якості та швидкодії пошуку текстур за розглянутими методами, висновки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.11.2023	
2	Аналіз завдання, підбір літератури	03.11.23-05.11.23	
3	Аналіз літератури з досліджуваної проблеми	05.11.23-10.11.23	
4	Аналіз технічних засобів	10.11.23-12.11.23	
5	Розробка методу	12.11.23-20.11.23	
6	Програмна реалізація	20.11.23-01.12.23	
7	Оформлення пояснювальної записки	01.12.23-03.12.23	
8	Перевірка на плагіат	03.12.23	
9	Рецензування	28.12.23	
10	Підготовка презентації та доповіді	29.12.23	
11	Занесення роботи в електронний архів	03.01.24	
12	Попередній захист кваліфікаційної роботи	09.01.24	

Дата видачі завдання 03.11.2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Яковлева О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 87 с., 1 табл., 75 рис., 31 джерело.

ТЕКСТУРА, РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ, ПОШУК ЗОБРАЖЕНЬ, МІРА ПОДІБНОСТІ, ЛІНІЙНІ БІНАРНІ ШАБЛОНИ, МАТРИЦЯ ЗБІГІВ, MOBILENETV2, PYTHON, GOOGLE COLAB, MAHOTAS, SCIKITS-IMAGE, OPENCV.

Об’єктом роботи є проблема класифікації та пошуку текстурних зображень. Метою роботи є дослідження методів аналізу текстурних зображень для вирішення задачі класифікації текстур.

Робота присвячена порівнянню точності та швидкодії класифікації на основі класичних методів отримання ознак текстурних зображень з нейромережевим підходом. Як класичні методи були розглянути метод локальних бінарних шаблонів та метод, на основі матриці збігів. Як приклад неромережевого підходу розглядалось донавчання нейронної моделі MobileNet2. Для проведення досліджень створено датасет зображень текстур, де на одному зображенні присутня текстура, що належить до одного класу. Розроблено алгоритм класифікації зображень на основі аналізу текстурних ознак та его програмна реалізація з використання мови Python, онлайн середовища Google Colab та бібліотек Scikits-image, OpenCV.

TEXTURE, IMAGE RECOGNITION, IMAGE SEARCH, SIMILARITY MEASURE, LINEAR BINARY PATTERNS, CO-OCCURRENCE MATRIX, MOBILENETV2, PYTHON, GOOGLE COLAB, MAHOTAS, SCIKITS-IMAGE, OPENCV.

The subject of the study is the problem of classification and texture image retrieval. The aim of the research is to investigate methods for analyzing texture images to address the task of texture classification.

The work is dedicated to comparing the accuracy and efficiency of classification based on classical texture feature extraction methods with a neural network approach. Classical methods such as Local Binary Patterns and Co-occurrence Matrix were considered. As an example of a non-neural network approach, fine-tuning of the MobileNet2 neural model was explored. To conduct the research, a dataset of texture images was created, where each image contains a texture belonging to a specific class. An image classification algorithm based on texture feature analysis was developed, and its software implementation was carried out using the Python language, the online environment Google Colab, and the Scikit-image and OpenCV libraries.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЛБШ – локальні бінарні шаблони

ШІ – штучний інтелект

МЗНМ – мультиколонна згорткова нейронна мережа

ВСТУП

Зацікавленість у комп'ютерному зорі стала однією з піонерських сфер у сфері штучного інтелекту, разом з такими сферами, як автоматизоване доказування теорем та ігри, що вимагають розуму. Концепція першої штучної нейронної мережі, персептрона, була запропонована Френком Розенблаттом, натхненна аналогією з людською сітківкою, і ця концепція була випробувана через задачу розпізнавання символьних зображень.

Проблематика комп'ютерного зору завжди вважалась значущою, проте її складність часто недооцінювалася. У 1966 році Марвін Мінський, один з піонерів штучного інтелекту, прийняв виклик вирішити цю проблему. Не усвідомлюючи повної складності завдання, він доручив його своєму студенту на літній період. У порівнянні, програмування програми для гри у шахи на рівні гросмейстера вимагало значно більше часу. Сьогодні вже ясно, що розробка програми для перемоги у шахах є простішою, ніж створення адаптивної системи управління з підсистемою комп'ютерного зору, здатною розпізнавати фігури на шаховій дошці.

Розвиток у сфері комп'ютерного зору залежить від двох основних чинників: прогресу в теоретичних знаннях, методиках та вдосконаленні апаратного забезпечення. Протягом тривалого періоду академічні дослідження та теоретичні основи опережали практичне застосування цих технологій.

З часом, завдяки удосконаленню процесорів і розвитку цифрових камер, ситуація стала змінюватися. Досягнення певного рівня продуктивності, необхідного для ефективної обробки зображень у прийнятний час, сприяло розробці багатьох програм у сфері комп'ютерного зору. Цей процес є поступовим і триває досі.

Сучасні системи машинного зору використовуються для автоматизації різноманітних виробничих процесів, таких як автоматичне розпізнавання промислових компонентів, їх класифікація, перевірка розмірів, упаковка

товарів, моніторинг їх розміщення в процесах зварювання, точна установка в певному місці та інше.

Системи машинного зору працюють шляхом аналізу двовимірних функцій зображення. Незважаючи на те, що зображення лише частково відтворюють фізичну реальність, вони надають важливу інформацію, яка використовується всіма біологічними системами, включно з людиною.

Інформація про стан поля зору походить від поля яскравості. Це поле, зазвичай неоднорідне і мінливе, у кожному просторовому елементі характеризується яскравістю, кольоровим тоном та насиченістю, які можуть змінюватися з часом.

Головні завдання систем машинного зору в роботизованих технічних системах полягають у визначенні робочого простору, що містить об’єкти для пошуку, виборі фону, розпізнаванні образів та вимірюванні статичних і динамічних характеристик об'єктів на основі їх візуального представлення.

Комплекс методик, що застосовується в процесі аналізу зображень в рамках систем штучного зору, класифікується відповідно до етапів обробки зображення і охоплює три основні категорії.:

– ініційна обробка: Цей етап включає дії такі як усунення шуму, розділення зображення на відповідні частини, і виявлення країв об'єктів;

– розробка деталізованих описів: На цій стадії відбувається визначення характеристичних векторів або характеристик, створення структурних елементів та побудова моделей;

– аналіз та інтерпретація описів: Заключний етап, який передбачає вирішення про відповідність між образами та стандартними моделями, що були сформовані в процесі аналізу, а також оцінка атрибутів об'єктів.

Два останні етапи відносяться до задачі "інтерпретації зображень", яка є ключовою у галузі штучного інтелекту. Сучасні методи обробки та аналізу зображень спираються, з одного боку, на фундаментальні математичні принципи, як-от теорія множин, математичний аналіз, алгебра, статистична теорія прийняття рішень, а з іншого - на можливість моделювати складні

візуальні алгоритми обробки даних за допомогою програмування та дослідження їх характеристик на сучасних комп'ютерах.

Одне з найважливіших завдань у області обробки зображень - знаходження зображення, яке відповідає заданому зразку. Незважаючи на наявність успішних рішень, наразі це завдання вирішується лише для конкретних випадків з певними обмеженнями щодо типу зображення, включаючи геометричні та яскравісні спотворення.

Головна мета дослідження полягає у вивченні методів розпізнавання текстур, заснованих на класичних підходах.

1 СУЧАСНИЙ СТАН ПИТАННЯ АНАЛІЗУ ТЕКСТУРНИХ ЗОБРАЖЕНЬ

1.1 Текстурні зображення та задачі, пов’язані з аналізом текстурних зображень

Текстура відображає розташування кольорів чи рівнів інтенсивності у зображенні. Часто вона зустрічається на зображеннях, що відтворюють природні сцени або сцени зі штучними об'єктами. Елементи, як-от пісок, камінь, трава, листя або цегла, створюють текстурний вміст зображення. На прикладі рисунку 1.1 можна побачити різні типи текстур [1].

а б в г д е

Рисунок 1.1 – Приклади текстур: а – галька; б – поверхня води; в – плями на шкурі леопарда; г – листя; д – текстура цегляної стіни; е – штучно створена текстура

Сучасні підходи до аналізу текстур у зображеннях знаходять застосування в широкому спектрі сфер для виконання різноманітних завдань. Це включає виявлення та ідентифікацію текстурних об'єктів, обчислення кількості осіб у місцях скупчення людей, оцінку площі лісових масивів, визначення збитків від лісових пожеж за допомогою аеро- та космічних знімків, аналіз даних дистанційного зондування планети, обробку біомедичних зображень, а також аналіз текстур у стокових фотографіях.

Попри багаторічні наукові дослідження та розробки в області обробки зображень, багато задач все ще залишаються невирішеними, зокрема, в

аспектах розуміння та аналізу зображень. Це зумовлено складністю задачі розпізнавання візуальних образів, яка є далеко не простою [2].

На даний момент, сфера аналізу зображень, особливо текстурних, активно розвивається. Вивчаються нові методології, використовуються інноваційні алгоритми, та створюються нові бібліотеки з готовими рішеннями для аналізу та обробки зображень.

1.2 Методи опису текстур

На сьогодні існує багато різних підходів і методів для розв'язання задачі розпізнавання, а також численні техніки, які оптимально підготовлюють зображення для розпізнавання шляхом виконання різних процесів обробки зображень.

Процес розпізнавання може відбуватися на основі аналізу інформативних характеристик зображень або самого зображення. Зазвичай методи опису текстур поділяють на дві основні категорії: класичні методи, що включають інтелектуально-аналітичний підхід, та методи, що базуються на штучному інтелекті, зокрема на використанні нейронних мереж.

1.2.1 Класичні підходи до аналізу текстури

Одним з ключових завдань у текстурному аналізі є точне визначення самої концепції текстури. Існують два основні підходи до цього:

– структурний підхід: В цьому контексті текстура розглядається як сукупність базових елементів, званих текселями, які розміщені за певним регулярним чи повторюваним порядком;

– статистичний підхід: В рамках цього підходу текстура оцінюється за допомогою кількісного аналізу розподілу значень інтенсивності у певній області зображення.

Перший підхід, хоча й привабливий, в основному ефективний для регулярних штучних образів. Проте на практиці частіше вдаються до другого, більш універсального підходу, який також є простішим у плані виконання обчислень.

Важливо відзначити, що характеристики текстурних ознак безпосередньо залежать від розміру оброблюваної ділянки зображення. Більше вікно дозволяє адекватно відобразити текстурні властивості на більшій площі цільового зображення, зменшуючи вплив окремих елементів вікна на оцінку текстури. Проте, при використанні малого вікна, може виявитися недостатньо інформації для адекватного опису цільових об'єктів. Типовим для текстурного розпізнавання є той факт, що в рамках окремих зон сканування можуть зустрічатися як однорідні текстури одного класу, так і межі різних текстур та їхні комбінації.

Одним з найчастіше використовуваних способів представлення текстури є використання вектора текстурного опису. Цей вектор містить числові компоненти, які відображають загальні параметри текстури заданого зображення або окремої його частини.

Лінійне бінарне розбиття (ЛБР) представляє собою ефективну та практичну техніку для аналізу текстур. Вона полягає в порівнянні інтенсивності кожного пікселя на зображенні з інтенсивністю його восьми сусідніх пікселів. Результати порівняння з цими восьми сусідами можуть бути відображені як 8-бітне двійкове число $b1b2b3b4b5b6b7b8$. Тут кожен біт bi дорівнює 0, якщо інтенсивність i-го сусіда менша або дорівнює інтенсивності розглядуваного пікселя. У протилежному випадку bi встановлюється в 1. На основі отриманих двійкових чисел можна побудувати гістограму, яка потім використовується як опис текстури зображення [2].

Текстура, зокрема, характеризується розподілом сірих відтінків, що дозволяє оцінювати особливості зображення, які залежать від статичних обчислень другого порядку. Одним із прикладів статистичного підходу є використання матриць співінцидентності, створених на основі вихідних

зображень, з подальшим розрахунком різноманітних статистичних моментів (або характеристик).

Метод матриці співінцидентності допомагає визначити, чи належать певні ділянки зображення до одного текстурного класу, шляхом розрахунку для кожної ділянки відповідної матриці співінцидентності.

Матриця співпадінь складається з двовимірного масиву P, де індекси рядків та стовпців представляють множину V дозволених значень пікселів у зображенні. Елемент $P(a, b)$ відображає частоту, з якою значення a виникає в певному просторовому відношенні до значення b на зображенні. Це просторове відношення визначається за допомогою вектора, який вказує на зсув між пікселем із яскравістю a та пікселем із яскравістю b. Наприклад, можна вибрати просторове відношення, де b є правим сусідом a. На базі цієї матриці можна виконати розрахунок різних характеристик текстури [3].

Побудова матриць співпадінь вимагає значної обчислювальної роботи, і науковці вказують на кілька складнощів, пов'язаних з їх використанням. Однією з основних проблем є відсутність універсального методу для точного вибору вектора зсувів та визначення, яка з характеристик матриці найбільш підходить для конкретної задачі.

Метод енергетичних характеристик, запропонований Лавсом (Laws), припускає ідентифікацію різних типів текстур за допомогою локальних масок. Лавс пропонує енергетичний підхід, де оцінюється зміна текстурного вмісту в межах вікна з фіксованим розміром.

Для розрахунку енергетичних характеристик використовується набір з дев'яти масок розміром 5×5. Після цього енергетичні характеристики кожного пікселя аналізованого зображення представляються у вигляді вектора, що складається з 9 чисел.

У методі енергетичних характеристик Лавса спочатку усувається вплив інтенсивності освітлення на зображення. Це досягається переміщенням малого вікна по зображенню та відніманням локального середнього значення від кожного пікселя. Ця попередня обробка утворює зображення, на якому

середня інтенсивність у кожному окремому піксельному сегменті наближається до нуля. Розмір вікна залежить від типу зображень. Після попередньої обробки, до зображення застосовуються шістнадцять масок розміром 5×5, в результаті чого утворюються шістнадцять профільтрованих зображень.

Після отримання енергетичних карт симетричні пари комбінуються, що призводить до створення дев'яти фінальних карт [4].

У структурних підходах до текстурного аналізу передбачається, що текстури складаються з простих текстурних примітивів, розташованих відповідно до певних правил, які регулярно або майже регулярно повторюються.

З геометричної точки зору, повторюваний розподіл текстурних елементів на зображенні можна оцінити за допомогою функції автокореляції, що дозволяє визначити рівень регулярності та зернистості текстури. Для крупнозернистих текстур значення автокореляційної функції зменшується менше, ніж для дрібнозернистих текстур. У геометричних методах текстура розглядається як сукупність 'елементів' або примітивів.

Таким чином, використання класичних підходів дозволяє визначати характеристики текстури шляхом накладання певних масок, за допомогою емпіричних методів або через визначення послідовностей чи закономірностей у створених векторах.

1.2.2 Нейромережевий підхід до опису текстур

Штучний інтелект (ШІ) є напрямком сучасної інформатики, який розвивається з надзвичайною швидкістю. Незважаючи на численні спроби точно визначити ШІ, універсальне визначення до цього часу відсутнє. ШІ - це область інформатики, що зосереджена на створенні комп'ютерних систем, здатних виконувати завдання, які традиційно вважаються вираженням

інтелекту. Таким чином, ШІ охоплює широкий спектр завдань, які людина вирішує легко, але які складно для виконання обчислювальними системами. Ці завдання сприяли розвитку різноманітних напрямків у сфері ШІ, включаючи представлення завдань і пошук рішень, доведення теорем, представлення знань, експертні системи, навчання і виявлення закономірностей, спілкування природною мовою, розпізнавання образів, комп'ютерний зір, мови програмування систем ШІ та інше.

У сфері розпізнавання образів ведуться дослідження зі сприйняття зорової, слухової та інших видів інформації [5].

Існують різні типи нейронних мереж, серед яких особливу увагу заслуговують глибокі та згорткові нейронні мережі. Глибокі нейронні мережі відрізняються від звичайних нейронних мереж наявністю декількох прихованих шарів, що дозволяє здійснювати багаторівневий процес розпізнавання образів. У мережах глибокого навчання кожен шар навчається розпізнавати певний набір ознак, заснований на вихідних даних попереднього шару. Чим далі дані проходять через нейронну мережу, тим складніші ознаки можуть бути виявлені, оскільки вони є комбінацією ознак попередніх шарів.

Під час тренування глибокої мережі на навчальних даних кожен шар автоматично вивчає ознаки, намагаючись мінімізувати відмінності між очікуваним та фактичним результатами.

Згорткові мережі – це нейронні мережі, які використовують згортку замість множення принаймні в одному з їх шарів [6]. Формулу згортки приведено нижче:

$$(B * Hq)_{ij} = \sum_{m,n=-(q-1)}^{q+1} Hq(q+m, q+n)B(i+m-1, j+n-1),$$

де *(w, x)* – розраховує мий розмір карти;

mW – ширина попередньої карти;

mH – висота попередньої карти.

Ось деякі з найбільш поширених архітектур нейронних мереж, що використовуються в сучасній інформатиці:

Мультиколонна згорткова нейронна мережа (МЗНМ): Вперше представлена у 2016 році, ця мережа характеризується своєю здатністю створювати карту щільності з вхідних зображень. МЗНМ призначена для вищої точності в порівнянні з попередніми техніками і може обробляти зображення різного розміру і роздільної здатності. Її особливість полягає у використанні різних розмірів ядра згортки для адаптації до перспективних змін розмірів у зображенні. Мережа включає три згорткові колонки з різними розмірами ядра, кожна з яких моделює карту щільності на різних масштабах, причому загальна кількість параметрів мережі сягає майже 130 тисяч [7].

Згорткова нейронна мережа для розпізнавання перевантажених сцен (CSRNet): Запропонована в 2018 році, CSRNet призначена для поліпшення методів підрахунку у складних сценах. Мережа використовує глибоке навчання для розпізнавання переповнених сцен та створення деталізованих карт щільності. Вона складається з двох частин: попередньо навченої мережі VGG-16 для вилучення ознак та CNN з розширеними ядрами згортки, замінюючи традиційні операції об'єднання. Загальна кількість параметрів для навчання перевищує 16 мільйонів [8].

Нейронна мережа з високою роздільною здатністю для підрахунку людей у натовпі (MRCNet): MRCNet, поділена на кодер на основі VGG-16 та декодер з дволінійними шарами підвищеної вибірки та шарами згортки, використовує дві функції втрат для оцінки зображення і створення карт щільності натовпу. Ця мережа розглядає процес створення карт щільності як дві взаємопов'язані задачі, що дозволяє досягти високої роздільної здатності [6].

Зазвичай застосовують згорткові нейронні мережі, які розробляються і оптимізуються за допомогою навчання під наглядом. Для цього спочатку потрібно підготувати мережу, а потім її використовувати, але для цього вимагається значний обсяг даних. Наприклад, потрібен датасет, який містить

50000 зображень для тренування, 5000 для валідації та ще 5000 зображень для тестування. Через відсутність відповідного датасету, у дослідженні, на яке посилається цей текст, не використовувались нейронні мережі для аналізу структур.

Таким чином, аналіз текстурних зображень за допомогою навчання під наглядом дозволяє формувати і визначати всі ключові ознаки текстури [9].

1.3 Задача пошуку та класіфікації текстурних зображень

Для оцінювання ступеня подібності між зображенням з бази даних та запитуваним зображенням, зазвичай використовуються певні методи вимірювання відстані або специфічні характеристики, які дозволяють кількісно оцінити рівень схожості між ними. Ці характеристики схожості зображень можуть бути класифіковані у чотири головні категорії:

- схожість кольорів;
- схожість текстури;
- схожість форми;
- схожість об’єктів і відносин між об’єктами.

Колірні характеристики подібності зазвичай є досить простими у застосуванні. Вони дозволяють порівнювати кольоровий склад одного зображення з іншим або з параметрами, вказаними в запиті. Такий метод пошуку часто ґрунтується на порівнянні гістограм кольорів. Користувач системи може вказати зразок зображення та запустити пошук інших зображень, що є близькими до нього згідно з критеріями гістограми кольорів. Міри відстані, що базуються на гістограмах кольорів, повинні враховувати схожість між різними кольорами.

Характеристики текстурної подібності більш складні, ніж кольорові. Зображення, схожі за текстурою, мають аналогічний розподіл кольорів або

рівнів яскравості, але самі кольори або яскравість можуть відрізнятися між двома зображеннями.

Вектор текстурного опису може бути застосований до текстури всього зображення, але це ефективно лише для зображень з однорідною текстурою. Для більш загальних зображень вектори текстурного опису зазвичай розраховуються для кожного пікселя в малих околицях (наприклад, 15×15). Після цього пікселі можуть бути згруповані за допомогою алгоритму кластеризації, який визначає унікальні мітки для кожної нової виявленої текстурної категорії.

Колір та текстура часто виступають як глобальні атрибути зображення. Міри відстані, які базуються на кольорі та текстурі, мають на меті визначити, чи має зображення заданий колір чи текстуру, і чи розташовані ці області так само, як і на запитуваному зображенні. Форма вважається не атрибутом цілого зображення, а скоріше окремої його частини. Для використання характеристик форми потрібні додаткові операції обробки, такі як ідентифікація областей, які часто виконуються вручну або за допомогою автоматичної сегментації в певних задачах.

Існують два основних способи формулювання завдань пошуку текстури в текстурних зображеннях:

– вибір зображень з колекції, що відповідають певним критеріям: Цей підхід зосереджений на визначенні зображень, які відповідають конкретним вимогам, наприклад, на основі оцінки подібності до обраного зразкового зображення для пошуку;

– класифікація зображень: У цьому випадку визначається, до якого класу належить введене зображення, та відображаються зображення, що належать до цього ж класу. Це може бути здійснено, наприклад, за допомогою методу k найближчих сусідів або класифікації на основі нейронних мереж..

Класифікація – це процес розподілу об'єктів або понять відповідно до певної основи (критерію чи ознаки), що веде до утворення класів або груп. При класифікації особливо важливим є вибір критерію або ознаки поділу. Ця

основа може бути істотною або неістотною. Класифікація, що ґрунтується на істотних ознаках, вважається природною, тоді як класифікація, заснована на несуттєвих ознаках, є штучною або допоміжною.

Метод *k*-найближчих сусідів (*k*-NN) застосовується для розв'язання задач класифікації. Він включає віднесення об'єкта до класу, до якого належить більшість з k його найближчих сусідів у багатовимірному просторі ознак. Цей метод є одним з найпростіших алгоритмів для навчання класифікаційних моделей.

Число *k* визначає кількість сусідніх об'єктів у просторі ознак, з якими порівнюється класифікований об'єкт. Наприклад, якщо $k = 10$, кожен об'єкт порівнюється з десятьма найближчими сусідами. Рисунок 1.2 демонструє приклад використання методу k-найближчих сусідів.

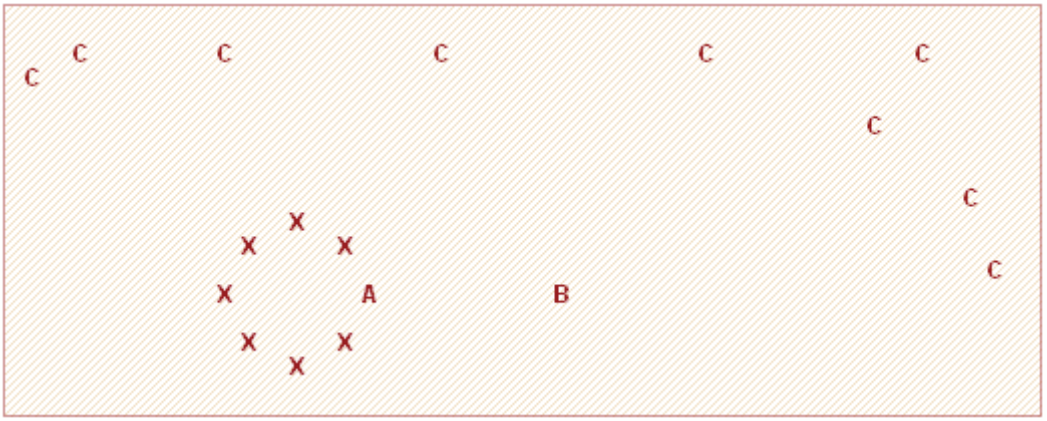


Рисунок 1.2 – Приклад застосування методу *k*-найближчих сусідів

Під час навчання алгоритму *k*-найближчих сусідів він зберігає всі вектори ознак та відповідні їм мітки класів. Коли алгоритм застосовується до нових даних, чиї класові мітки ще невідомі, він розраховує відстань між вектором нового спостереження та векторами, що були збережені під час навчання. Після цього алгоритм визначає *k* найближчих векторів до нового спостереження і класифікує його до класу, до якого належить більшість із цих *k* сусідів. Вибір оптимального значення *k* є складним. З одного боку, збільшення *k* може підвищити достовірність класифікації, але з іншого боку,

це може розмити межі між класами. На практиці часто використовуються евристичні методи для визначення *k*, наприклад, метод перехресної перевірки [10].

Незважаючи на свою алгоритмічну простоту, метод *k*-найближчих сусідів часто показує високу ефективність. Однак основним недоліком методу є його висока обчислювальна складність, яка зростає квадратично з підвищенням кількості навчальних прикладів..

1.4 Сучасні програмні засоби для вирішення задач комп’ютерного зору

На сьогодні Python є однією з найпопулярніших мов програмування для аналізу даних і машинного навчання. Це інтерпретована, об’єктно-орієнтована мова високого рівня з динамічною типізацією. Високорівневі структури даних, динамічна семантика і зв'язування роблять Python ідеальним для швидкого розроблення програм і як інструмент для інтеграції існуючих компонентів. Python підтримує модулі та пакети, що сприяє модульності коду і його повторному використанню. Інтерпретатор Python і його стандартні бібліотеки доступні у скомпільованому та вихідному коді на більшості платформ. Python також підтримує кілька парадигм програмування, включаючи об’єктно-орієнтовану, процедурну, функціональну та аспектно-орієнтовану [11].

Сьогодні існує багато інтерпретаторів для Python. Давайте розглянемо деякі з найбільш популярних на даний момент:

- JupyterLab: Це веб-інтерактивне середовище для розробки Jupyter ноутбуків, коду та даних. JupyterLab дозволяє налаштовувати і організовувати інтерфейс для підтримки різних робочих процесів у науці даних, наукових обчисленнях та машинному навчанні. JupyterLab також є розширюваним і модульним, дозволяючи додавати нові компоненти та інтегруватися з існуючими [12].

- PyCharm: Це інтегроване середовище розробки для Python, яке пропонує інструменти для аналізу коду, графічний відладчик і підтримку юніт-тестів, а також підтримує веб-розробку на Django. PyCharm створений компанією JetBrains на основі IntelliJ IDEA, що є популярним для розробки на Java.

- Google Colab: Це безкоштовне інтерактивне хмарне середовище від Google, що дозволяє працювати з кодом разом з колегами. Colab базується на Jupyter ноутбуках і зберігає дані на Google Диску. Вона підтримує текст, формули, зображення, HTML і більше, дозволяючи програмувати на Python без необхідності завантаження бібліотек чи навантаження комп'ютера. Одна з ключових особливостей Colab – це безкоштовні GPU та TPU,

1.5 Існуючі датасети

Цифрові зображення стали широко використовуватися в різних галузях людської діяльності. За останні роки актуальність наукових досліджень у цьому напрямку значно зросла завдяки поширенню комп'ютерів, засобів обробки і зберігання даних, сигналів та зображень, а також великому застосуванню цифрових зображень як інструменту для аналізу в біології та незамінного джерела інформації. Давайте розглянемо існуючі набори даних з зображеннями.

Open Images - це датасет, що складається приблизно з 9 мільйонів зображень. Він містить мітки, що визначають області об'єктів на зображеннях, сегментаційні маски об'єктів і візуальні зв'язки. Цей датасет є унікально великим і різноманітним набором даних, який призначений сприяти розвитку сучасних методів аналізу та розуміння зображень.

До кожного зображення додані позитивні мітки об'єктів, що вказують на наявність певних класів об'єктів, і негативні мітки, що вказують на відсутність певних класів. У конкурсі оцінюються тільки позитивні мітки об'єктів на

зображенні. Для кожної позитивної мітки об'єкта в зображенні існує анотація для кожного екземпляра цього класу об'єктів на зображенні.

Навчальний набір Open Images V4 містить інформацію про 14,6 мільйонів обмежують прямокутників для 600 різних класів об'єктів, яка охоплює 1,74 мільйона зображень. Це робить його найбільшим з наявних наборів даних з анотаціями розташування об'єктів. Більшість прямокутників були створені професійними експертами вручну для забезпечення точності та однаковості. Зображення у цьому наборі даних дуже різноманітні і часто включають складні сцени з кількома об'єктами, в середньому 8,4 об'єкта на зображенні. Крім того, набір даних містить мітки рівня зображення, які охоплюють тисячі класів.

Dlib - цей датасет містить навчені моделі і використовується як частина прикладних програм бібліотеки dlib. Він надається як навчальний матеріал, який пояснює, як користуватися різними функціями бібліотеки dlib. Детектор обличчя, який включений до цієї бібліотеки, базується на методі гістограми орієнтації. Він здатен виконувати швидке вирівнювання обличчя з використанням ансамблю дерев регресії, і це зазвичай забирає всього одну мілісекунду.

PyTorch надає два основних об'єкти для роботи з даними: torch.utils.data.DataLoader та torch.utils.data.Dataset. Вони дозволяють використовувати попередньо завантажені набори даних або створювати власні дані. Набір даних містить зразки і відповідні мітки, а DataLoader допомагає легко ітерувати через набір даних для забезпечення зручного доступу до зразків.

Бібліотеки у галузі PyTorch надають низку попередньо завантажених наборів даних, таких як FashionMNIST, які в подробицях реалізують інтерфейс torch.utils.data.Dataset та містять функції, специфічні для конкретних даних [15].

TensorFlow також пропонує набори даних, готові для використання з TensorFlow та іншими Python ML-фреймворками, такими як Jax. Усі ці набори

даних представлені у форматі tf.data.Datasets, що спрощує їх використання та надає високопродуктивні можливості для обробки даних [16].

Scikit-learn постачається з невеликими стандартними наборами даних, які можна використовувати без необхідності завантаження зовнішніх файлів з веб-сайтів. Ці набори даних добре підходять для швидкої демонстрації різних алгоритмів, доступних в scikit-learn. Однак їхній обсяг часто занадто малий для реальних завдань машинного навчання.

Набір даних Downsampled Imagenet використовується для експериментів у сфері оцінки щільності та генеративного моделювання, а також для завдань оцінки глибини зображень та генерації. Він містить понад 1.3 мільйона зображень об'єктів, сцен, транспортних засобів, людей та інших об'єктів [17].

Набір даних Lsun містить широкомасштабні зображення різних об'єктів з визначених категорій, таких як спальні, вежі і інші. Цей набір даних використовується для завдань розуміння сцен (scene understanding). Він включає більше 9 мільйонів зображень сцен, які розділені на категорії.

Набір даних Places 365 складається з 1.8 мільйонів зображень різних місць і сцен, включаючи офіси, котеджі, піри, поїзди з 365 категорій сцен. Він використовується для навчання глибоких нейронних мереж CNN Places365. У цьому наборі є 50 зображень на категорію для набору для перевірки та 900 зображень на категорію для набору для тестування.

VGGFace2 є одним із найбільших наборів даних з зображеннями. Зображення у цьому наборі завантажуються з Google Image Search і можуть значно відрізнятися за позами, віками, висвітленням, етнічною приналежністю і професіями. VGGFace2 містить зображення людей різних національностей, акцентів, професій та вікових категорій. Всі фотографії осіб були зроблені "в дикій природі" з різними позами, виразами обличчя та при різних умовах освітлення і закриття об'єктів. Розподіл осіб у цьому наборі даних різний для різних особистостей і коливається від 87 до 843, і в середньому міститься 362 зображення для кожної особи.

COCO, зібраний у співпраці між Google, FAIR, Caltech та іншими організаціями, призначений для розв'язання завдань розпізнавання об'єктів і сегментації об'єктів на зображеннях, а також для розпізнавання об'єктів в контексті та суперпіксельної сегментації об'єктів. Даний набір даних містить 1,5 мільйона прикладів об'єктів, охоплює 80 категорій об'єктів і 91 категорію речей, а також містить 250 000 зображень людей з відзначеними ключовими точками [18].

CC0Textures - це набір текстурних зображень, які доступні у роздільній здатності до 8K, іноді навіть вище, тобто це зображення високої якості. Усі ці текстури супроводжуються картами PBR (фізичної обробки променів) та плитками. Для багатьох нових матеріалів використовується фотограметрія, що забезпечує докладні карти переміщення аж до рівня окремих пікселів. Багато з цих матеріалів постачаються з файлами речовин, що дає змогу налаштовувати їх згідно з вашими власними потребами [19].

Отже, відомо, що існує значна кількість готових наборів даних з зображеннями, які застосовуються для розв'язання різноманітних завдань. Більшість з цих наборів не є рівномірними і не мають великої кількості текстурних зображень у вигляді цілих структур. Декілька прикладів таких ситуацій включають у себе, наприклад, зображення цегляної стіни, де нижня частина може бути вкрита травою. У контексті початкової роботи з алгоритмами це може значно вплинути на результати досліджень. Тому такі набори даних не є ідеальними для аналізу текстурних зображень.

1.6 Постановка задачі

Таким чином, дослідження методів аналізу текстурних зображень є актуальним завданням для обробки і розпізнавання зображень.

Об’єктом роботи є проблема пошуку та класифікації текстурних зображень.

Для досягнення мети дослідження методів аналізу текстурних зображень і вирішення завдань пошуку та класифікації текстур, потрібно виконати наступні завдання:

– розглянути поточний стан питання аналізу текстурних зображень.

– дослідити різні методи опису текстур (включаючи класичні підходи та нейромережеві підходи) для збагачення розуміння їх ефективності і областей застосування.

– розібрати задачі пошуку та класифікації текстурних зображень, з'ясувати їх характеристики та вимоги.

– вивчити сучасні програмні засоби, які можуть бути використані для вирішення задач комп'ютерного зору в контексті аналізу текстур.

– дослідити існуючі датасети зображень, які можуть бути використані для навчання та тестування методів аналізу текстур.

– сформулювати конкретну мету та завдання дослідження.

– провести дослідження для вибору критерію прийняття рішення щодо схожості текстурних зображень.

– розробити алгоритм для пошуку та класифікації текстурних зображень в електронній колекції, використовуючи вивчені методи та програмні засоби.

– підготувати датасет текстурних зображень, який буде використовуватися для експериментів.

– реалізувати алгоритми пошуку та класифікації на основі локальних бінарних шаблонів, матриці збігів та нейронної моделі.

– провести порівняння результатів класифікації текстурних зображень, отриманих за допомогою досліджуваних методів.

– здійснити аналіз швидкодії реалізованих методів.

– зробити висновки щодо роботи цих алгоритмів та їхньої ефективності в контексті аналізу текстурних зображень.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ОПИСУ ТЕКСТУР ТА КЛАСИФІКАЦІЇ ТЕКСТУР

2.1 Математична модель методу лінійних бінарних шаблонів

Метод локальних бінарних шаблонів - це спосіб опису для кожного пікселя, який вказує напрям, в якому зменшується яскравість. Цей метод є ефективним оператором, який представляє кожен піксель зображення у вигляді бінарного числа, залежно від інтенсивності його сусідніх пікселів.

В процесі виконання цього методу, центральний піксель візьмемо як поріг і порівняємо його значення яскравості з кожним з пікселів, які оточують його. Якщо значення більше порогу або рівне йому, то піксель отримує значення 1, інакше - 0. Отримане восьмибітне число відображає характеристику оточуючих пікселів і визначає околицю пікселя. Усього існує 256 можливих варіантів таких чисел (28).

Отже, кожному пікселю на зображенні присвоюється одна з 256 міток, які відображають його характеристики. З цих даних можна побудувати гістограму і порівнювати текстури зображень за допомогою гістограм ЛБШ [20]. На рисунку 2.1 наведено приклад роботи оператора ЛБШ над напівтоновим зображенням.

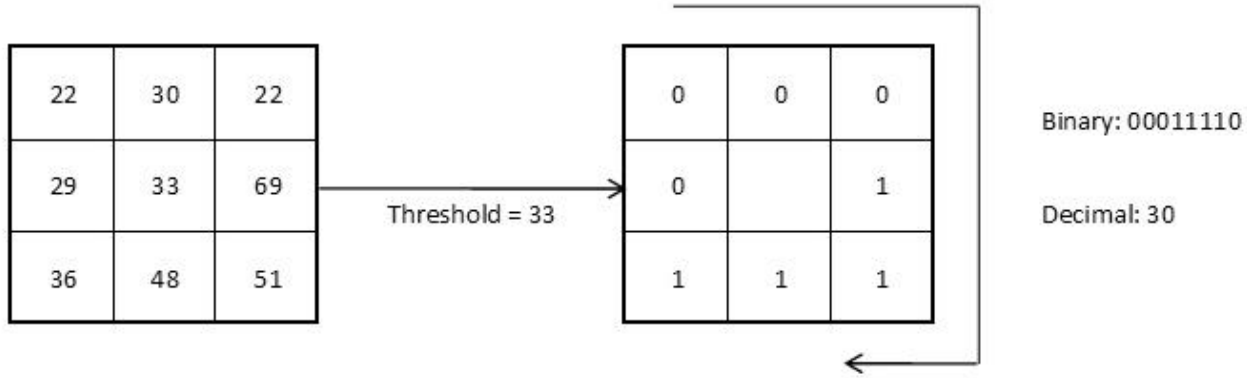


Рисунок 2.1 – Приклад роботи оператора ЛБШ

Існує варіація методу ЛБШ, в якій використовуються пікселі, що розташовані на певній відстані від центрального пікселя. У цьому випадку пікселі околиці розташовані на колі з радіусом *R*. Кількість точок на цьому колі може бути вибрана довільно і позначається як *P*. Для обчислення значень в цих точках для різних значень радіусу *R* і кількості точок *P* використовується білінійна інтерполяція [21].

На рисунку 2.2 представлені набори пікселів на колі для різних значень *P* та *R*.

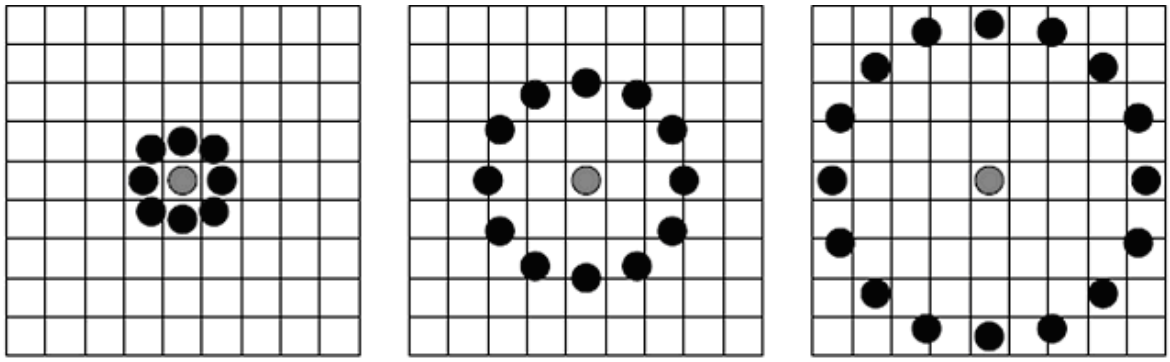
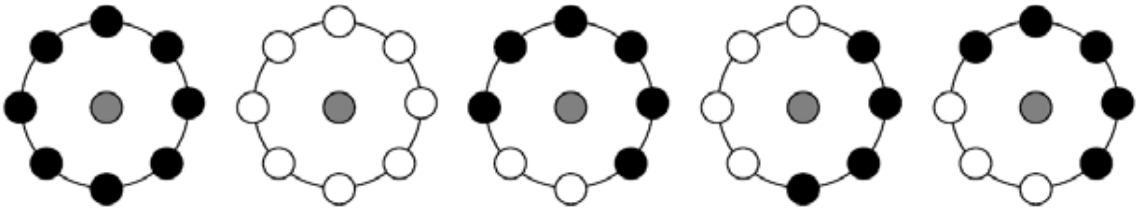


Рисунок 2.2 – Модифікований ЛБШ для різних *P* і *R*:
а) кількість точок *P* = 8, радіус околу *R* = 1.0;
б) кількість точок *P* = 12, радіус околу *R* = 2.5;
в) кількість точок *P* = 16, радіус околу *R* = 4

Деякі бінарні коди несуть в собі більше інформації, ніж інші. Наприклад, локальний бінарний шаблон називається рівномірним, якщо в ньому міститься не більше трьох послідовних серій «0» і «1» (наприклад, 00000000, 001110000 і 11100001). Ці рівномірні ЛБШ важливі, оскільки вони визначають тільки суттєві локальні особливості зображення, такі як кінці ліній, межі, кути і плями (див. рисунок 2.3). Крім того, вони дозволяють значно заощадити пам'ять, вимагаючи менше різних шаблонів (P(P–1)+2), ніж у випадку інших методів (2P).



а б в г д

Рисунок 2.3 – Приклади локальних особливостей, що детектуються ЛБШ, де а – пляма, б – пляма або фон, в – кінець лінії, г – грань, д – кут

Математичний підхід до опису функціонування методу Локальних Бінарних Шаблонів (ЛБШ) може бути виражений через сумування різниць інтенсивностей між центральним пікселем і його сусідніми пікселями на околиці. Ця процедура визначає бінарний шаблон для центрального пікселя. [22]:

$$LBP(x,y) = \sum_{i=0}^{P-1} s\,(g_i - g_c)\,2^i,$$

де *x, y* – координати центрального пікселя;

i – номер пікселя, що розглядається;

g_c – значення яскравості центрального пікселя;

g_i – значення яскравості пікселя, що розглядається.

s(z) дорівнює:

$$s(z) = \begin{cases} 1, z \geq 0; \\ 0, z < 0. \end{cases}$$

Застосовуючи оператор Локальних Бінарних Шаблонів (ЛБШ) до кожного пікселя на зображенні, ми отримуємо можливість побудувати гістограму, в якій кожному рівномірному коду ЛБШ відповідає окремий стовпець. У цій гістограмі також присутній додатковий стовпець, який містить інформацію про всі нерівномірні шаблони.

Зображення можна розглядати як набір різноманітних локальних особливостей, які добре описуються за допомогою ЛБШ. Проте гістограма, побудована для всього зображення в цілому, відображає лише наявність різних локальних особливостей, не зберігаючи інформацію про їх розташування на зображенні. Для врахування такої інформації зображення може бути поділене на підобласті, в кожній з яких обчислюється власна гістограма ЛБШ. Шляхом об'єднання (конкатенації) цих гістограм може бути отримана загальна гістограма, яка враховує як локальні, так і глобальні особливості зображення. [23].

Отже, ми отримуємо зображення, в якому кожен піксель має свою унікальну мітку від 0 до 256 відповідно до методу Локальних Бінарних Шаблонів (ЛБШ). Після цього ми будуємо гістограму g на основі цих міток. Гістограма g може бути використана для подальшої класифікації або аналізу зображення. Приклад побудови такої гістограми можна побачити на рисунку 2.4.

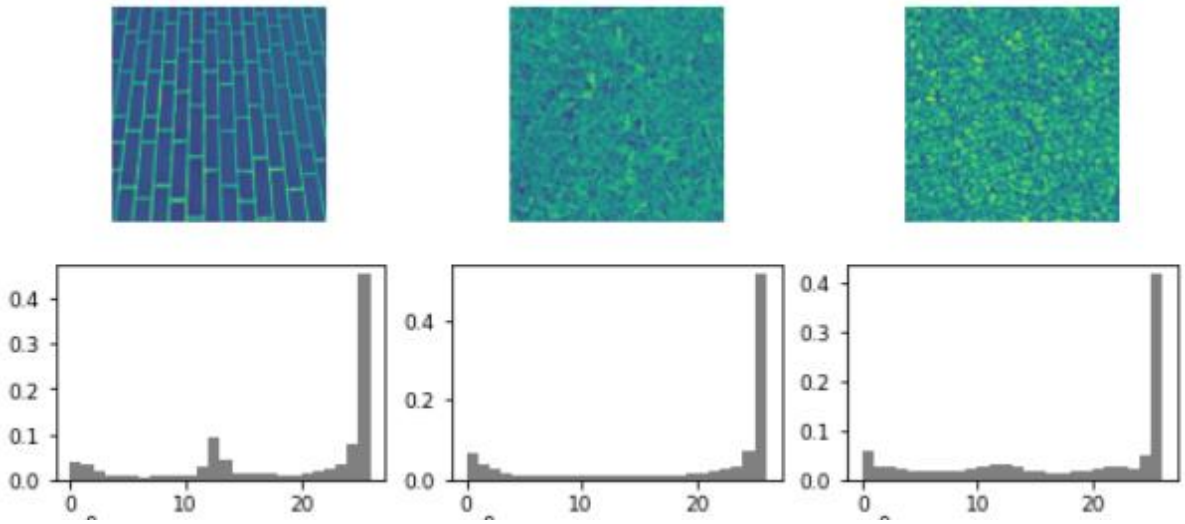


Рисунок 2.4 – Приклад відображення гістограм після обробки зображень ЛБШ

2.2 Математична модель методу, на основі матриці збігів

Матриця збігів (в англійській літературі також використовується термін "coincidence matrix") - це двовимірний масив P, в якому індекси рядків і стовпців представляють собою безліч V можливих значень пікселів, допустимих на зображенні. Наприклад, для монохромних зображень V може бути набором можливих значень інтенсивності (яскравості), а для кольорових зображень V відповідає набору можливих кольорів. Значення P (a, b) показує, скільки разів значення "a" зустрічається на зображенні у відношенні до значення "b" при певному заданому просторовому зсуві між пікселями a і b [24]. Наприклад, цей зсув може бути визначений за допомогою вектора, який описує "значення b являється правим сусідом для значення a" [25].

Розглянемо процес побудови матриці спільного виникнення рівнів сірого на прикладі, який показаний на рисунку 2.5.

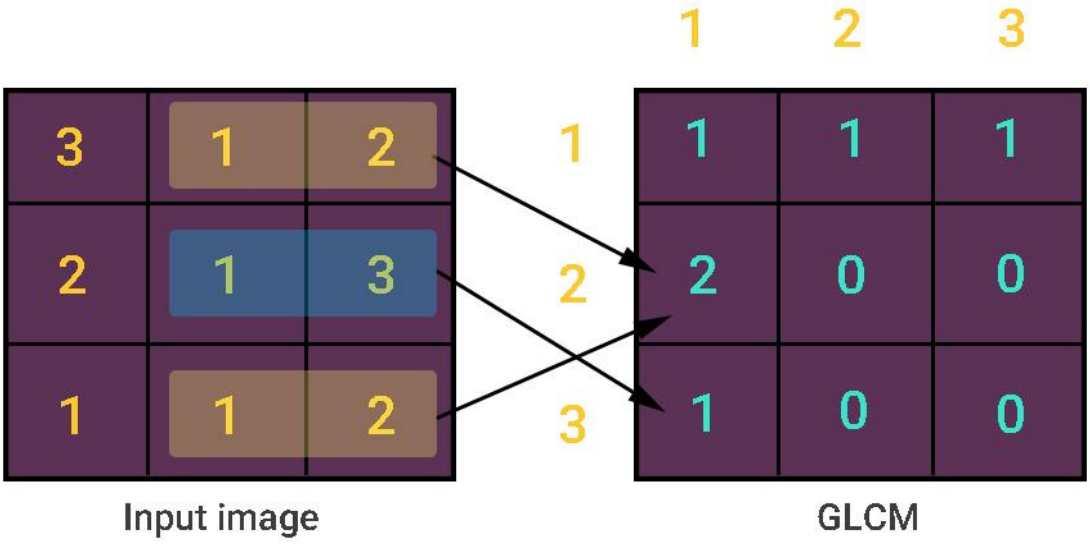


Рисунок 2.5 – Матриця спільного виникнення рівня сірого

Формально матриця збігів записується в такий спосіб [26]:

$$P_d(\mathrm{a}, \mathrm{b}) = \sum_{\mathrm{r},s} p(\mathrm{a}, \mathrm{b}, (\mathrm{r}, \mathrm{s}), (\mathrm{t}, \mathrm{v})),$$

де $p(a,\ b,\ (r,\ s),(t,\ v)) = \begin{cases} 1, \text{якщо } B(r,\ s) = a,\ B(t,\ v) = b; \\ 0, \text{в іншому випадку } (t, v) = (r + dx, s + dy). \end{cases}$

Використаємо нормовану матрицю, яка визначається виразом:

$$N_d = \frac{P_d(\text{a,b})}{\sum_a \sum_b P_d(\text{a,b})}.$$

Обчислені значення входжень дійсно належать діапазону [0, 1].

Матриці збігів є репрезентацією властивостей текстури, проте для безпосереднього аналізу зображень вони не завжди є зручними, наприклад, для порівняння двох різних текстур. Отже, дані матриці збігів використовуються для обчислення числових характеристичних ознак, які можуть служити більш компактним представленням текстури. У вашій роботі були використані наступні характеристики [2, 5, 3]:

$$x_1 = \sum_a \sum_b {N_d}^2(\text{a,b}),$$

$$x_2 = \sum_a \sum_b \frac{N_d(\text{a,b})}{1+a+b},$$

$$x_3 = \sum_a \sum_b (a-b)^2 {N_d}^2(\text{a,b}),$$

$$x_4 = \max_{\text{a,b}} N_d(\text{a,b}),$$

$$x_5 = \sum_a \sum_b \text{ab} \text{N}_d(\text{a,b}),$$

$$x_6 = \sum_a \sum_b N_d(\text{a,b}) \log_2 N_d(\text{a,b}),$$

$$x_7 = \sum_a \sum_b \frac{N_d(\text{a,b})}{1+(a-b)^2}.$$

Оскільки деякі характеристики мають значення більше 1, ми можемо знормалізувати їх за допомогою наступної функції [27]:

$$f(x) = \frac{e^{\text{ax}} - e^{-\text{ax}}}{e^{\text{ax}} + e^{-\text{ax}}},$$

де коефіцієнт *a* визначає кривизну графіка.

Після проведення нормалізації, всі характеристики набувають значень в межах від нуля до одиниці.

2.3 Критерій схожості зображень

У контексті аналізу даних, поняття критерію схожості відноситься до спеціальної функції, задача якої - вимірювання відстані між об'єктами, представленими як багатовимірні вектори в просторі характеристик.

Давайте розглянемо приклад, коли ми працюємо з двовимірним простором, де кожен об'єкт має свої власні координати:

– об’єкт *p* – *(x, y)*;

– об’єкт *q* – *(s, t)*;

– об’єкт *z* – *(v, w)*.

Таким чином, функція *d* класифікується як функція відстані або критерій схожості, за умови дотримання певних критеріїв:

– $d(p,q) \geq 0$;

– $d(p,q) = d(q,p)$;

– $d(p,q) \leq d(p,q) + d(q,z)$.

Часто в аналізі даних застосовуються такі метрики, як Евклідова відстань. Ця відстань є геометричним виміром у багатовимірному просторі і визначається на основі теореми Піфагора.

Нехай в n-вимірному просторі задані дві точки: $p(p_1, p_2, \ldots p_n)$ та $q(q_1, q_2, \ldots q_n)$.

Тоді Евклідова відстань між двома точками визначається за допомогою певної формули, яка використовує координати цих точок у просторі:

$$d(p,q) = \sum_{i=1}^{n} p_i - q_i \, .$$

Евклідова відстань вважається найбільш інтуїтивно зрозумілою і легко інтерпретованою мірою різниці або схожості об'єктів, представлених векторами ознак у багатовимірному просторі. Вона відображає основні властивості відстані між точками і тому широко використовується в аналізі даних для класифікації спостережень у класи та кластери, оцінки помилок, а також у візуалізаційних інструментах, наприклад, на картах Кохонена.

Відстань міських кварталів, також відома під кількома іншими назвами, такими як Манхеттенська відстань, метрика прямокутного міста, метрика *L1*, метрика міського кварталу, метрика таксі, метрика Манхеттена, прямокутна метрика, метрика прямого кута, була розроблена Германом Мінковським. Ця метрика розраховує відстань між двома точками як суму абсолютних значень різниць їх координат. Назва "Манхеттенська відстань" походить від сітчастого планування вулиць Манхеттена, і вона вимірює суму довжин проекцій відрізка між точками на осі координат у *n*-вимірному векторному просторі:

$$d(p,q) = \sum_{i=1}^{n} |p_i \ - \ q_i| \ .$$

Манхеттенська відстань є чутливою до обертання координатної системи, але залишається незмінною при відображенні відносно осі координат або при перенесенні. У геометрії, яка базується на Манхеттенській відстані, дотримуються всі основні аксіоми.

Також існує підхід до розгляду дивергенції Куллбека-Лейблера (*KL*) як метрики відстані, хоча вона не є симетричною. Ця метрика кількісно оцінює відмінності між двома розподілами ймовірностей. Замість безпосередньої роботи з розподілом, можна використовувати інший розподіл з відомими характеристиками, наприклад, нормальний розподіл, для спрощення задачі. Дивергенція *KL* дозволяє визначити, чи краще використовувати розподіл Пуассона або нормальний розподіл для апроксимації даних. Для розподілів *P* і *Q* неперервної випадкової величини, дивергенція Куллбека-Лейблера обчислюється за допомогою інтегралу.

$$d(p,q) \ = \ \int p(x) \log(\frac{p(x)}{q(x)}) \ dx \ .$$

З іншої сторони, коли P і Q є розподілами ймовірностей дискретної випадкової величини, дивергенція Куллбека-Лейблера визначається шляхом сумування відповідних виразів. Це означає, що для дискретних розподілів процес обчислення цієї дивергенції відрізняється від методу, що застосовується для неперервних розподілів.

$$d(p,q) \ = \ \sum p_i(x) \log(\frac{p(x)}{q(x)}) \ .$$

2.4 Оцінка класифікатора за текстурними ознаками

У сфері машинного навчання важливим аспектом є використання метрик для оцінки ефективності моделей і порівняння різних алгоритмів. Основний етап перевірки ефективності цих моделей - це тестування на вибірці, де вже визначені відповідності між документами та їх класами. Отримання такої тестової вибірки може бути складним і зазвичай вимагає значної ручної роботи, оскільки вона часто формується людьми.

Коли тестова вибірка готова, її використовують для оцінки працездатності класифікатора документів, порівнюючи його рішення з вже відомими правильними відповідями. Щоб визначити, чи алгоритм працює краще чи гірше, потрібно обрати відповідну числову метрику для оцінки його ефективності.

Далі розглянемо кілька критеріїв якості в задачах класифікації та фактори, які важливі при виборі метрики [28]. У простому випадку, така метрика може бути застосована до документів з використанням класифікатора, що вважається правильним рішенням.

$$Accuracy = \frac{P}{N},$$

У контексті оцінки машинного навчання:

– P означає кількість документів, для яких класифікатор прийняв правильне рішення.

– N - це розмір навчальної вибірки.

Ця метрика має особливість, яку необхідно враховувати: вона прирівнює всі документи за вагою. Це може бути проблематично у випадках, коли розподіл документів у навчальній вибірці є сильно зміщеним на користь одного чи декількох класів. В таких випадках класифікатор може бути більш точним у класифікації документів цих класів, в той час як інші класи залишаються недостатньо представленими.

Один із способів вирішення цієї проблеми - навчання класифікатора на збалансованому наборі документів. Однак, це може позбавити класифікатор інформації про відносну частоту класів, що також є важливою.

Альтернативний підхід - змінити методику формальної оцінки якості, використовуючи такі метрики, як точність (precision) та повнота (recall). Ці метрики часто використовуються для оцінки алгоритмів вилучення інформації. Точність - це частка документів, які насправді належать даному класу, щодо всіх документів, які система віднесла до цього класу. Повнота - це частка документів, знайдених класифікатором, які належать до класу, щодо всіх документів цього класу у тестовій вибірці. Ці значення можна легко розрахувати на основі контингентної таблиці, яка складається для кожного класу окремо.окремо (табл. 2.1).

Таблиця 2.1 – Для визначення точності та повноти алгоритму в машинному навчанні використовуються такі позначення

Категорія і		Експертна оцінка	
		Позитивна	Негативна
Оцінка системи	Позитивна	TP	FP
	Негативна	FN	TN

У таблиці, яка використовується для оцінки алгоритму класифікації, відображається інформація про кількість правильних та неправильних рішень, які система ухвалила для документів певного класу. Вона включає:

- TP (True Positive): Випадки, коли система правильно ідентифікувала документи як належні до заданого класу.

- TN (True Negative): Випадки, коли система правильно визнала, що документи не належать до заданого класу.

- FP (False Positive): Випадки, коли система невірно визначила документи як належні до класу.

- FN (False Negative): Випадки, коли система невірно визначила документи як не належні до класу.

На основі цих показників можна обчислити точність та повноту:

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}.$$

У практичному застосуванні, обчислення значень точності та повноти є значно зручнішим за допомогою матриці неточностей (confusion matrix), особливо коли кількість класів не перевищує 100-150.

Матриця неточностей представляє собою матрицю розміром N times N, де N - кількість класів. Стовпці матриці представляють експертні рішення, а рядки - рішення класифікатора. При класифікації документа з тестової вибірки відповідна комірка матриці (на перетині рядка і стовпця, що відповідають класам, визначеним класифікатором і реальним класом документа) інкрементується.

Точність та повнота для кожного класу в матриці неточностей розраховуються так:

– точність для класу визначається як відношення діагонального елемента матриці, що відповідає цьому класу, до суми елементів у відповідному рядку класу

$$Precision = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{c,i}},$$

– повнота для класу - це відношення того ж діагонального елемента до суми елементів у стовпці, що відповідає цьому класу

$$Recall = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{i,c}}.$$

У реальному світі часто буває складно досягти високих значень точності та повноти одночасно. Зазвичай, поліпшення одного з цих показників веде до зниження іншого. Тому існує потреба у метриці, яка забезпечувала б компромісний баланс між точністю та повнотою. Однією з таких метрик є *F*-міра.

F-міра - це метрика, яка поєднує в собі точність та повноту, використовуючи гармонійне середнє. Вона допомагає оцінити загальну якість класифікації, особливо коли потрібно знайти баланс між точністю та повнотою. *F*-міра знижується до нуля, якщо або точність, або повнота наближаються до нуля, тим самим підкреслюючи важливість обох показників. Її можна виразити як:

$$F = 2\frac{Precision * Recall}{Precision + Recall}.$$

Формула стандартної *F*-міри, яка призначена для рівного врахування точності та повноти, може бути адаптована таким чином, щоб надати різну вагу цим двом метрикам. Це може бути корисним, коли ви віддаєте перевагу одній метриці над іншою у вашому алгоритмі.

$$F = (\beta^2 + 1)\frac{Precision * Recall}{\beta^2 Precision + Recall},$$

– якщо β<1, формула надає більшу вагу точності;

– якщо β>1, більша вага надається повноті;

– при β=1, формула стає стандартною *F*-мірою (або *F*1-мірою), яка враховує точність і повноту порівну рисунок 2.6 – 2.8.

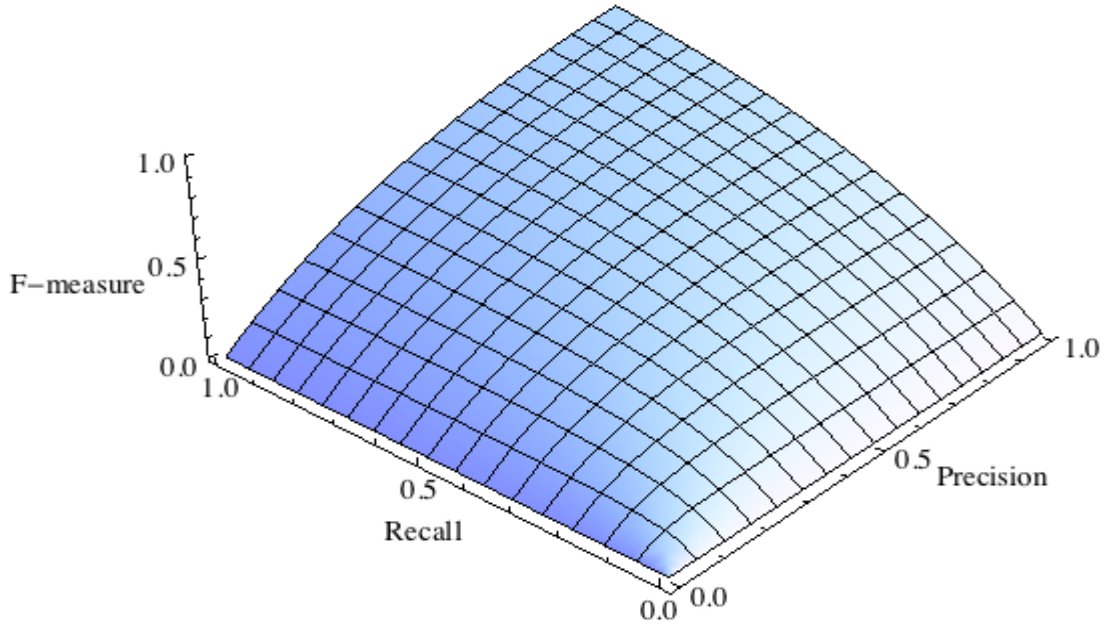


Рисунок 2.6 – Збалансована *F*-міра

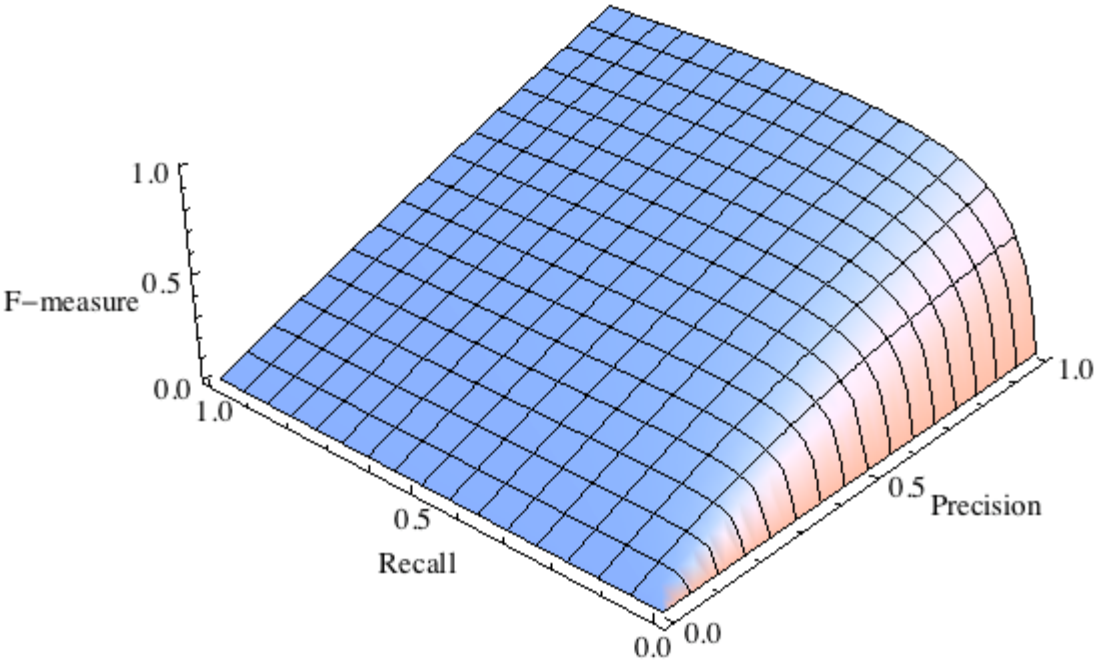


Рисунок 2.7 – *F*-міра з пріоритетом точності *(β^2 = 0,25)*

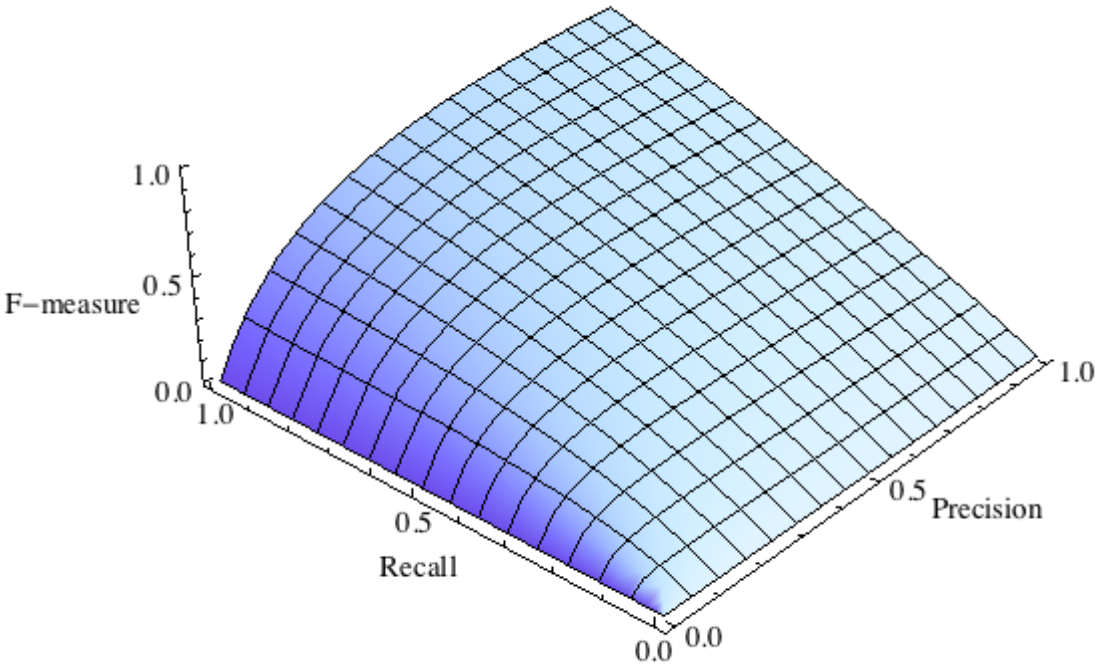


Рисунок 2.8 – *F*-міра з пріоритетом повноти *($\beta^2 = 2$)*

Ця міра дозволяє більш гнучко налаштувати оцінку якості класифікатора, акцентуючи увагу на тих аспектах, які вважаються найбільш критичними для конкретної задачі. Використання *F*-міри як формальної метрики оцінки якості класифікатора дозволяє об'єктивно виміряти його ефективність, спрощуючи процес прийняття рішень щодо подальших модифікацій алгоритму.

2.5 Нейронная мережа MobileNetV2 та принципи донавчання нейронних мереж

Головною трудністю, яка виникає при навчанні нейронних мереж, є потреба у великих обчислювальних ресурсах та обширних наборах даних, які служать основою для навчання цих мереж. Для подолання цих труднощів було розроблено підхід, що полягає у використанні передньо навчених нейронних мереж. Ці мережі вже мають навчені параметри на великих обсягах

різноманітних даних, в залежності від конкретної мережі. Користувачеві залишається лише змінити останній шар моделі - вихідний, щоб отримати результати класифікації для обраної ним кількості класів [21]. На рисунку 2.2 цей шар позначений як "Output classes".

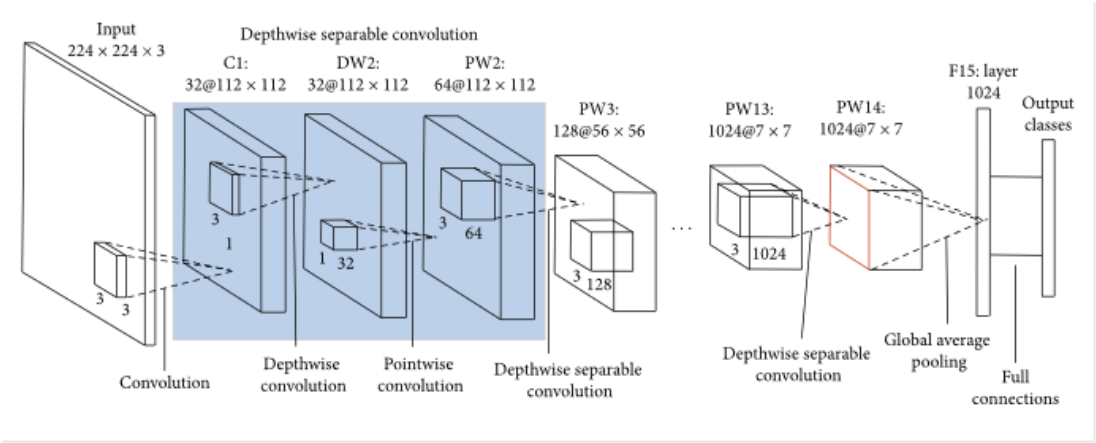


Рисунок 2.9 – Демонструє графічне відображення навченої нейронної мережі Mobilenet V2

Крім того, ці моделі можуть бути піддані додатковому навчанню. Якщо не відключати перерозподіл ваг моделі, вона буде адаптуватися до нового набору даних, який буде переданий їй користувачем. Проте, якщо «заморозити» модель, тоді навчання відбуватиметься лише у вихідному шарі, який спочатку буде ініціалізований випадковими значеннями. Зрозуміло, що оптимальніше навчати лише останній шар моделі, оскільки в цьому випадку значення останнього шару не вплинуть безпосередньо на всі ваги моделі, які не були «заморожені».

2.6 Алгоритм пошуку та класифікації текстурних зображень

Для розробки алгоритму пошуку текстурних зображень спочатку потрібно мати доступ до електронної колекції таких зображень. Алгоритм працює таким чином:

1. Підготовка Зображення-Зразка: Спочатку подається текстурне зображення як зразок. Для цього зображення збираються і зберігаються його текстурні характеристики.

2. Вибір Методу Обробки: Визначається, чи буде використовуватися класифікація зображень (тобто віднесення зображення-зразка до певного класу) або метод пошуку за критерієм схожості.

3. Встановлення Критеріїв:

- Якщо обрана класифікація, визначається поріг для віднесення зображення до класу.

- Якщо обрано пошук за критерієм схожості, встановлюється відповідний критерій.

4. Порівняння Зображень: Виконується порівняння текстурних характеристик зразка з текстурними характеристиками інших зображень у колекції.

5. Обробка Результатів:

- При використанні пошуку схожих зображень: якщо зображення відповідає критерію схожості, воно заноситься до результуючого масиву.

- При використанні класифікації: якщо результат порівняння менший за обраний поріг, зображення додається до результуючого масиву.

6. Впорядкування Результатів: Сформований масив з зображеннями, що відповідають критеріям пошуку, впорядковується за зростанням міри подібності.

Цей алгоритм дозволяє ефективно сортувати та вибирати текстурні зображення з великої колекції, використовуючи задані параметри схожості або класифікації.

Таким чином, можемо сформувати такий алгоритм:

Крок 1. У випадку, коли доступна електронна бібліотека текстурних знімків, ми вносимо зразок текстурного зображення для аналізу.

Крок 2. Витягуємо і зберігаємо ключові текстурні особливості вказаного знімку.

Крок 3. Визначаємо, який алгоритм буде використовуватися для обробки даних.

Крок 4. Встановлюємо критерії входу до класу, у разі вибору класифікації, або визначаємо параметри пошуку.

Крок 5. Процедура порівняння текстурних характеристик поданого зразка з іншими зображеннями у електронній бібліотеці.

Крок 6. Розташовуємо зображення у послідовності відповідно до зростаючої схожості, створюючи впорядкований масив з знімками, що відповідають критеріям пошуку.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ КЛАСИФІКАЦІЇ ТЕКСТУРНИХ ЗОБРАЖЕНЬ, ЇХ ПОРІВНЯЛЬНИЙ АНАЛІЗ

3.1 Створення датасету текстур

Раніше було зазначено, що хоча наявних датасетів існує велика кількість, вони не відповідають потребам, оскільки специфічних наборів даних з текстурними зображеннями не так багато.

Ці зображення часто містять нерівномірно розподілені текстури або комбінації різних текстур. Тому виникає необхідність створення власного датасету спеціально для цього дослідження.

Для зручності зберігання та доступу до цих зображень обрано використання хмарного сховища Google Drive. Ілюстрація хмарного сховища Google Drive представлена на рисунку 3.1.

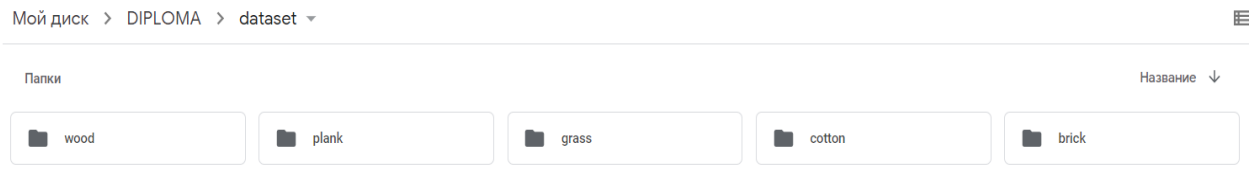


Рисунок 3.1 – Сховище для електронної колекції текстурних зображень

Отже, ми формуємо датасет, що включає 5 типів текстур, серед яких представлені як природні, так і штучні текстури, а саме, до датасету вийшли текстури:

– деревина;

– цегляна стена;

– трава;

– тканина;

– дерев’янний паркет.

Нижче подані приклади зображень з цього датасету (рис 3.2 – 3.6).

Рисунок 3.2 – Приклади датасету текстур: текстура деревини

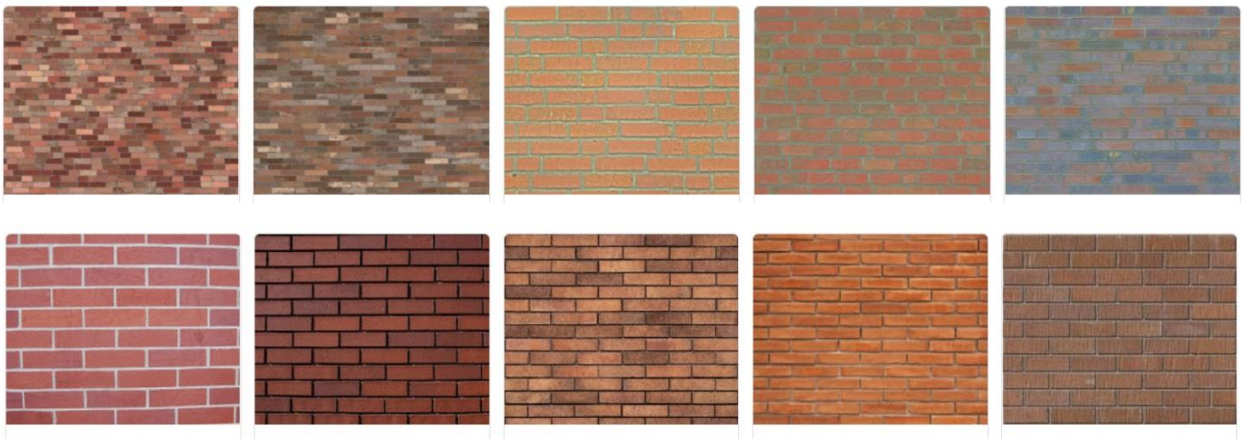


Рисунок 3.3 – Приклади датасету текстур: цегляна стіна

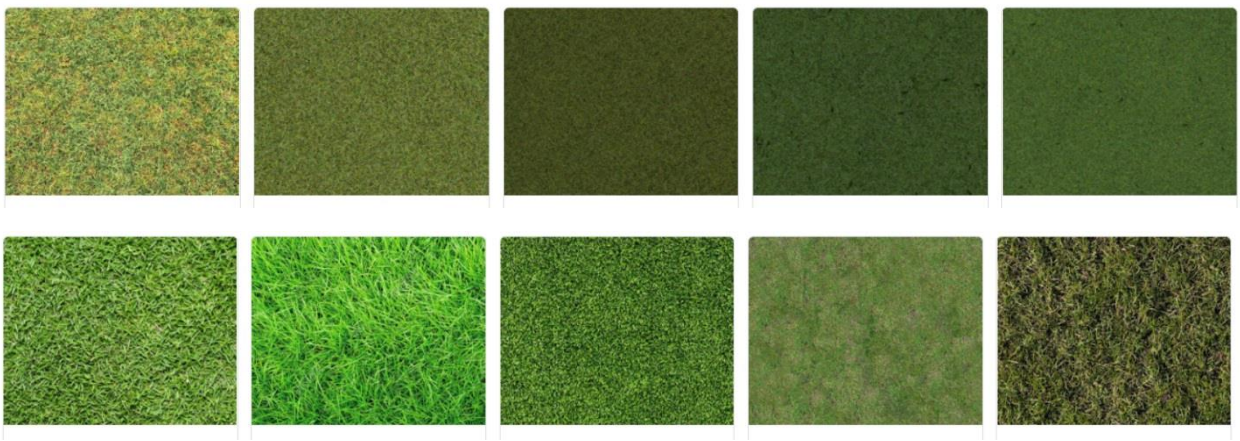


Рисунок 3.4 – Приклади датасету текстур: трава

Рисунок 3.5 – Приклади датасету текстур: бавовняна тканина

Рисунок 3.6 – Приклади датасету текстур: дерев’янний паркет

3.2 Налаштування програмного середовища

У цьому дослідженні ми вирішили застосувати мову програмування Python та робоче середовище Colab для роботи з кодом. Матриця, яку ми використовуємо, розраховує частоту виявлення точок з певними значеннями яскравості в заданих просторових відносинах. Метод LBP використовує спеціальний оператор, який перетворює кожен піксель зображення у бінарне число, засноване на інтенсивності сусідніх пікселів. На основі отриманих таким чином чисел формується гістограма, яка служить як маркер текстури.

Під час аналізу відкритих бібліотек комп'ютерного зору для Python було виявлено, що для екстракції текстурних характеристик можна

використовувати такі бібліотеки та модулі, як Mahotas, Scikits-image і OpenCV [28]. Найбільш гнучкі налаштування для GLCM надає skimage.feature.graycoprops() із Scikit-image. Метод LBP доступний у пакетах scikit-image та Mahotas (mahotas.features.lbp та skimage.feature.texture.local_binary_pattern, відповідно). Загалом, рекомендується скористатися реалізацією LBP від Scikit-image, оскільки вона пропонує більше можливостей для керування типами гістограм LBP.

Використання вже готових реалізацій методів екстракції текстурних характеристик з бібліотек Mahotas, Scikits-image, OpenCV дозволяє заощадити час та зосередитись на аналізі текстурних зображень. Таким чином, для методу лінійних бінарних шаблонів була використана бібліотека skimage.feature та її метод local_binary_pattern [29]. А для методу на основі матриці збігів були застосовані skimage.feature та її методи greycomatrix та greycoprops [30].

Для полегшення роботи з кодом у цьому проекті були використані наступні бібліотеки:

– Google.colab.patches - для візуалізації зображень;

– Google.colab - для доступу до хмарного сховища;

– Matplotlib.pyplot - для створення графіків та гістограм;

– Skimage.io - для коректного читання зображень з хмарного сховища Google Drive;

– Collections - для створення словника, який використовується для зберігання результатів аналізу;

– Numpy - для конвертації вектора, отриманого після аналізу зображення, у масив для подальшого порівняння з іншими;

– Math - для застосування різних математичних функцій;

– Cv2 - для обробки зображень, зокрема зміни їх розмірів та перетворення у чорно-біле зображення;

– Sys - для взаємодії з інтерпретатором Python, зокрема доступу до системних змінних і функцій;

– Os - для читання файлів з хмарного сховища;

– Time - для вимірювання ефективності коду.

Завантаження цих необхідних бібліотек для реалізації програмної моделі можна побачити на рисунку 3.7.

```
from skimage.feature import local_binary_pattern
from skimage.feature import greycomatrix, greycoprops
from google.colab.patches import cv2_imshow
from google.colab import drive
import matplotlib.pyplot as plt
from skimage.io import imread
from collections import defaultdict
import numpy as np
import math
import cv2
import sys
import os
import time
```

Рисунок 3.7 – Лістинг завантаження необхідних бібліотек для виконання програмної моделі

3.3 Підготовка зображень для використання медів бінарних шаблонів

На початку нашої роботи ми маємо справу з кольоровими зображеннями, які відрізняються за розмірами.

Щоб стандартизувати їх для подальшого аналізу, ми приводимо всі зображення до єдиного розміру, конкретно до 512×512 пікселів, та перетворюємо їх у чорно-білі.

Вихідний набір зображень, які ми маємо на руках на цьому етапі, відображено на рисунку 3.8.

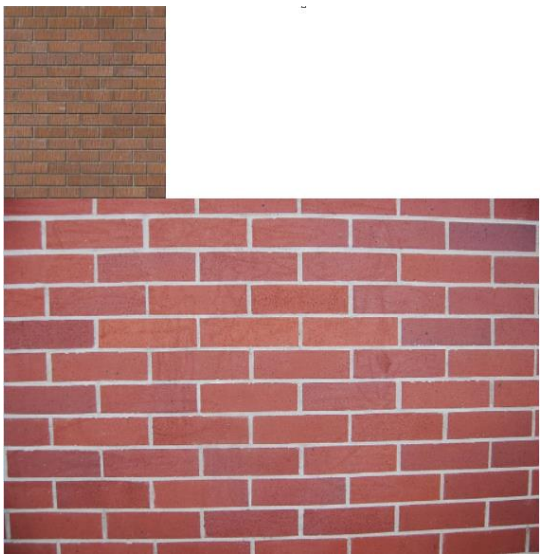


Рисунок 3.8 – Приклад вхідних зображень, що мають різний розмір

Після того, як зображення були приведені до єдиного розміру, ми отримуємо результати, що відображені на відповідному рисунку 3.9.

Рисунок 3.9 – Приклад зображень після приведення зображень до єдиного розміру – 512×512 пікселів

В кінці переводимо кольореве зображення у сірий колір, як на рисунку 3.10.

Рисунок 3.10 – Приклад зображень після приведення зображень до сірого кольору

3.4 Реалізація методу бінарних шаблонів

У даній роботі метод лінійних бінарних шаблонів (ЛБШ) був реалізований за наступним алгоритмом:

Крок 1. Завантаження зображення. Початок роботи з вибраним зображенням.

Крок 2. Нормалізація зображення. Приведення зображення до стандартного розміру та конвертація його у чорно-білий формат.

Крок 4. Застосування методу ЛБШ. Обробка отриманого зображення за допомогою методу лінійних бінарних шаблонів.

Крок 4. Перетворення у гістограму. Конвертація обробленого за методом ЛБШ зображення у гістограму.

Крок 5 Класифікація зображень:

– вибір зразкового зображення;

– порівняння гістограм зразкового зображення з гістограмами з електронної колекції з використанням критерію схожості;

– запис результатів порівняння у результативний масив;

– сортування результативного масиву;

– класифікація: визначення, чи входять зображення з електронної колекції до класу зразкового зображення.

На рисунку 3.11 демонструється процес завантаження зображення, його нормалізації, обробки за методом ЛБШ, перетворення у гістограму та додавання цієї гістограми до електронної колекції.

```
#brick
imgs_brick = os.listdir(dir_images_brick)
for count, imgnm in enumerate(imgs_brick,1):
  img = cv2.imread(os.path.join(dir_images_brick, imgnm))
  img_rsz = cv2.resize(img, (512, 512), interpolation = cv2.INTER_AREA)
  img_rsz_gr = cv2.cvtColor(img_rsz, cv2.COLOR_BGR2GRAY)
  img_lbp = local_binary_pattern(img_rsz_gr, n_points, radius, METHOD)
  hist, _ = np.histogram(img_lbp, density= True, bins=n_bins, range=(0, img.max()))
  textures["brick{}".format(count)] = hist
```

Рисунок 3.11 – Лістинг коду

Приклад зображень після накладання маски ЛБШ на зображення класів відображено на рисунках 3.12 – 3.16.

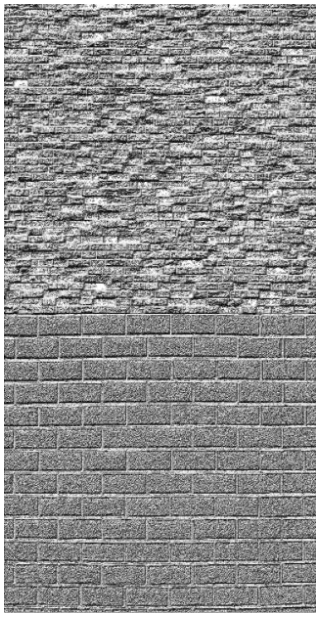


Рисунок 3.12 – Приклад зображень з класу цегляної стіни
після накладання маски методу ЛБШ

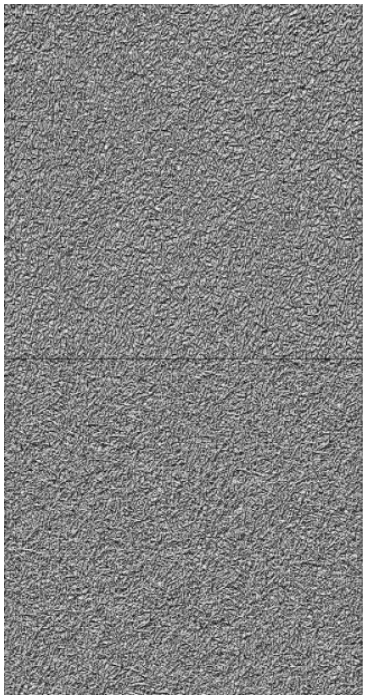


Рисунок 3.13 – Приклад зображень з класу трави
після накладання маски методу ЛБШ

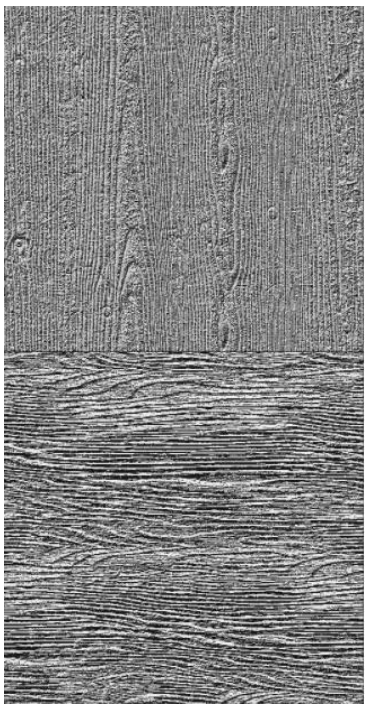


Рисунок 3.14 – Приклад зображень з класу текстури дерева
після накладання макски методу ЛБШ

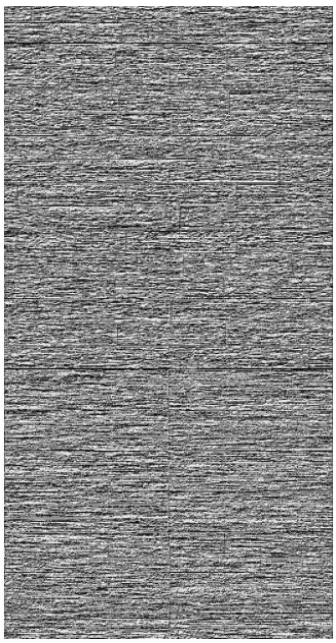


Рисунок 3.15 – Приклад зображень з класу текстури дерев’яного паркету після накладання макски методу ЛБШ

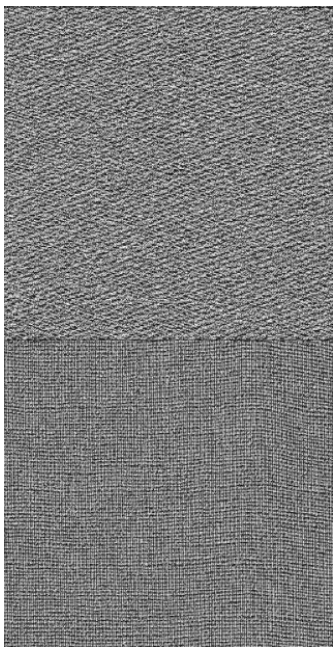


Рисунок 3.16 – Приклад зображень з класу текстури бавовняної тканини після накладання макски методу ЛБШ

На рисунку 3.17 показано обирання зображення зразку та порівняння з зображеннями з електронної колекції, проведення класифікації та найбільш схожого зображення з колекції.

```
def match(refs, img, img_name):
    global array_euclidean, array_manhattan
    best_score = 1
    best_name = 100
    classification = []
    for name, ref in refs.items():
        score = manhattan(img, ref)
        if score<threshold:
          classification.append(name)
        if score < best_score:
            best_score = score
            best_name = name
```

Рисунок 3.17 – Лістинг коду

Код програми для проведення класифікації зображень представлено на рисунку 3.18. Результати цього процесу класифікації можна побачити на рисунку 3.19. А рисунок 3.20 ілюструє результат виявлення найбільш схожого зображення на вибраний зразок у рамках цього дослідження.

```
#визначаємо ім'я класу зображення зразку
classname = ''.join([i for i in img_name if not i.isdigit()])
for name, key in refs.items():
  #визначаємо ім'я класу зображенняз електронної колекції, з яким порівнювали
  current_class = ''.join([i for i in name if not i.isdigit()])
  #перевірка чи війшло зображення до класу при перевірці score < threshold
  choiced_class = name in classification
  #перевіряємо чи війшло зображення до класу зображення зразку
  real_class = current_class == classname
```

Рисунок 3.18 – Лістинг коду

```
Зображення зразок: cotton1
Назва класу зображення зразку: cotton
Назва зображення, що увійшло до класу: grass1
Назва зображення, що увійшло до класу: grass2
Назва зображення, що увійшло до класу: grass3
Назва зображення, що увійшло до класу: grass4
Назва зображення, що увійшло до класу: grass5
Назва зображення, що увійшло до класу: grass10
Назва зображення, що увійшло до класу: brick2
Назва зображення, що увійшло до класу: brick4
Назва зображення, що увійшло до класу: brick10
Назва зображення, що увійшло до класу: cotton1
Назва зображення, що увійшло до класу: cotton2
Назва зображення, що увійшло до класу: cotton3
Назва зображення, що увійшло до класу: cotton4
Назва зображення, що увійшло до класу: cotton5
Назва зображення, що увійшло до класу: cotton6
Назва зображення, що увійшло до класу: cotton7
Назва зображення, що увійшло до класу: cotton8
Назва зображення, що увійшло до класу: cotton9
Назва зображення, що увійшло до класу: cotton10
```

Рисунок 3.19 – Результат виконання коду

```
Результат знаходження найбільш схожого зображення на  grass1 :  grass5
Результат знаходження найбільш схожого зображення на  grass2 :  cotton4
Результат знаходження найбільш схожого зображення на  grass3 :  brick10
Результат знаходження найбільш схожого зображення на  grass4 :  grass2
Результат знаходження найбільш схожого зображення на  grass5 :  grass1
Результат знаходження найбільш схожого зображення на  grass6 :  grass9
Результат знаходження найбільш схожого зображення на  grass7 :  grass8
Результат знаходження найбільш схожого зображення на  grass8 :  plank10
Результат знаходження найбільш схожого зображення на  grass9 :  grass6
Результат знаходження найбільш схожого зображення на  grass10 :  grass3
Результат знаходження найбільш схожого зображення на  brick1 :  brick7
Результат знаходження найбільш схожого зображення на  brick2 :  brick10
Результат знаходження найбільш схожого зображення на  brick3 :  brick8
Результат знаходження найбільш схожого зображення на  brick4 :  cotton4
Результат знаходження найбільш схожого зображення на  brick5 :  brick6
Результат знаходження найбільш схожого зображення на  brick6 :  brick5
Результат знаходження найбільш схожого зображення на  brick7 :  cotton5
Результат знаходження найбільш схожого зображення на  brick8 :  brick3
Результат знаходження найбільш схожого зображення на  brick9 :  brick4
Результат знаходження найбільш схожого зображення на  brick10 :  brick2
Результат знаходження найбільш схожого зображення на  wood1 :  brick5
Результат знаходження найбільш схожого зображення на  wood2 :  wood6
Результат знаходження найбільш схожого зображення на  wood3 :  plank4
Результат знаходження найбільш схожого зображення на  wood4 :  grass5
Результат знаходження найбільш схожого зображення на  wood5 :  wood10
Результат знаходження найбільш схожого зображення на  wood6 :  wood7
Результат знаходження найбільш схожого зображення на  wood7 :  wood6
Результат знаходження найбільш схожого зображення на  wood8 :  brick6
Результат знаходження найбільш схожого зображення на  wood9 :  plank1
Результат знаходження найбільш схожого зображення на  wood10 :  plank9
Результат знаходження найбільш схожого зображення на  plank1 :  wood2
Результат знаходження найбільш схожого зображення на  plank2 :  plank5
Результат знаходження найбільш схожого зображення на  plank3 :  plank7
Результат знаходження найбільш схожого зображення на  plank4 :  wood3
Результат знаходження найбільш схожого зображення на  plank5 :  plank2
Результат знаходження найбільш схожого зображення на  plank6 :  plank7
Результат знаходження найбільш схожого зображення на  plank7 :  plank3
Результат знаходження найбільш схожого зображення на  plank8 :  wood10
Результат знаходження найбільш схожого зображення на  plank9 :  wood10
Результат знаходження найбільш схожого зображення на  plank10 :  wood3
Результат знаходження найбільш схожого зображення на  cotton1 :  cotton4
Результат знаходження найбільш схожого зображення на  cotton2 :  cotton3
Результат знаходження найбільш схожого зображення на  cotton3 :  cotton2
Результат знаходження найбільш схожого зображення на  cotton4 :  brick4
Результат знаходження найбільш схожого зображення на  cotton5 :  cotton4
Результат знаходження найбільш схожого зображення на  cotton6 :  grass1
Результат знаходження найбільш схожого зображення на  cotton7 :  grass10
Результат знаходження найбільш схожого зображення на  cotton8 :  cotton5
Результат знаходження найбільш схожого зображення на  cotton9 :  grass4
Результат знаходження найбільш схожого зображення на  cotton10 :  grass4
```

Рисунок 3.20 – Результат виконання коду

3.5 Реалізація методу, на основі матриці збігів

У цій роботі метод на основі матриці збігів був реалізований за такими кроками:

Крок 1. Завантаження зображення. Починаємо обробку з вибору конкретного зображення.

Крок 2. Нормалізація зображення. Приводимо зображення до єдиного стандартного розміру та перетворюємо його у чорно-білий формат.

Крок 3. Застосування методу на основі матриці збігів. Обробка отриманого зображення за допомогою обраного методу.

Крок 4. Видобування текстурних характеристик. Отримання текстурних характеристик зображення на основі гістограми, їх нормування та сумування.

Крок 5. Класифікація зображень:

– вибір зразкового зображення;

– порівняння результатів сум зразкового зображення та сум з електронної колекції з використанням критерію схожості;

– запис результатів порівняння у результативний масив;

– сортування результативного масиву;

– класифікація: визначення, чи входять зображення з електронної колекції до класу зразкового зображення.

На рисунку 3.21 ілюструється процес завантаження зображення, його нормалізації, застосування методу на основі матриці збігів, отримання та сумування текстурних характеристик, а також додавання отриманих результатів до електронної колекції.

Що стосується результатів класифікації зображень, то вони відображені на рисунку 3.22.

```
#cotton
dir_images_cotton = '/content/drive/My Drive/DIPLOMA/dataset/cotton/'
imgs_cotton = os.listdir(dir_images_cotton)
for count, imgnm in enumerate(imgs_cotton,1):
  img = cv2.imread(os.path.join(dir_images_cotton, imgnm))
  img_rsz = cv2.resize(img, (512, 512), interpolation = cv2.INTER_AREA)
  img_rsz_gr = cv2.cvtColor(img_rsz, cv2.COLOR_BGR2GRAY)
  img_glcm = greycomatrix(img_rsz_gr, distances=[5], angles=[0], levels=256, symmetric=True, normed=True)
  contrast = greycoprops(img_glcm, 'contrast')[0][0]
  dissimilarity = greycoprops(img_glcm, 'dissimilarity')[0][0]
  homogeneity = greycoprops(img_glcm, 'homogeneity')[0][0]
  ASM = greycoprops(img_glcm, 'ASM')[0][0]
  energy = greycoprops(img_glcm, 'energy')[0][0]
  correlation = greycoprops(img_glcm, 'correlation')[0][0]
  img_sum = contrast + dissimilarity + homogeneity + ASM + energy + correlation
  textures["cotton{}".format(count)] = img_sum
```

Рисунок 3.21 – Лістинг коду

```
Назва класу зображення зразку: cotton
Назва зображення, що увійшло до класу: cotton1
Назва зображення, що увійшло до класу: cotton2
Назва зображення, що увійшло до класу: cotton3
Назва зображення, що увійшло до класу: cotton5
Назва зображення, що увійшло до класу: cotton7
Назва зображення, що увійшло до класу: cotton8
Назва зображення, що увійшло до класу: cotton9
Назва зображення, що увійшло до класу: cotton10
```

Рисунок 3.22 – Результат виконання коду

Результат знаходження найбільш схожого зображення, окрім самого порівнюваного зображення, на зображення зразок на рисунку 3.23.

```
Результат знаходження найбільш схожого зображення на  grass1 :  grass5
Результат знаходження найбільш схожого зображення на  grass2 :  grass4
Результат знаходження найбільш схожого зображення на  grass3 :  grass9
Результат знаходження найбільш схожого зображення на  grass4 :  grass5
Результат знаходження найбільш схожого зображення на  grass5 :  grass1
Результат знаходження найбільш схожого зображення на  grass6 :  grass7
Результат знаходження найбільш схожого зображення на  grass7 :  grass8
Результат знаходження найбільш схожого зображення на  grass8 :  grass7
Результат знаходження найбільш схожого зображення на  grass9 :  grass3
Результат знаходження найбільш схожого зображення на  grass10 :  grass5
Результат знаходження найбільш схожого зображення на  brick1 :  brick5
Результат знаходження найбільш схожого зображення на  brick2 :  grass9
Результат знаходження найбільш схожого зображення на  brick3 :  wood8
Результат знаходження найбільш схожого зображення на  brick4 :  brick5
Результат знаходження найбільш схожого зображення на  brick5 :  brick1
Результат знаходження найбільш схожого зображення на  brick6 :  wood3
Результат знаходження найбільш схожого зображення на  brick7 :  wood1
Результат знаходження найбільш схожого зображення на  brick8 :  wood10
Результат знаходження найбільш схожого зображення на  brick9 :  wood3
Результат знаходження найбільш схожого зображення на  brick10 :  plank3
Результат знаходження найбільш схожого зображення на  wood1 :  brick7
Результат знаходження найбільш схожого зображення на  wood2 :  wood9
Результат знаходження найбільш схожого зображення на  wood3 :  brick8
Результат знаходження найбільш схожого зображення на  wood4 :  wood7
Результат знаходження найбільш схожого зображення на  wood5 :  plank4
Результат знаходження найбільш схожого зображення на  wood6 :  wood3
Результат знаходження найбільш схожого зображення на  wood7 :  wood4
Результат знаходження найбільш схожого зображення на  wood8 :  brick3
Результат знаходження найбільш схожого зображення на  wood9 :  wood2
Результат знаходження найбільш схожого зображення на  wood10 :  brick8
Результат знаходження найбільш схожого зображення на  plank1 :  brick9
Результат знаходження найбільш схожого зображення на  plank2 :  plank10
Результат знаходження найбільш схожого зображення на  plank3 :  plank7
Результат знаходження найбільш схожого зображення на  plank4 :  plank6
Результат знаходження найбільш схожого зображення на  plank5 :  plank10
Результат знаходження найбільш схожого зображення на  plank6 :  plank4
Результат знаходження найбільш схожого зображення на  plank7 :  plank3
Результат знаходження найбільш схожого зображення на  plank8 :  plank9
Результат знаходження найбільш схожого зображення на  plank9 :  plank7
Результат знаходження найбільш схожого зображення на  plank10 :  plank5
Результат знаходження найбільш схожого зображення на  cotton1 :  cotton5
Результат знаходження найбільш схожого зображення на  cotton2 :  cotton3
Результат знаходження найбільш схожого зображення на  cotton3 :  cotton2
Результат знаходження найбільш схожого зображення на  cotton4 :  cotton9
Результат знаходження найбільш схожого зображення на  cotton5 :  cotton2
Результат знаходження найбільш схожого зображення на  cotton6 :  cotton1
Результат знаходження найбільш схожого зображення на  cotton7 :  cotton10
Результат знаходження найбільш схожого зображення на  cotton8 :  cotton1
Результат знаходження найбільш схожого зображення на  cotton9 :  cotton2
Результат знаходження найбільш схожого зображення на  cotton10 :  cotton7
```

Рисунок 3.23 – Результат виконання коду

3.6 Донавчання нейронної мережі MobileNetV2

3.6.1 Розширення датасету текстур та підготовка до завантаження в модель

Навчання нейронних мереж потребує дуже великих датасетів. Але в даній роботі використовується вже навчена нейронна мережа MobileNetV2, для якої необхідно перенавчити лише шар прийняття остаточного рішення – до якого класу належить зображення. Тобто щоб донавчити модель необхідно наш датасет текстур розширити. Однак кількість зразків зображень одного класу може буди значно меншою у порівнянні і кількістю для навчання моделі з нуля. В роботі було вирішено розширити датасет до 100 зразків у кожному класі.

Модель MobileNetV2 була навчена на датасеті ImageNet, де розміри зображень дорівнювали 224×224 пікселів. Тому необхідно всі зображення перед навчанням перевести до даного розміру. На рисунку 3.23, 3.24 наведені прикладі випадково обраних зображень з розширеного датасету, а на рисунку 3.25 показаний лістинг коду для формування масиву x та вектору відповідей y для 5-і класів brick, grass, fabric, wood, laminate.

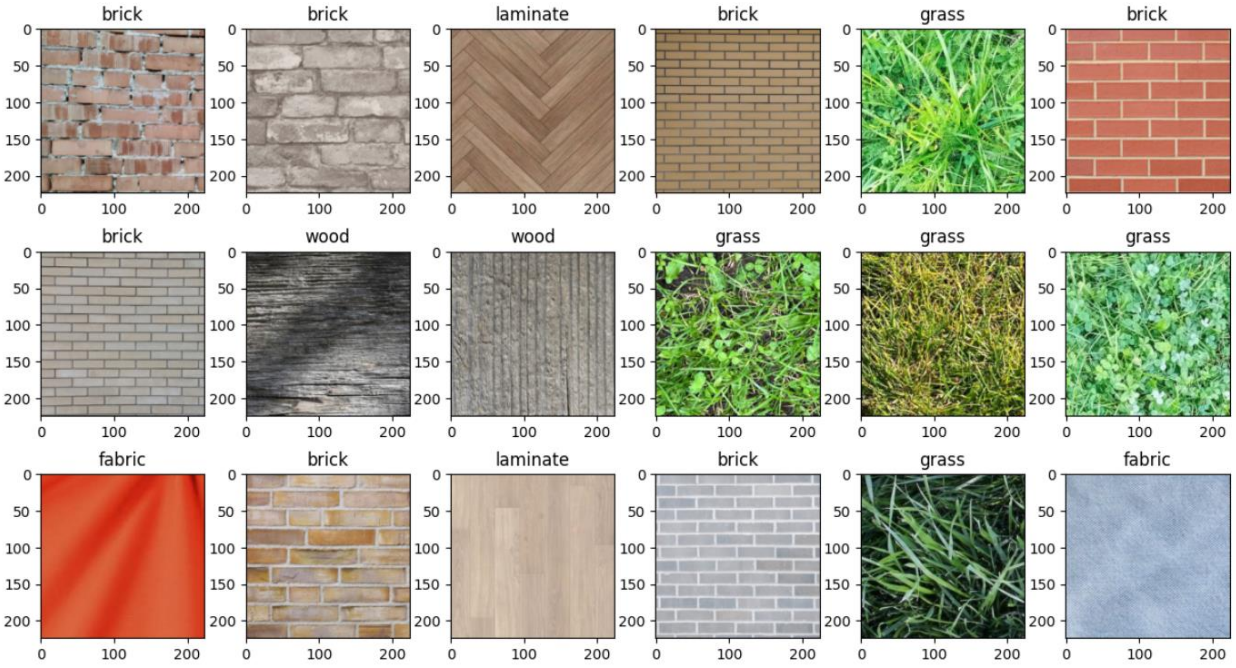


Рисунок 3.23 – Приклади розширеного датасету на 5 класів (500 зображень)

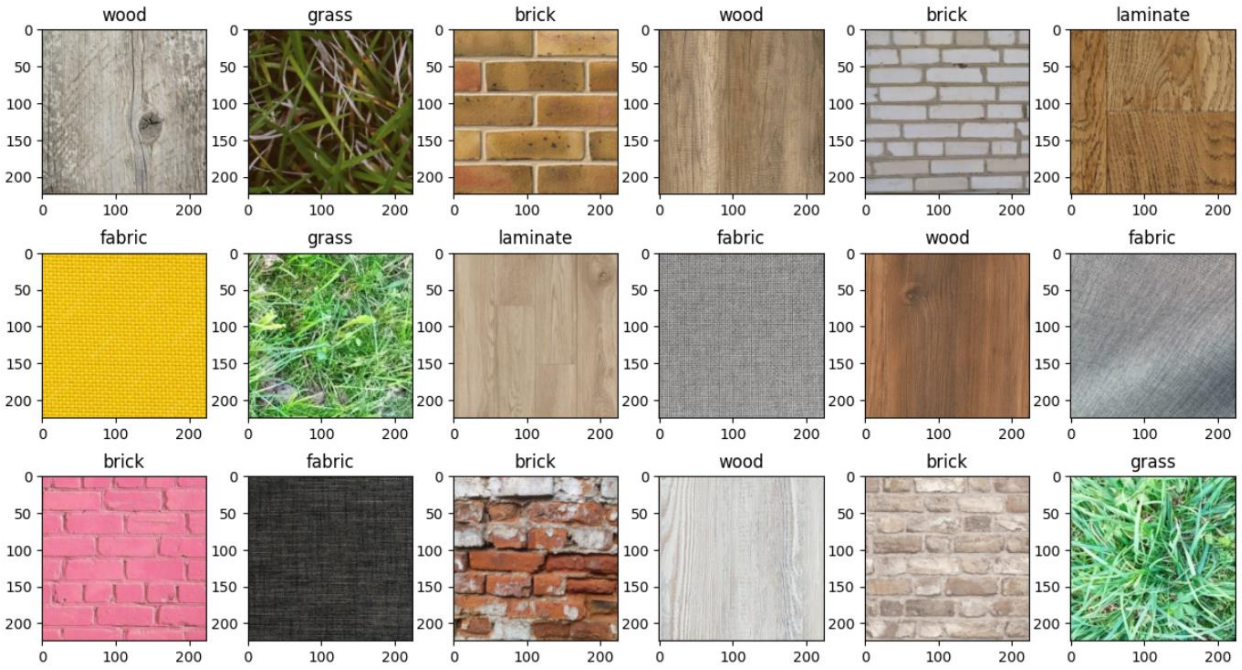


Рисунок 3.24 – Приклади розширеного датасету на 5 класів (500 зображень)

```
# 500 зображень (5 класів по 100 зразків)
# запис усіх зображень в x, формування вектора відповідей y
# приведення зображень до розміру 224 (цього вимагає вхід моделі MobileNetV2)
# brick, grass, fabric, wood, laminate
size = 224
x500, y500 = [], []
name_img = []
folder = "Texture500"
samples = os.listdir(folder)
for sample in tqdm(samples):
    img = cv2.cvtColor(cv2.imread(os.path.join(folder, sample)), cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (size, size))
    name_img.append(sample)
    x500.append(img)
    if 'brick' in sample:
        y500.append(0)
    elif 'fabric' in sample:
        y500.append(1)
    elif 'grass' in sample:
        y500.append(2)
    elif 'laminate' in sample:
        y500.append(3)
    elif 'wood' in sample:
        y500.append(4)
    else:
        raise ValueError()
```

Рисунок 3.25 – Лістинг коду для формування масиву x та вектору відповідей y для 5-і класів brick, fabric, grass, laminate, wood

3.6.2 Донавчання MobileNetV2

Для навчання моделі треба завантажити ряд бібліотек (рис. 3.26), а також задати назва класів (рис. 3.27).

```
# import the necessary packages
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
#import argparseimport os
from tqdm import tqdm
from time import time
import tensorflow as tf
import cv2
from tensorflow import keras
from keras.callbacks import EarlyStopping, ModelCheckpoint
# Set the seeds for reproducibility
from numpy.random import seed
from tensorflow.random import set_seed
seed_value = 1234578790
seed(seed_value)
set_seed(seed_value)
import shutil
```

Рисунок 3.26 – Лістинг для завантаження бібліотек щодо донавчання нейромережі MobileNetV2

```
# Mapping from class ID to class name
classes500 = {0:'brick', 1:'fabric', 2:'grass', 3:'laminate', 4:'wood'}
```

Рисунок 3.27 – Лістинг для задання класів

Далі необхідно поділити датасет з 500 зображень на частину для тренування та тестування. На частину для тестування вирішено залишити 20 відсотків (рис. 3.28). Оскільки під час навчання планується використання метрики softmax, вектор відповідей необхідно перекодувати у формат one hot encoding розміром 5 (рис. 3.29).

```
# поділ датасету для тренування і тестування
x500_train, x500_test, y500_train, y500_test = train_test_split(x500, y500, train_size=0.8, shuffle=True)
x500_train = (x500_train)/255
x500_test = (x500_test)/255
print(x500_train.shape, len(y500_train))
print(x500_test.shape, len(y500_test))
```

```
(400, 224, 224, 3) 400
(100, 224, 224, 3) 100
```

Рисунок 3.28 – Лістинг коду для поділу датасету для тренування та тестування

```
# оскільки під час навчання планується використання метрики softmax,
# вектор відповідей у необхідно перекодувати у формат one hot encoding
# для 5 класів
y500_train = tf.keras.utils.to_categorical(y500_train, 5)
y500_test = tf.keras.utils.to_categorical(y500_test, 5)
# зображення перших 4 векторів з масиву відповідей у в форматі one hot encoding
print(y500_train[0:4])
print(y500_test[0:4])
```

```
[[0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]]
[[0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]]
```

Рисунок 3.29 – Лістинг коду для перекодування вектору відповіді у формат one hot encoding

Для донавчання моделі MobileNetV2 треба завантажити модель з наступними налаштуваннями (рис. 3.30):

– використати ваги для датасету ImageNet;

– завантаження моделі без повнозв'язних підсумкових шарів (include_top=False);

– задання вхідного формату даних input_shape=(224, 224, 3);

– заморожування завантаженої моделі (щоб шари, які формують ознаки, не навчалися).

Модель MobileNetV2 без підзв'язних шарів має 2 257 984 параметрів та важить 8,61 MB (Non-trainable params).

На рисунку 3.31 наведено створення повнозв'язних шарів для навчання задачі класифікації текстур для 5 класів. Створені шари додали 8 028 933 параметрів (30,63 MB), які необхідно натренувати (навчити). Усього модель має 10 286 917 параметрів (39,24 MB), 2 257 984 з них заморожені.

```
# створення екземпляра моделі MobileNetV2 з такими налаштуваннями:
# - завантаження ваг для датасету Imagenet
# - завантаження моделі без повнозв'язних підсумкових шарів (include_top=False)
# - назначення вхідного формату даних input_shape=(224, 224, 3)
# - заморожування завантаженої моделі (щоб шари, які формують ознаки, не навчалися)
baseModel = MobileNetV2(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
baseModel.trainable = False
baseModel.summary()
# модель без підзв'язних шарів має 2257984 параметрів (Non-trainable params, 8.61 MB)
```

Рисунок 3.30 – Лістинг коду для завантаження моделі MobileNetV2 із заморожуванням шарів для отримання ознак

```
# створення повнозв'язних шарів для навчання задачі класифікації текстур (5 класів)
# створені шари додали 8028933 параметрів (30.63 MB), які необхідно натренувати (навчити)
# усього модель має 10286917 параметрів (39.24 MB), 2257984 з них заморожені
headModel = baseModel.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(5, activation="softmax")(headModel)
model = Model(inputs=baseModel.input, outputs=headModel)
model.summary()
```

Рисунок 3.31 – Лістинг коду для створення повнозв’язних шарів для класифікації на 5 класів

Донавчання проводилося для 100 епох, где розмір банчу дорівнював 40. Мережа навчалась 87,98 мс и отримала точність на тренувальному датасеті 0,997 та на валідаційному – 1,00 (рис. 3.33).

```
epochs = 100
batch_size = 40

# Compile the model
model.compile(loss="categorical_crossentropy", optimizer=Adam(1e-6), metrics=["accuracy"])
# Train the model
start = time()
# history = model.fit(x500_train, y500_train, batch_size=batch_size, epochs=epochs, validation_split=0.1, callbacks=callbacks_list)
history = model.fit(x500_train, y500_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
print('Elapsed time', time() - start)
```

Рисунок 3.32 – Лістинг коду для створення повнозв'язних шарів для класифікації на 5 класів

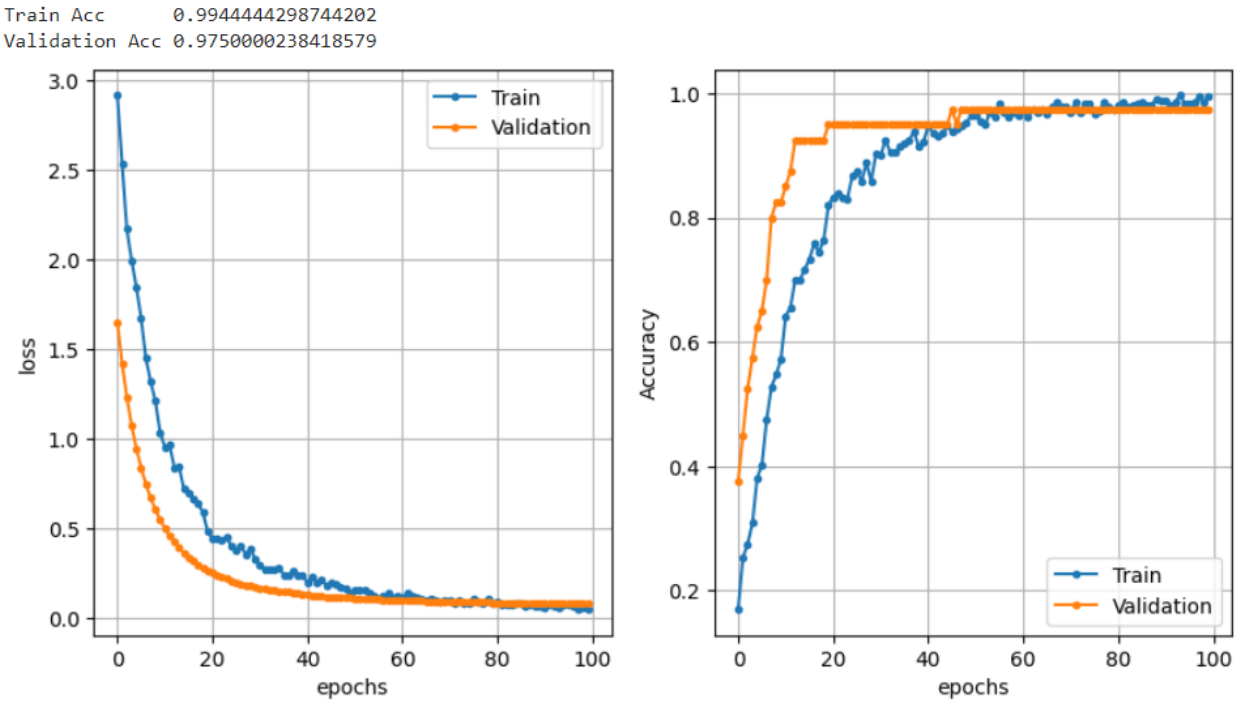


Рисунок 3.33 – Ілюстрація змін loss та accuracy для тренувальної та валідаційної частини датасету

Приклади результатів класифікації наведені на рисунках 3.34 та 3.35.

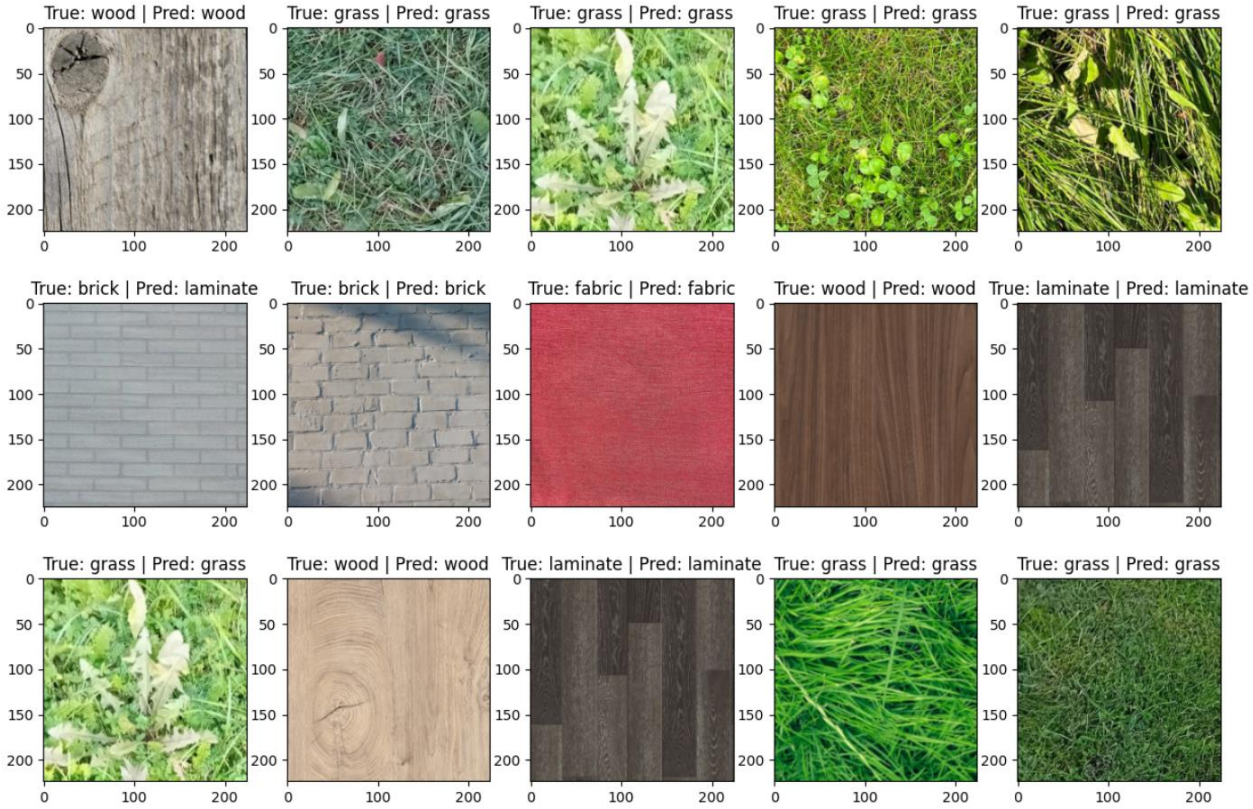


Рисунок 3.34 – Приклади результатів класифікації

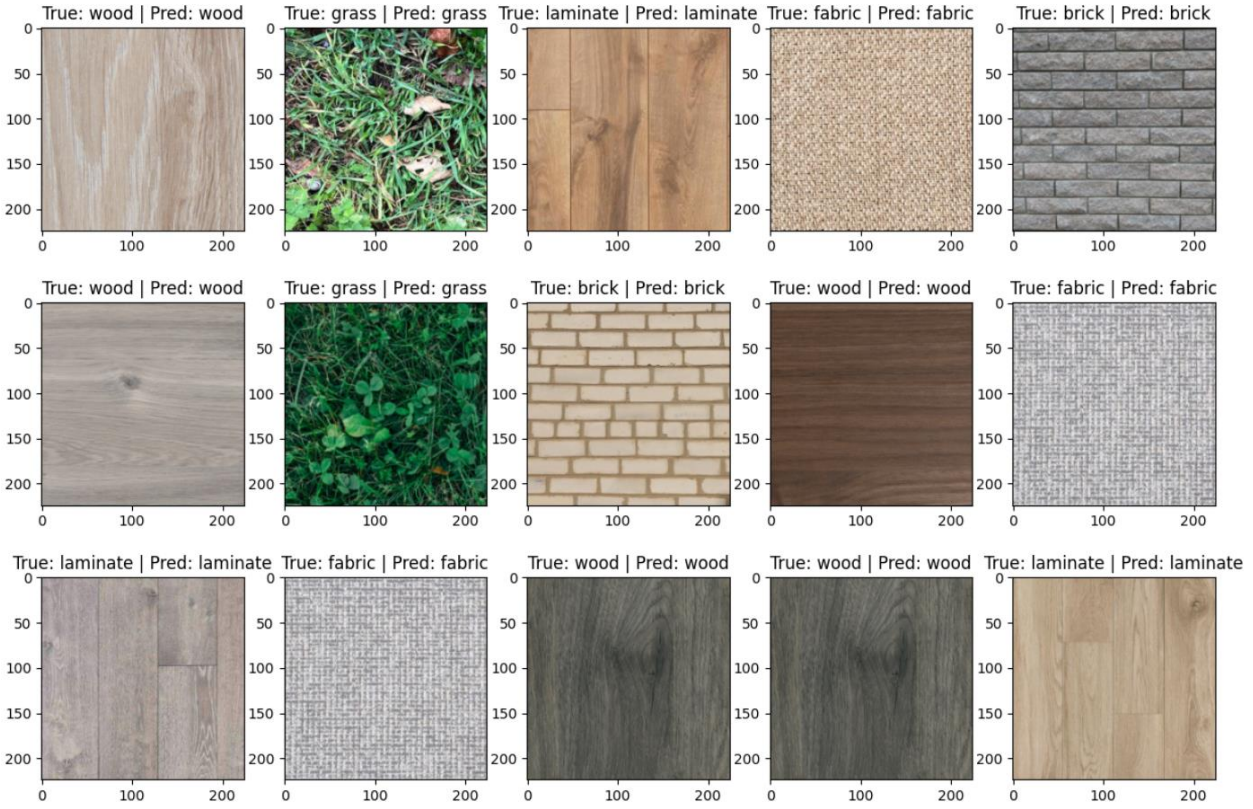


Рисунок 3.35 – Приклади результатів класифікації

3.7 Оцінка у порівняльному аспекті якості та швидкодії пошуку текстур за розглянутими методами

3.7.1 Оцінка якості класифікації методом ЛБШ та матриць збігів

Оцінка ефективності алгоритму класифікації часто здійснюється через аналіз показників класифікатора. Для цього використовуються критерії *TP* (True Positives), *TN* (True Negatives), *FP* (False Positives), *FN* (False Negatives).

Важливими метриками оцінки ефективності класифікатора є Precision (Точність) та Recall (Повнота).

Результати виконання цих метрик Precision та Recall для кожного зображення, а також для обох методів, відображено на рисунках 3.36 – 3.39.

```
Для зображення  grass1 TP -  6 , TN -  4 , FP -  11 , FN -  29 , тоді precision =  0.35294117647058826 , a recall =  0.17142857142857143
Для зображення  grass2 TP -  6 , TN -  4 , FP -  13 , FN -  27 , тоді precision =  0.3157894736842105 , a recall =  0.18181818181818182
Для зображення  grass3 TP -  5 , TN -  5 , FP -  9 , FN -  31 , тоді precision =  0.35714285714285715 , a recall =  0.1388888888888889
Для зображення  grass4 TP -  5 , TN -  5 , FP -  13 , FN -  27 , тоді precision =  0.2777777777777778 , a recall =  0.15625
Для зображення  grass5 TP -  6 , TN -  4 , FP -  10 , FN -  30 , тоді precision =  0.375 , a recall =  0.16666666666666666
Для зображення  grass6 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  grass7 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  grass8 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  grass9 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  grass10 TP -  6 , TN -  4 , FP -  10 , FN -  30 , тоді precision =  0.375 , a recall =  0.16666666666666666
Для зображення  brick1 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick2 TP -  4 , TN -  6 , FP -  13 , FN -  27 , тоді precision =  0.23529411764705882 , a recall =  0.12903225806451613
Для зображення  brick3 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick4 TP -  3 , TN -  7 , FP -  15 , FN -  25 , тоді precision =  0.16666666666666666 , a recall =  0.10714285714285714
Для зображення  brick5 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick6 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick7 TP -  2 , TN -  8 , FP -  4 , FN -  36 , тоді precision =  0.3333333333333333 , a recall =  0.05263157894736842
Для зображення  brick8 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick9 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  brick10 TP -  3 , TN -  7 , FP -  13 , FN -  27 , тоді precision =  0.1875 , a recall =  0.1
Для зображення  wood1 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  wood2 TP -  2 , TN -  8 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.047619047619047616
Для зображення  wood3 TP -  1 , TN -  9 , FP -  1 , FN -  39 , тоді precision =  0.5 , a recall =  0.025
Для зображення  wood4 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  wood5 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  wood6 TP -  3 , TN -  7 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.06976744186046512
Для зображення  wood7 TP -  2 , TN -  8 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.047619047619047616
Для зображення  wood8 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  wood9 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  wood10 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank1 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank2 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank3 TP -  2 , TN -  8 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.047619047619047616
Для зображення  plank4 TP -  1 , TN -  9 , FP -  1 , FN -  39 , тоді precision =  0.5 , a recall =  0.025
Для зображення  plank5 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank6 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank7 TP -  2 , TN -  8 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.047619047619047616
Для зображення  plank8 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank9 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  plank10 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , a recall =  0.024390243902439025
Для зображення  cotton1 TP -  10 , TN -  0 , FP -  9 , FN -  31 , тоді precision =  0.5263157894736842 , a recall =  0.24390243902439024
Для зображення  cotton2 TP -  7 , TN -  3 , FP -  8 , FN -  32 , тоді precision =  0.4666666666666667 , a recall =  0.1794871794871795
Для зображення  cotton3 TP -  7 , TN -  3 , FP -  8 , FN -  32 , тоді precision =  0.4666666666666667 , a recall =  0.1794871794871795
Для зображення  cotton4 TP -  10 , TN -  0 , FP -  9 , FN -  31 , тоді precision =  0.5263157894736842 , a recall =  0.24390243902439024
Для зображення  cotton5 TP -  8 , TN -  2 , FP -  10 , FN -  30 , тоді precision =  0.4444444444444444 , a recall =  0.21052631578947367
Для зображення  cotton6 TP -  6 , TN -  4 , FP -  4 , FN -  36 , тоді precision =  0.6 , a recall =  0.14285714285714285
Для зображення  cotton7 TP -  8 , TN -  2 , FP -  9 , FN -  31 , тоді precision =  0.47058823529411764 , a recall =  0.20512820512820512
Для зображення  cotton8 TP -  9 , TN -  1 , FP -  9 , FN -  31 , тоді precision =  0.5 , a recall =  0.225
Для зображення  cotton9 TP -  6 , TN -  4 , FP -  5 , FN -  35 , тоді precision =  0.5454545454545454 , a recall =  0.14634146341463414
Для зображення  cotton10 TP -  5 , TN -  5 , FP -  4 , FN -  36 , тоді precision =  0.5555555555555556 , a recall =  0.12195121951219512
```

Рисунок 3.36 – Результат виконання коду обчислення Precision та Recall для кожного зображення для методу ЛБШ

```
Загальний результат для precision =  0.7415690619150372 , а для recall =  0.08280656990882522
```

Рисунок 3.37 – Результат виконання коду обчислення Precision та Recall для всіх зображень для методу ЛБШ

```
Для зображення  grass1 TP -  7 , TN -  3 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.14893617021276595
Для зображення  grass2 TP -  4 , TN -  6 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.09090909090909091
Для зображення  grass3 TP -  6 , TN -  4 , FP -  2 , FN -  38 , тоді precision =  0.75 , а recall =  0.13636363636363635
Для зображення  grass4 TP -  7 , TN -  3 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.14893617021276595
Для зображення  grass5 TP -  6 , TN -  4 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.13043478260869565
Для зображення  grass6 TP -  4 , TN -  6 , FP -  3 , FN -  37 , тоді precision =  0.5714285714285714 , а recall =  0.0975609756097561
Для зображення  grass7 TP -  3 , TN -  7 , FP -  3 , FN -  37 , тоді precision =  0.5 , а recall =  0.075
Для зображення  grass8 TP -  4 , TN -  6 , FP -  2 , FN -  38 , тоді precision =  0.6666666666666666 , а recall =  0.09523809523809523
Для зображення  grass9 TP -  7 , TN -  3 , FP -  1 , FN -  39 , тоді precision =  0.875 , а recall =  0.15217391304347827
Для зображення  grass10 TP -  6 , TN -  4 , FP -  1 , FN -  39 , тоді precision =  0.8571428571428571 , а recall =  0.13333333333333333
Для зображення  brick1 TP -  5 , TN -  5 , FP -  4 , FN -  36 , тоді precision =  0.5555555555555556 , а recall =  0.12195121951219512
Для зображення  brick2 TP -  2 , TN -  8 , FP -  5 , FN -  35 , тоді precision =  0.2857142857142857 , а recall =  0.05405405405405406
Для зображення  brick3 TP -  3 , TN -  7 , FP -  3 , FN -  37 , тоді precision =  0.5 , а recall =  0.075
Для зображення  brick4 TP -  6 , TN -  4 , FP -  4 , FN -  36 , тоді precision =  0.6 , а recall =  0.14285714285714285
Для зображення  brick5 TP -  5 , TN -  5 , FP -  5 , FN -  35 , тоді precision =  0.5 , а recall =  0.125
Для зображення  brick6 TP -  4 , TN -  6 , FP -  3 , FN -  37 , тоді precision =  0.5714285714285714 , а recall =  0.0975609756097561
Для зображення  brick7 TP -  3 , TN -  7 , FP -  4 , FN -  36 , тоді precision =  0.42857142857142855 , а recall =  0.07692307692307693
Для зображення  brick8 TP -  4 , TN -  6 , FP -  4 , FN -  36 , тоді precision =  0.5 , а recall =  0.1
Для зображення  brick9 TP -  3 , TN -  7 , FP -  3 , FN -  37 , тоді precision =  0.5 , а recall =  0.075
Для зображення  brick10 TP -  1 , TN -  9 , FP -  3 , FN -  37 , тоді precision =  0.25 , а recall =  0.02631578947368421
Для зображення  wood1 TP -  4 , TN -  6 , FP -  4 , FN -  36 , тоді precision =  0.5 , а recall =  0.1
Для зображення  wood2 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.024390243902439025
Для зображення  wood3 TP -  3 , TN -  7 , FP -  4 , FN -  36 , тоді precision =  0.42857142857142855 , а recall =  0.07692307692307693
Для зображення  wood4 TP -  4 , TN -  6 , FP -  2 , FN -  38 , тоді precision =  0.6666666666666666 , а recall =  0.09523809523809523
Для зображення  wood5 TP -  1 , TN -  9 , FP -  1 , FN -  39 , тоді precision =  0.5 , а recall =  0.025
Для зображення  wood6 TP -  3 , TN -  7 , FP -  4 , FN -  36 , тоді precision =  0.42857142857142855 , а recall =  0.07692307692307693
Для зображення  wood7 TP -  4 , TN -  6 , FP -  4 , FN -  36 , тоді precision =  0.5 , а recall =  0.1
Для зображення  wood8 TP -  3 , TN -  7 , FP -  2 , FN -  38 , тоді precision =  0.6 , а recall =  0.07317073170731707
Для зображення  wood9 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.024390243902439025
Для зображення  wood10 TP -  2 , TN -  8 , FP -  8 , FN -  32 , тоді precision =  0.2 , а recall =  0.058823529411764705
Для зображення  plank1 TP -  1 , TN -  9 , FP -  1 , FN -  39 , тоді precision =  0.5 , а recall =  0.025
Для зображення  plank2 TP -  3 , TN -  7 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.06976744186046512
Для зображення  plank3 TP -  4 , TN -  6 , FP -  4 , FN -  36 , тоді precision =  0.5 , а recall =  0.1
Для зображення  plank4 TP -  3 , TN -  7 , FP -  1 , FN -  39 , тоді precision =  0.75 , а recall =  0.07142857142857142
Для зображення  plank5 TP -  7 , TN -  3 , FP -  1 , FN -  39 , тоді precision =  0.875 , а recall =  0.15217391304347827
Для зображення  plank6 TP -  4 , TN -  6 , FP -  3 , FN -  37 , тоді precision =  0.5714285714285714 , а recall =  0.0975609756097561
Для зображення  plank7 TP -  5 , TN -  5 , FP -  2 , FN -  38 , тоді precision =  0.7142857142857143 , а recall =  0.11627906976744186
Для зображення  plank8 TP -  4 , TN -  6 , FP -  4 , FN -  36 , тоді precision =  0.5 , а recall =  0.1
Для зображення  plank9 TP -  5 , TN -  5 , FP -  1 , FN -  39 , тоді precision =  0.8333333333333334 , а recall =  0.11363636363636363
Для зображення  plank10 TP -  4 , TN -  6 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.09090909090909091
Для зображення  cotton1 TP -  7 , TN -  3 , FP -  1 , FN -  39 , тоді precision =  0.875 , а recall =  0.15217391304347827
Для зображення  cotton2 TP -  9 , TN -  1 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.1836734693877551
Для зображення  cotton3 TP -  9 , TN -  1 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.1836734693877551
Для зображення  cotton4 TP -  5 , TN -  5 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.1111111111111111
Для зображення  cotton5 TP -  8 , TN -  2 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.16666666666666666
Для зображення  cotton6 TP -  1 , TN -  9 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.024390243902439025
Для зображення  cotton7 TP -  8 , TN -  2 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.16666666666666666
Для зображення  cotton8 TP -  8 , TN -  2 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.16666666666666666
Для зображення  cotton9 TP -  9 , TN -  1 , FP -  0 , FN -  40 , тоді precision =  1.0 , а recall =  0.1836734693877551
Для зображення  cotton10 TP -  8 , TN -  2 , FP -  1 , FN -  39 , тоді precision =  0.8888888888888888 , а recall =  0.1702127659574468
```

Рисунок 3.38 – Результат виконання коду обчислення Precision та Recall для кожного зображення для методу, на основі матриці збігів

```
Загальний результат для precision =  0.7148650793650794 , а для recall =  0.10408142584805292
```

Рисунок 3.39 – Результат виконання коду обчислення Precision та Recall для всіх зображень для методу, на основі матриці збігів

Також, розглянемо різницю роботи алгоритмів при використані різних критеріїв оцінки, а саме манхеттенської та евклідової метрики. Результат

порівняння манхеттенської та евклідової метрики та двох методів відображено на рисунках 3.40, 3.41.

```
Результат порівняння  cotton6  та  grass1 : манхеттенська -  0.0013388241038602945 , евклідова -  0.0003025322048508521
Результат порівняння  cotton6  та  grass2 : манхеттенська -  0.0019415761910232848 , евклідова -  0.0004445729355997569
Результат порівняння  cotton6  та  grass3 : манхеттенська -  0.0028140797334558825 , евклідова -  0.0006856389219004052
Результат порівняння  cotton6  та  grass4 : манхеттенська -  0.0018791049134497555 , евклідова -  0.0004216605481008681
Результат порівняння  cotton6  та  grass5 : манхеттенська -  0.0016607845530790446 , евклідова -  0.0003690498172581119
Результат порівняння  cotton6  та  grass6 : манхеттенська -  0.005648626100255368 , евклідова -  0.0011945015646993417
Результат порівняння  cotton6  та  grass7 : манхеттенська -  0.007459517468793048 , евклідова -  0.0015611492151118495
Результат порівняння  cotton6  та  grass8 : манхеттенська -  0.0071240879164556235 , евклідова -  0.001504609498261371
Результат порівняння  cotton6  та  grass9 : манхеттенська -  0.006041270639435067 , евклідова -  0.0012789618100434328
Результат порівняння  cotton6  та  grass10 : манхеттенська -  0.0021726236506433824 , евклідова -  0.0005263830962175378
Результат порівняння  cotton6  та  brick1 : манхеттенська -  0.004237527101926019 , евклідова -  0.0010049761045867209
Результат порівняння  cotton6  та  brick2 : манхеттенська -  0.0029993094649969354 , евклідова -  0.0006676145447752779
Результат порівняння  cotton6  та  brick3 : манхеттенська -  0.005282039510287935 , евклідова -  0.0010012554128986414
Результат порівняння  cotton6  та  brick4 : манхеттенська -  0.002151878207337622 , евклідова -  0.0004913088051352694
Результат порівняння  cotton6  та  brick5 : манхеттенська -  0.0053939495895243186 , евклідова -  0.0012252713121321574
Результат порівняння  cotton6  та  brick6 : манхеттенська -  0.0050092165009764525 , евклідова -  0.0009943016514575025
Результат порівняння  cotton6  та  brick7 : манхеттенська -  0.003305971436412532 , евклідова -  0.0006730450894554443
Результат порівняння  cotton6  та  brick8 : манхеттенська -  0.005257120828557087 , евклідова -  0.0010484553904091192
Результат порівняння  cotton6  та  brick9 : манхеттенська -  0.003947486367488697 , евклідова -  0.0007811846650080203
Результат порівняння  cotton6  та  brick10 : манхеттенська -  0.002759656719132966 , евклідова -  0.0006419914037446266
Результат порівняння  cotton6  та  wood1 : манхеттенська -  0.006020831479214038 , евклідова -  0.0014285131994060066
Результат порівняння  cotton6  та  wood2 : манхеттенська -  0.004542146902751153 , евклідова -  0.001019352069539253
Результат порівняння  cotton6  та  wood3 : манхеттенська -  0.007884264539527598 , евклідова -  0.001956153231636141
Результат порівняння  cotton6  та  wood4 : манхеттенська -  0.005085568833979999 , евклідова -  0.0009415382405222828
Результат порівняння  cotton6  та  wood5 : манхеттенська -  0.011431623551511297 , евклідова -  0.0034157959265487944
Результат порівняння  cotton6  та  wood6 : манхеттенська -  0.005466440834665487 , евклідова -  0.0014126209894321053
Результат порівняння  cotton6  та  wood7 : манхеттенська -  0.005294677308357453 , евклідова -  0.001343386761327155
Результат порівняння  cotton6  та  wood8 : манхеттенська -  0.005653816222096066 , евклідова -  0.0013346621106225553
Результат порівняння  cotton6  та  wood9 : манхеттенська -  0.004421678181784818 , евклідова -  0.0009282036492120803
Результат порівняння  cotton6  та  wood10 : манхеттенська -  0.007849921088569644 , евклідова -  0.002124716631410729
Результат порівняння  cotton6  та  plank1 : манхеттенська -  0.004624370201313792 , евклідова -  0.0010777100458835215
Результат порівняння  cotton6  та  plank2 : манхеттенська -  0.007276081481239403 , евклідова -  0.0023126046585861546
Результат порівняння  cotton6  та  plank3 : манхеттенська -  0.007079113984064614 , евклідова -  0.0017572527278879998
Результат порівняння  cotton6  та  plank4 : манхеттенська -  0.0077247386250375374 , евклідова -  0.0017982770341797476
Результат порівняння  cotton6  та  plank5 : манхеттенська -  0.007643169150634132 , евклідова -  0.002121482034559855
Результат порівняння  cotton6  та  plank6 : манхеттенська -  0.007908565411128596 , евклідова -  0.0020750538478981847
Результат порівняння  cotton6  та  plank7 : манхеттенська -  0.007300242411796257 , евклідова -  0.0018451619438221768
Результат порівняння  cotton6  та  plank8 : манхеттенська -  0.006776211646100077 , евклідова -  0.0018006033607637299
Результат порівняння  cotton6  та  plank9 : манхеттенська -  0.007129301821194515 , евклідова -  0.00187509125517756
Результат порівняння  cotton6  та  plank10 : манхеттенська -  0.007757963304520924 , евклідова -  0.0019843004402844183
Результат порівняння  cotton6  та  cotton1 : манхеттенська -  0.0016406489353553925 , евклідова -  0.0003636514152923952
Результат порівняння  cotton6  та  cotton2 : манхеттенська -  0.003281028597962622 , евклідова -  0.000701677729539402
Результат порівняння  cotton6  та  cotton3 : манхеттенська -  0.0032514684340533085 , евклідова -  0.0006914742114068196
Результат порівняння  cotton6  та  cotton4 : манхеттенська -  0.0018516390931372553 , евклідова -  0.0004000071919826922
Результат порівняння  cotton6  та  cotton5 : манхеттенська -  0.002435601926317402 , евклідова -  0.0005174076705831624
Результат порівняння  cotton6  та  cotton6 : манхеттенська -  0.0 , евклідова -  0.0
Результат порівняння  cotton6  та  cotton7 : манхеттенська -  0.0017844405828737748 , евклідова -  0.00040262362142783254
Результат порівняння  cotton6  та  cotton8 : манхеттенська -  0.0020376467237285542 , евклідова -  0.00046023382763219796
Результат порівняння  cotton6  та  cotton9 : манхеттенська -  0.0018674962660845589 , евклідова -  0.0004070526954706676
Результат порівняння  cotton6  та  cotton10 : манхеттенська -  0.001994301533418886 , евклідова -  0.0004200261876754409
```

Рисунок 3.40 – Результат виконання коду порівняння манхеттенської та евклідової метрики для методу ЛБШ

```
Результат порівняння  cotton6  та  grass1 : манхеттенська -  0.27022298304692843 , евклідова -  0.27022298304692843
Результат порівняння  cotton6  та  grass2 : манхеттенська -  0.3617535125639468 , евклідова -  0.3617535125639468
Результат порівняння  cotton6  та  grass3 : манхеттенська -  0.2964512511674159 , евклідова -  0.2964512511674159
Результат порівняння  cotton6  та  grass4 : манхеттенська -  0.30656614201815413 , евклідова -  0.30656614201815413
Результат порівняння  cotton6  та  grass5 : манхеттенська -  0.27160421854184885 , евклідова -  0.27160421854184885
Результат порівняння  cotton6  та  grass6 : манхеттенська -  0.38737379919232 , евклідова -  0.38737379919232
Результат порівняння  cotton6  та  grass7 : манхеттенська -  0.4123799813082721 , евклідова -  0.4123799813082721
Результат порівняння  cotton6  та  grass8 : манхеттенська -  0.4011424690570124 , евклідова -  0.4011424690570124
Результат порівняння  cotton6  та  grass9 : манхеттенська -  0.30324817558089273 , евклідова -  0.30324817558089273
Результат порівняння  cotton6  та  grass10 : манхеттенська -  0.2417322027241103 , евклідова -  0.2417322027241103
Результат порівняння  cotton6  та  brick1 : манхеттенська -  0.42311957757554547 , евклідова -  0.42311957757554547
Результат порівняння  cotton6  та  brick2 : манхеттенська -  0.36352724799092134 , евклідова -  0.36352724799092134
Результат порівняння  cotton6  та  brick3 : манхеттенська -  0.3568102790303931 , евклідова -  0.3568102790303931
Результат порівняння  cotton6  та  brick4 : манхеттенська -  0.4199171647324464 , евклідова -  0.4199171647324464
Результат порівняння  cotton6  та  brick5 : манхеттенська -  0.41852578114837824 , евклідова -  0.41852578114837824
Результат порівняння  cotton6  та  brick6 : манхеттенська -  0.4988982618579665 , евклідова -  0.4988982618579665
Результат порівняння  cotton6  та  brick7 : манхеттенська -  0.40477613036623256 , евклідова -  0.40477613036623256
Результат порівняння  cotton6  та  brick8 : манхеттенська -  0.5196444593383351 , евклідова -  0.5196444593383351
Результат порівняння  cotton6  та  brick9 : манхеттенська -  0.5847124830470146 , евклідова -  0.5847124830470146
Результат порівняння  cotton6  та  brick10 : манхеттенська -  0.5274465497354155 , евклідова -  0.5274465497354155
Результат порівняння  cotton6  та  wood1 : манхеттенська -  0.393154766050056 , евклідова -  0.393154766050056
Результат порівняння  cotton6  та  wood2 : манхеттенська -  0.6860769137837547 , евклідова -  0.6860769137837547
Результат порівняння  cotton6  та  wood3 : манхеттенська -  0.561308604078055 , евклідова -  0.561308604078055
Результат порівняння  cotton6  та  wood4 : манхеттенська -  0.3431406475718563 , евклідова -  0.3431406475718563
Результат порівняння  cotton6  та  wood5 : манхеттенська -  0.7537574337855117 , евклідова -  0.7537574337855117
Результат порівняння  cotton6  та  wood6 : манхеттенська -  0.4923041259581504 , евклідова -  0.4923041259581504
Результат порівняння  cotton6  та  wood7 : манхеттенська -  0.3535781490447589 , евклідова -  0.3535781490447589
Результат порівняння  cotton6  та  wood8 : манхеттенська -  0.29399779465705816 , евклідова -  0.29399779465705816
Результат порівняння  cotton6  та  wood9 : манхеттенська -  0.541331992390054 , евклідова -  0.541331992390054
Результат порівняння  cotton6  та  wood10 : манхеттенська -  0.5124266475478277 , евклідова -  0.5124266475478277
Результат порівняння  cotton6  та  plank1 : манхеттенська -  0.6595034539945485 , евклідова -  0.6595034539945485
Результат порівняння  cotton6  та  plank2 : манхеттенська -  0.6597923516541416 , евклідова -  0.6597923516541416
Результат порівняння  cotton6  та  plank3 : манхеттенська -  0.4676847322189122 , евклідова -  0.4676847322189122
Результат порівняння  cotton6  та  plank4 : манхеттенська -  0.6571345825236733 , евклідова -  0.6571345825236733
Результат порівняння  cotton6  та  plank5 : манхеттенська -  0.5940890796380095 , евклідова -  0.5940890796380095
Результат порівняння  cotton6  та  plank6 : манхеттенська -  0.5918769552020385 , евклідова -  0.5918769552020385
Результат порівняння  cotton6  та  plank7 : манхеттенська -  0.49910258849496497 , евклідова -  0.49910258849496497
Результат порівняння  cotton6  та  plank8 : манхеттенська -  0.47030890932720243 , евклідова -  0.47030890932720243
Результат порівняння  cotton6  та  plank9 : манхеттенська -  0.5022684410747755 , евклідова -  0.5022684410747755
Результат порівняння  cotton6  та  plank10 : манхеттенська -  0.6259823430150551 , евклідова -  0.6259823430150551
Результат порівняння  cotton6  та  cotton1 : манхеттенська -  0.1535131134945716 , евклідова -  0.1535131134945716
Результат порівняння  cotton6  та  cotton2 : манхеттенська -  0.15976433432634404 , евклідова -  0.15976433432634404
Результат порівняння  cotton6  та  cotton3 : манхеттенська -  0.1647954391590436 , евклідова -  0.1647954391590436
Результат порівняння  cotton6  та  cotton4 : манхеттенська -  0.24117486647049527 , евклідова -  0.24117486647049527
Результат порівняння  cotton6  та  cotton5 : манхеттенська -  0.14576290300601205 , евклідова -  0.14576290300601205
Результат порівняння  cotton6  та  cotton6 : манхеттенська -  0.0 , евклідова -  0.0
Результат порівняння  cotton6  та  cotton7 : манхеттенська -  0.19529414399311304 , евклідова -  0.19529414399311304
Результат порівняння  cotton6  та  cotton8 : манхеттенська -  0.19124081981752536 , евклідова -  0.19124081981752536
Результат порівняння  cotton6  та  cotton9 : манхеттенська -  0.1853657067914797 , евклідова -  0.1853657067914797
Результат порівняння  cotton6  та  cotton10 : манхеттенська -  0.20564612161140525 , евклідова -  0.20564612161140525
```

Рисунок 3.41 – Результат виконання коду порівняння манхеттенської та евклідової метрики для методу, на основі матриці збігів

На основі оцінки класифікатора було встановлено, що метод лінійних бінарних шаблонів (ЛБШ) виявився більш ефективним за точністю. Це означає, що більша частка документів, визначених цим методом як належні до певного класу, дійсно належать до цього класу порівняно з усіма документами, які система віднесла до цього класу. Однак метод на основі матриці збігів показав кращі результати за показником повноти, тобто здатністю виявляти позитивні випадки серед усіх можливих позитивних випадків, хоча загальні показники повноти були низькими для обох методів.

Ці два показники — точність та повнота — є ключовими для оцінки ефективності класифікатора. Оскільки кожен з методів має свої переваги в різних аспектах, не можна однозначно стверджувати, який з методів є кращим. Це залежить від конкретних вимог та цілей дослідження. В таких випадках часто використовується додатковий показник, такий як F-міра, який комбінує точність та повноту, щоб надати більш комплексну оцінку ефективності алгоритму.

3.7.2 Оцінка якості класифікації на основі моделі MobileNetV2

Для передбачення класу за допомогою донавченної в розділі 3.6.2 мережі MobileNetV2 точність класифікації набагато вища. На тестовому наборі даних точність дорівнює 0.95 (рис.3.42).

```
# оцінка роботи на тестовому наборі даних
ev = model.evaluate(x500_test, y500_test)
print('Test loss  ', ev[0])
print('Test metric', ev[1])

4/4 [==============================] - 2s
Test loss   0.14514638483524323
Test metric 0.949999988079071
```

Рисунок 3.42 – Значення accuracy моделі на тестовому наборі

Далі були розраховані accuracy (рис. 3.43), precision, recall, f1-score для кожного класу (рис. 3.44).

```
# Compute and print the accuracy for each class
for class_id, class_name in classes500.items():
    mask = y500_true == class_id
    tp = np.sum(y500_pred[mask] == class_id)
    total = np.sum(mask)
    acc = tp/total
    print('Class -', class_name, ' acc', acc)

Class - brick  acc 0.9583333333333334
Class - fabric  acc 0.9166666666666666
Class - grass  acc 1.0
Class - laminate  acc 0.88
Class - wood  acc 1.0
```

Рисунок 3.43 – Лістинг коду та результат обчислення accuracy для кожного класу

```
from sklearn.metrics import classification_report
report = classification_report(y500_true, y500_pred)
print(report)
```

	precision	recall	f1-score	support
0	1.00	0.96	0.98	24
1	1.00	0.92	0.96	12
2	1.00	1.00	1.00	18
3	0.96	0.88	0.92	25
4	0.84	1.00	0.91	21
accuracy			0.95	100
macro avg	0.96	0.95	0.95	100
weighted avg	0.96	0.95	0.95	100

Рисунок 3.44 – Лістинг коду та результат обчислення precision, recall, f1-score для кожного класу

Таким чином, класифікація на основі навченої у 3.6.2 моделі MobileNetV2 набагато краща ніж класичними методами, середнє значення precision, recall дорівнює 0,96 та 0,95 відповідно. Але з рисунку 3.44 видно, що точність окремих класів, значно нижча, наприклад для класів 3 –laminate, 4 – wood, імовірно це пов'язано зі схожістю зображень у цих класах. Також з рисунку 3.44 видно, що під час розділення датасету на тренувальних та тестових набір не була збережена пропорціональність між кількістю зображень в класах. Тому було вирішено провести додаткові експерименти з метою підвищення точність моделі.

3.7.3 Підвищення якості класифікації на основі MobileNetV2

Для підвищення точності моделі по-перше необхідно збалансувати класи під час розподілу датасету на тренувальну та тестову вибірки. Для цього для функції train_test_split був використаний параметр stratify. Параметр stratify гарантує, що навчальний і тестовий набори матимуть ту саму пропорцію зразків кожного класу, що й вихідний датасет. В датасеті 5 класів по 100 зображень у кожному, необхідно виконати поділ так, щоб у тестовому

наборі було рівно 20 зображень від кожного класу, а в навчальному - решта 80. Для цього параметру tratify було призначено y500. Масив y500 має містити мітки класів для кожного зображення. Функція train_test_split збереже пропорцію класів у навчальному і тестовому наборах такою самою, як у вихідних даних (рис. 3.45).

```
# поділ датасету для тренування і тестування
x500_train, x500_test, y500_train, y500_test = train_test_split(x500, y500, train_size=0.8, shuffle=True, stratify=y500)
```

Рисунок 3.45 – Лістинг коду для пропорційного ділення датасету на тренувальний та тестовий набір

Також була використана можливість обрання найкращої моделі під час навчання та можливість ранньої зупинки (рис. 3.46, 3.47).

```
# Налаштування EarlyStopping, ModelCheckpoint
from keras.callbacks import EarlyStopping, ModelCheckpoint
callbacks_list = [EarlyStopping(monitor='loss', patience=8, restore_best_weights=True),
                  ModelCheckpoint(filepath='model_clasif5_best.h5', monitor='loss',
                                  save_best_only=True)]
```

Рисунок 3.46 – Налаштування функцій ранньої зупинки та збереження найкращої моделі

```
epochs = 100
batch_size = 40
# Compile the model
model.compile(loss="categorical_crossentropy", optimizer=Adam(1e-6), metrics=["accuracy"])
# Train the model
start = time.time()
history = model.fit(x500_train, y500_train, batch_size=batch_size, epochs=epochs, validation_split=0.1, callbacks=callbacks_list)
print('Elapsed time', time.time() - start)
```

Рисунок 3.47 – Навчання моделі з використання ранньої зупинки та збереженням найкращої моделі (за параметром функції втрат loss0

Модель навчалась тільки 63 із заданих 100 епох, досягла точності 0.98 на валідаційному наборі і під час навчання була збережена найкраща модель model_clasif5_best.h5, яка на тестовому наборі теж показала точність 0.98

(рис. 3.48). Точність класифікації для окремих класів також значно покращилась (рис. 3.49, 3.50), але все одно деякі класи мають більше помилок класифікації.

```
# оцінка роботи на тестовому наборі даних
ev = model_load.evaluate(x500_test, y500_test)
print('Test loss  ', ev[0])
print('Test metric', ev[1])

4/4 [==============================] - 1s 45ms/step
Test loss   0.1116376519203186
Test metric 0.9800000190734863
```

Рисунок 3.48 – Значення loss та accuracy для тестового набора

```
# Compute and print the accuracy for each class
for class_id, class_name in classes500.items():
    mask = y500_true == class_id
    tp = np.sum(y500_pred[mask] == class_id)
    total = np.sum(mask)
    acc = tp/total
    print('Class -', class_name, ' acc', acc)

Class - brick  acc 1.0
Class - fabric  acc 1.0
Class - grass  acc 1.0
Class - laminate  acc 0.95
Class - wood  acc 0.95
```

Рисунок 3.49 – Значення accuracy для кожного класу

```
from sklearn.metrics import classification_report
report = classification_report(y500_true, y500_pred)
print(report)

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.95      1.00      0.98        20
           2       1.00      1.00      1.00        20
           3       1.00      0.95      0.97        20
           4       0.95      0.95      0.95        20

    accuracy                           0.98       100
   macro avg       0.98      0.98      0.98       100
weighted avg       0.98      0.98      0.98       100
```

Рисунок 3.50 – Значення precision, recall, f1-score для кожного класу

3.7.4 Розширення датасету текстур та класифікація текстур з розширеного датасету на основі моделі MobileNetV2

Для реальних задач кількість класів може бути суттєво більша ніж 5, тому було вирішено розширити датасет текстур до 10 класів донавчити модель MobileNetV2 та оцінити значенні точності класифікації. Таким чином, в розширеному датасеті буде 10 класів: leaves, brick, grass, fabric, wood, laminate, water, asphalt, soil, bark (рис. 3.51, 3.52)

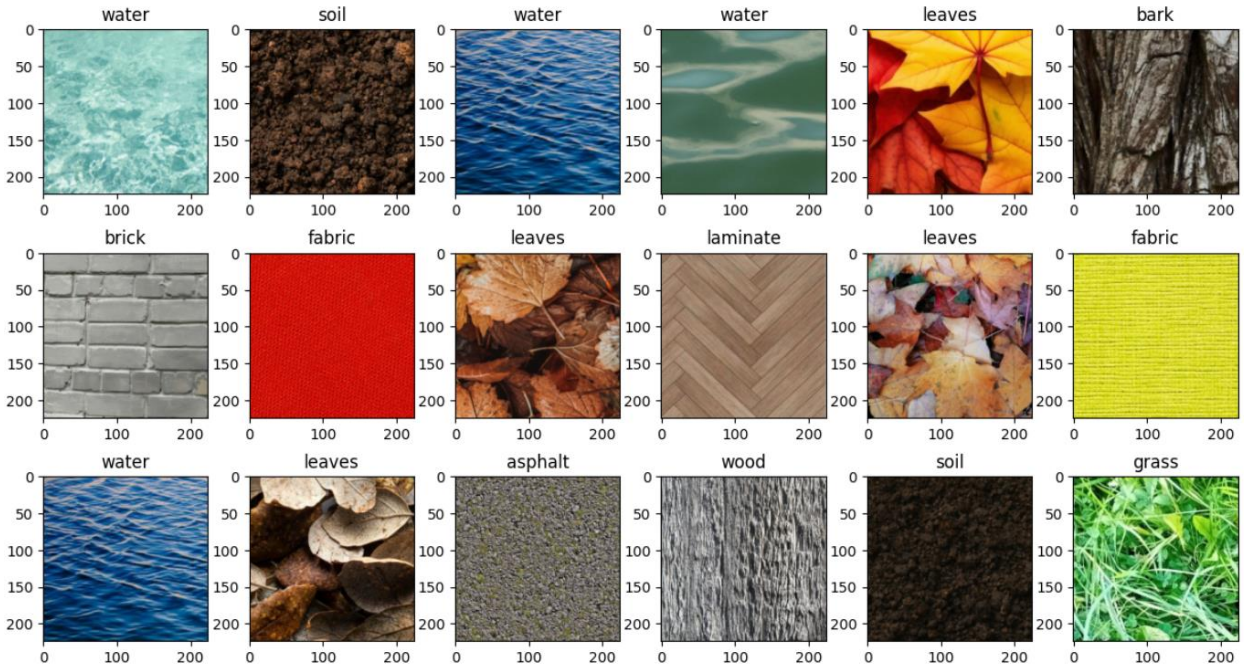


Рисунок 3.51 – Випадково вибрані зображення з розширеного датасету, що містить 10 класів

```
# Mapping from class ID to class name
classes1000 = {0:'leaves', 1:'brick', 2:'grass', 3:'fabric', 4:'wood', 5:'laminate', 6:'water', 7:'asphalt', 8:'soil', 9:'bark'}
```

Рисунок 3.52 – Лістинг для задання 10 класів

Далі були для навчання були використані налаштування, як на рисунках 3,45 – 3,47. Модель для класифікації на 10 класів донавчилась за 162,62 мс. Навчання було закінчено на 84 епохі з 100, точність моделі на 84 епохі на тренувальній виборці дорівнювала 0,973, на валідаційній – 0,975 (рис. 3.53).

При чому найкраща модель показала на тестовий виборці точність 0,9449999, тобто приблизно 0,95 (рис. 3.54). Точність для кожного класу наведена на рисунках 3.55, 3.56.

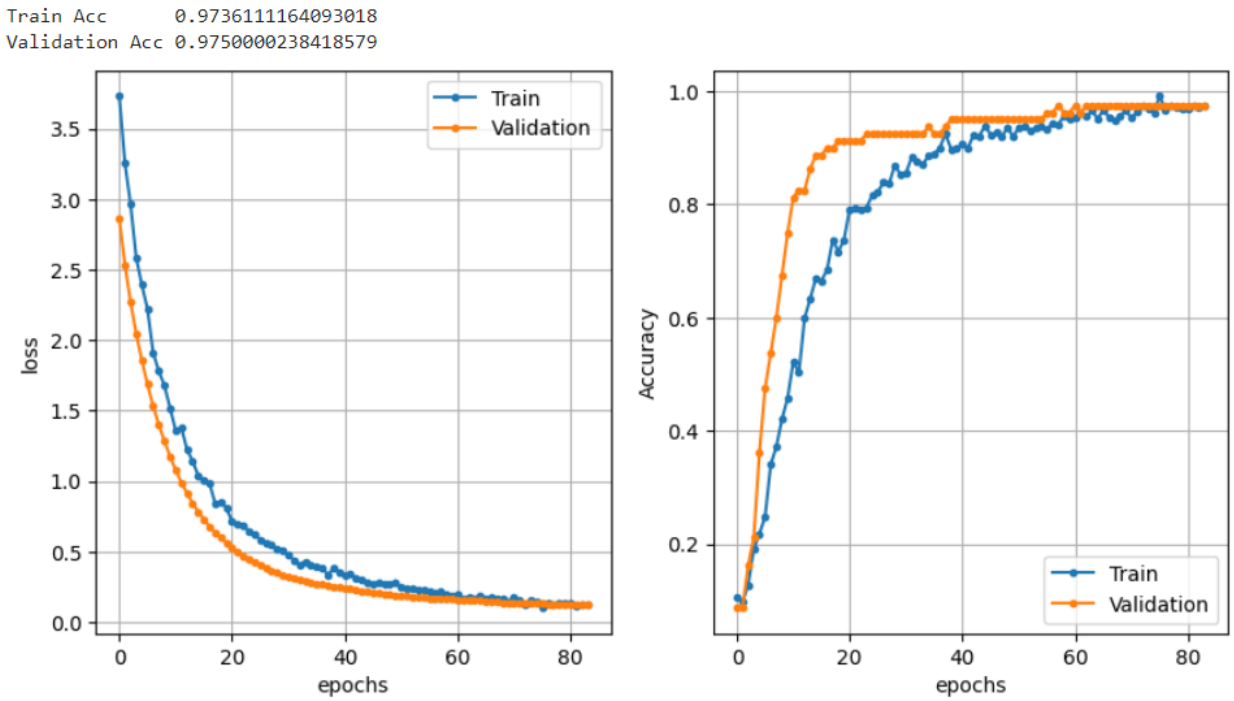


Рисунок 3.53 – Ілюстрація змін loss та accuracy для тренувальної та валідаційної частини розширеного датасету

```
# оцінка роботи на тестовому наборі даних
ev = model_load.evaluate(x_test, y_test)
print('Test loss  ', ev[0])
print('Test metric', ev[1])

7/7 [==============================] - 0s
Test loss   0.17850644886493683
Test metric 0.9449999928474426
```

Рисунок 3.54 – Значення loss та accuracy для тестового набора розширенного датасету

```
# Compute and print the accuracy for each class
for class_id, class_name in classes1000.items():
    mask = y_true == class_id
    tp = np.sum(y_pred[mask] == class_id)
    total = np.sum(mask)
    acc = tp/total
    print('Class -', class_name, ' acc', acc)
```

```
Class - leaves  acc 1.0
Class - brick  acc 1.0
Class - grass  acc 1.0
Class - fabric  acc 0.95
Class - wood  acc 0.95
Class - laminate  acc 0.85
Class - water  acc 0.95
Class - asphalt  acc 0.95
Class - soil  acc 0.9
Class - bark  acc 0.9
```

Рисунок 3.55 – Значення accuracy для кожного класу розширеного датасету

```
from sklearn.metrics import classification_report
report = classification_report(y_true, y_pred)
print(report)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.95	1.00	0.98	20
2	1.00	1.00	1.00	20
3	0.90	0.95	0.93	20
4	0.83	0.95	0.88	20
5	1.00	0.85	0.92	20
6	1.00	0.95	0.97	20
7	0.86	0.95	0.90	20
8	1.00	0.90	0.95	20
9	0.95	0.90	0.92	20
accuracy			0.94	200
macro avg	0.95	0.95	0.95	200
weighted avg	0.95	0.94	0.95	200

Рисунок 3.56 – Значення precision, recall, f1-score для кожного класу розширеного датасету

Для розуміння проблеми з класифікацію окремих класів була побудована матриця помилок (confusion matrix) (рис.3.57). Матриця помилок у разі багатокласової класифікації являє собою таблицю, де кожен рядок представляє справжні класи, а кожен стовпець – передбачені класи. Ця матриця дає змогу візуалізувати якість алгоритму класифікації щодо кожного класу. Для багатокласової класифікації матриця зазвичай має розміри N x N, де N – кількість класів. Діагональні елементи матриці показують кількість правильних передбачень для кожного класу. Це означає, що для комірки [i, i],

вона показує, скільки разів клас і було правильно класифіковано. Недіагональні елементи показують помилки класифікації. Значення в комірці [i, j] показує, скільки разів зразки, які насправді належать до класу i, були помилково класифіковані як такі, що належать до класу j. Використовуючи матрицю помилок, можна визначити:

– загальну точність моделі (accuracy), де кількість правильних передбачень ділиться на загальну кількість передбачень (сума всіх значень у матриці);

– повноту для кожного класу (class recall), де кількість правильних передбачень для класу ділиться на загальну кількість зразків у цьому класі (сума значень у відповідному рядку), тобто TP/(TP+FN);

– чутливість для кожного класу (class presition), де кількість правильних передбачень для класу ділиться на загальну кількість зразків, які справді належать до цього класу (сума значень у відповідному стовпчику), TP/(TP+FP).

Матриця помилок особливо корисна для ідентифікації класів, які часто плутаються, що може вказувати на необхідність подальшої оптимізації моделі або попередньої обробки даних.

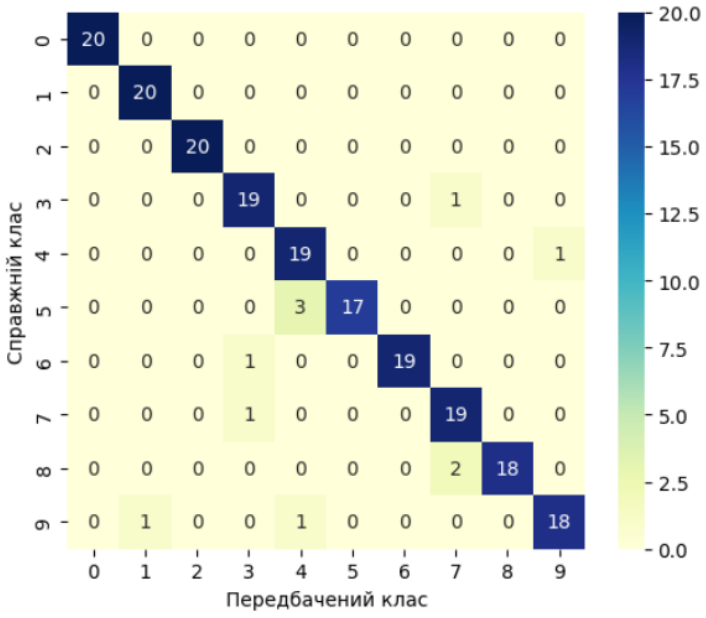


Рисунок 3.57 – Матриця помилок класифікації для розширенного датасету

Результати класифікації показані на рисунках 3.58 – 3.61, де на рисунку 3.61 присуня помилка класифікації.

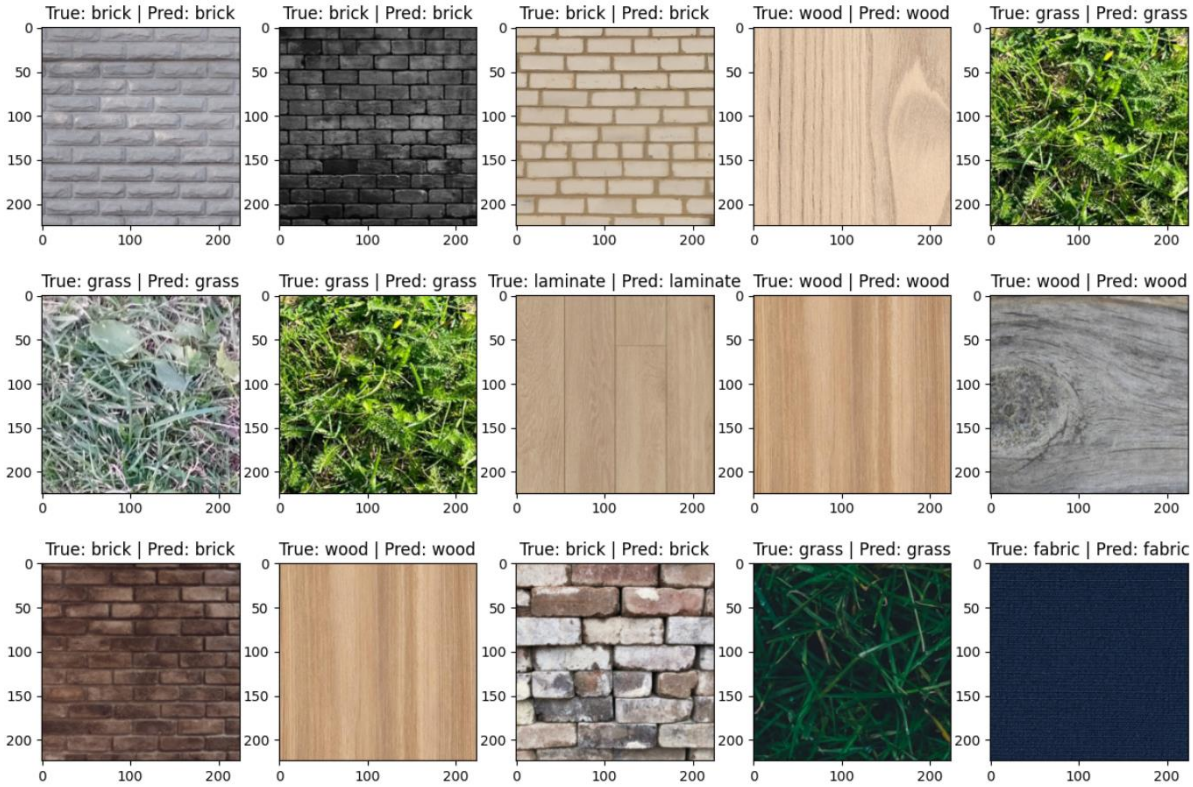


Рисунок 3.58 – Приклади класифікації розширеного датасету (без помилок)

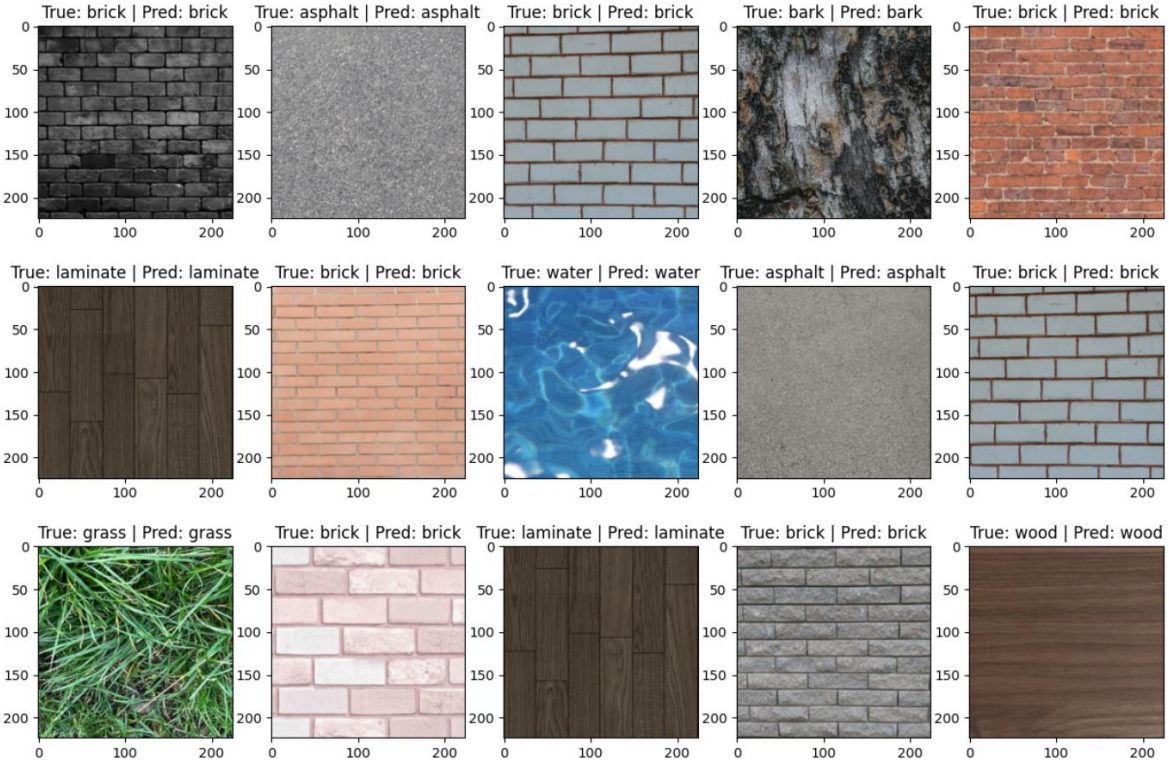


Рисунок 3.59 – Приклади класифікації розширеного датасету (без помилок)

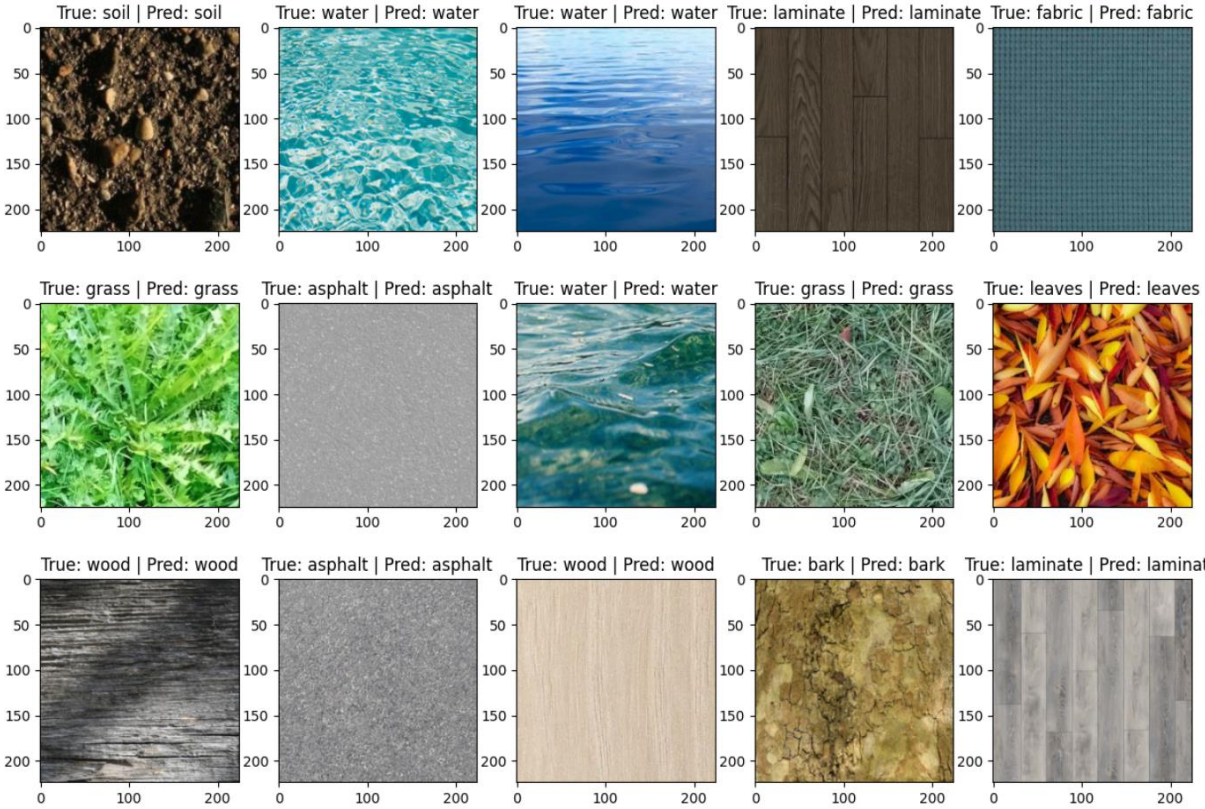


Рисунок 3.60 – Приклади класифікації розширеного датасету (без помилок)

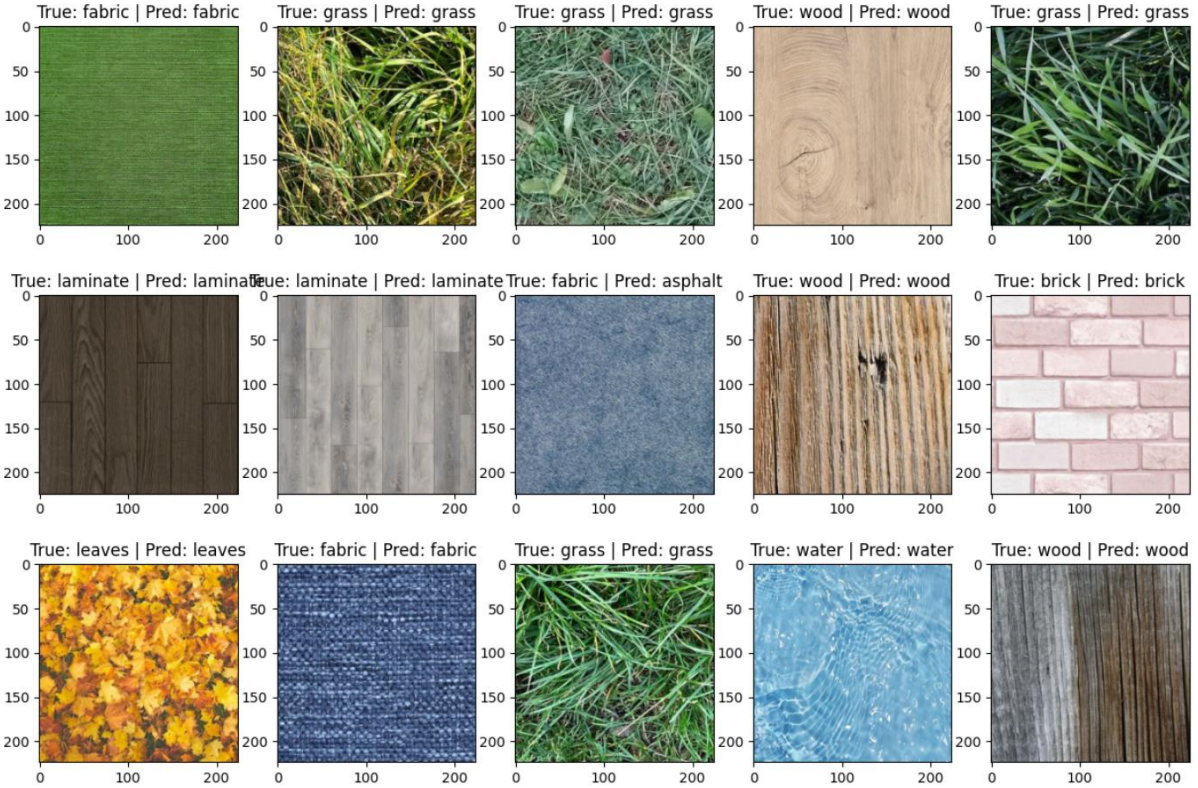


Рисунок 3.61 – Приклади класифікації розширеного датасету (присутня помилка)

Таким чином, точність класифікації розширеного датасету троху нижча, та дорівнює 0,95 (в порівнянні з 0,98 для датасету для 5 класів), але залишається досить високою для використання для багатьох реальних задач.

3.7.5 Порівняння швидкодії

Ще одним варіантом оцінки роботи алгоритму є оцінка швидкодії (рис. 3.62 , 3.63)

```
Загальний час роботи програмної моделі для ЛБШ : 5.336711406707764
```

Рисунок 3.62 – Результат виконання коду розрахунку швидкодії для методу ЛБШ

```
Загальний час роботи програмної моделі для методу, на основі матриці збігів : 28.914587259292603
```

Рисунок 3.63 – Результат виконання коду розрахунку швидкодії для методу, на основі матриці збігів

На основі аналізу ефективності, можна зробити висновок, що техніка ЛБШ демонструє більш високу швидкість обробки. Виявлена відмінність є значущою, хоча на даний момент, з огляду на обмежений розмір датасету, вона не впливає критично. Проте, з ростом обсягу електронного архіву, ефективність обробки даних стане ключовим фактором для вибору відповідного алгоритму.

Швидкість класифікації на основі донавченної моделі MobileNetV2 дорівнює близько 67 мс (рис. 3.64).

```
Час класифікації на основі моделі MobileNetV2: 67.78 ms
```

```
Час класифікації на основі моделі MobileNetV2 (для розширеного датасету): 67.43 ms
```

Рисунок 3.64 – Результат виконання коду розрахунку швидкодії для класифікації на основі донавченної моделі MobileNetV2

3.8 Висновки щодо розглянутих методів

Для аналізу алгоритмів були обрані два ключові параметри:

– якість класифікації, яка оцінюється через точність та повноту роботи алгоритму;

– швидкість обробки даних.

На основі проведеного дослідження можна зробити висновок, що обидва традиційні підходи не показали високих результатів у термінах класифікації. Максимально можливі показники точності та повноти становлять 1, але зафіксована точність для методу ЛБШ становить 0,76, а для методу на базі матриці збігів – 0,87. Щодо повноти, метод ЛБШ показав результат 0,59, тоді як метод з матрицею збігів – 0,54. Таким чином, метод ЛБШ переважає за точністю, а метод на основі матриці збігів – за повнотою. З погляду швидкості обробки даних, метод ЛБШ значно перевершує, будучи приблизно в 5 разів швидшим.

Класифікація на основі донавченої нейронної моделі MobileNetV2 дає набагато кращі результати. В випадку класифікації на 5 класів все метрики (accuracy, precision, recall, f1-score) приймали значення 0,98. В випадку класифікації на 10 класів точність за розглянутими метриками знизилась до 0,95. Що говорить про необхідність подальших експериментів, наприклад, щодо збільшення кількості зразків для кожного класу.

ВИСНОВКИ

В рамках кваліфікаційної роботи було досліджено детальне вивчення методів пошуку текстурних зображень, зокрема, за допомогою класичних методів аналізу текстури, таких як локальні бінарні шаблони та метод на базі матриці збігів. Отримані результати цього дослідження були застосовані при розробці нового алгоритму та його програмній реалізації для виявлення текстурних зображень.

Протягом дослідження були розглянуті наступні теоретичні аспекти:

– аналіз сучасного стану досліджень в області аналізу текстурних зображень;

– вивчення різних методів опису текстур, включаючи класичні та нейромережеві підходи;

– дослідження проблематики пошуку текстурних зображень;

– огляд сучасного програмного забезпечення для задач комп'ютерного зору;

– аналіз наявних датасетів зображень;

– вибір критеріїв для оцінки схожості текстур;

– розробка алгоритму пошуку текстурних зображень у цифрових колекціях;

– створення датасету текстурних зображень;

– реалізація алгоритму локальних бінарних шаблонів та методу на основі матриці збігів;

– доопрацювання нейромережі MobileNetV2;

– порівняльний аналіз результатів пошуку текстур за допомогою розглянутих методів;

– оцінка швидкодії вивчених методів;

– висновки щодо ефективності цих алгоритмів.

Експерименти з класичними алгоритмами виявили, що найшвидшим був

метод локальних бінарних шаблонів. Щодо метрик якості, результати обох методів були порівнянно ефективними (з легкою перевагою методу на основі матриці збігів за точністю та методу локальних бінарних шаблонів за повнотою), проте жоден з методів не демонстрував достатньо високу якість для задач, де висока ціна помилки може мати серйозні наслідки, наприклад, у сферах, пов'язаних з життям та здоров'ям людини. Однак, ці методи можуть бути корисними у менш критичних застосуваннях, як-от при пошуку текстур у стокових фотографіях.

Класифікація текстур на основі нейромережевого підходу, а саме на основі донавченої моделі MobileNetV2, показала значно кращу якість у порівнянні з класичними алгоритмами, а саме точність дорівнювала 0.98. Швидкодія же була в 10 разів гірша в порівнянні з методом ЛБШ, а в 3 рази гірша в порівнянні в матрицями збігів.

Таким чином, можна зробити висновок, що розглянути класичні алгоритми рекомендується використовувати в задачах, де потрібно швидко обробити великі об’єми даних, але лише в задачах не критичних, не пов’язаних з життям та безпекою. Навпаки нейроні моделі показали високу точність і можуть використатися в задачах більш критичних, де важливіша точність ніж швидкість. Однак швидкість обробки одного кадру в 67 мс достатня для обробки кадрів в реальному часі, тобто підходить для багатьох задач з реального життя.

У подальших дослідженням для підвищення якості класифікації на основі нейронних моделей необхідно збільшити датсет, додавши кількість зразків для кожного класу.

Результати досліджень були апробовані на IV міжнародній науково-технічній конференції «The world of modern technologies and inventions».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2023) Transforming image descriptions as a set of descriptors to construct classification features. Бантул, Джок'якарта, Індонезія, Indonesian Journal of Electrical Engineering and Computer Science.

2. Yanholenko, O., Cherednichenko, O., Yakovleva, O., & Arkatov, D. (2020). A Model for Estimating the Security Level of Mobile Applications: a Fuzzy Logic Approach. In IntelITSIS (pp. 252-266).

3. Cherednichenko, O., Yanholenko, O., Liutenko, I., & Iakovleva, O. (2013). Monitoring and Evaluation Problems in Higer Education-Comprehensive Assessment Framework Development. In CSEDU (pp. 455-460).

4. Гречишкін Д.С. (2023). Аналіз у порівняльному аспекті моделей MCNN та CSRNET для вирішення задачі підрахунку людей у натовпі. Матеріали XXVII Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті», Конференція «Сучасні методи обробки зображень», Т. 7. Харків: ХНУРЕ. (pp. 48-49).

5. Yakovleva, O., & Nikolaieva, K. (2020). Research Of Descriptor Based Image Normalization And Comparative Analysis Of SURF, SIFT, BRISK, ORB, KAZE, AKAZE Descriptors. Advanced Information Systems, 4(4), 89-101. doi:10.20998/2522-9052.2020.4.13.

6. Yakovleva O., Nebeský L, Liakhov P. (2023). Research methods of texture image analysis to solve the texture search problem. Proceedings of the IV International Scientific and Practical Conference. Vienna, Austria. 2023. pp. 252-261.

7. Bahmanyar, R., Vig, E., & Reinartz, P. (2019, September 27). MRCNet: Crowd Counting and Density Map Estimation in Aerial and Ground Imagery. URL: arXiv.org. https://arxiv.org/abs/1909.12743. (дата звернення: 5.02.2018).

8. CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes. IEEE Xplore. (n.d.). URL: https://ieeexplore.ieee.org/document/8578218. (дата звернення: 30.09.2023).

9. А.Р. Ковтуненко, О.В. Яковлева, В.А. Любченко, & О.В. Янголенко (2020) Дослідження сумісного використання математичної морфології та згорткових нейронних мереж для вирішення задачі розпізнавання цінників. Вісник Національного технічного університету ХПІ (3). 24–31.

10. Yakovleva O., Nebeský L., Kirichenko A. (2023). Using the GPT models for responses based on custom content to develop neural consultant for university applicants. Abstracts of V International Scientific and Practical Conference. Madrid, Spain. Pp. 172-178.

11. Welcome to Python.org. Python.org. (n.d.). URL: https://www.python.org/about/ (дата звернення: 21.05.2011).

12. Project Jupyter. (n.d.). URL: https://jupyter.org/ (дата звернення: 5.01.2021).

13. Yakovleva, O., Kovtunenko, A., Liubchenko, V., Honcharenko, V., & Kobylin, O. (2023). Face Detection for Video Surveillance-based Security System. CEUR Workshop Proceedings Vol. 3403. pp. 69-86. ISSN 1613-0073.

14. dlib C++ Library. (n.d.).URL: http://dlib.net/face_landmark_detection.py (дата звернення: 18.10.2023).

15. Datasets & Dataloaders – PyTorch Tutorials 1.8.1+cu102 documentation. URL:https://pytorch.org/tutorials/beginner/basics/dat_tutorial.html (дата звернення: 12.04.2019).

16. TensorFlow Datasets. TensorFlow. (n.d.). URL: https://www.tensorflow.org/datasets (дата звернення: 6.10.2023).

17. Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2017, August 23). A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. URL: arXiv.org. https://arxiv.org/abs/1707.08819 (дата звернення: 24.11.2023).

18. Common Objects in Context. COCO. (n.d.). URL: https://cocodataset.org/#home (дата звернення: 30.10.2023).

19. CC0 Textures – Free Public Domain PBR Materials. (n.d.). URL: https://cc0textures.com/ (дата звернення: 7.10.2023).

20. Neeraj, Rosebrock, A., Arun, Charles, Michael, Sveder, Abu, Suyuancheng, Abu, Dharmendra, Pradeep, Rui, Z., Wanderson, Danieal, Mahmod, Aquib, G., P., Bhat, S., Nisha, … Fast. (2021, April 17). Local Binary Patterns with Python & OpenCV. PyImageSearch. URL: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/ (дата звернення: 17.10.2023).

21. Яковлева, Е. В., & Кускова, И. В. (2006). Исследование результатов сегментации изображений методом матриц совпадений. Вісник Національного технічного університету «ХПІ». №39 - С.164 -171.

22. Klyuchka, Y. A., Cherednichenko, O. Y., Vasylenko, A. V., & Yakovleva, O. V. (2017). Forecasting the results of football matches on the internet based information. Bulletin of National Technical University “KhPI”. Series: System Analysis, Control and Information Technologies, 0(55), 51–59. https://doi.org/10.20998/2079-0023.2017.55.09.

23. Яковлева, Е. В., & Панченко, И. А. (2007). Применение энергетических характеристик Лавса для сегментации изображений. Бионика интеллекта : научно-технический журнал. №2(67). – С.94-98.

24. Yakovleva, O., Kovač, M., Ardasov, V. & Yeremenko, I. (2023). Study on adding functionality to the Zoom online conference system for monitoring the participant activities. Public Administration and Regional Development, 19(1), pp. 158–184.

25. Яковлєва, О. В., & Нестерова, Є. П. (2009). Порівняльний аналіз методів характеристик Лавса та матриць збігів у завданнях сегментації текстурних зображень

26. Cherednichenko, O., Kanishcheva, O., Yakovleva, O., & Arkatov, D. (2020). Collection and Processing of a Medical Corpus in Ukrainian. corpus, 2(4), 7-14.

27. Cherednichenko, O., Yanholenko, O., Iakovleva, O., & Kustov, O. (2014). Models of Research Activity Measurement: Web-based monitoring implementation. Information Systems: Education, Applications, Research, 75–87. https://doi.org/10.1007/978-3-319-11373-9_7.

28. Cherednichenko, O., Vovk, M., Yanholenko, O., & Yakovleva, O. (2020). Towards the Technology of Employers' Requirements Collection Development. In Integrated Computer Technologies in Mechanical Engineering (pp. 228-239). Springer, Cham.

29. Cherednichenko, O., Vovk, M., Yanholenko, O., & Yakovleva, O. (2020). Towards the Technology of Employers' Requirements Collection Development. In Integrated Computer Technologies in Mechanical Engineering (pp. 228-239). Springer, Cham.

30. Іщенко О.І. (2023). Розробка методу аналізу графічного контенту в електронній колекції текстових документів для вирішення задачі пошуку плагіатних зображень. Матеріали XXVII Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті», Конференція «Сучасні методи обробки зображень», Т. 7. Харків: ХНУРЕ. (pp. 56-67).