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)

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ПОШУКУ ПЛАГІАТУ ЗОБРАЖЕНЬ З ПОДАЛЬШИМ ЙОГО ВПРОВАДЖЕННЯМ В СИСТЕМУ ВИЯВЛЕННЯ ПЛАГІАТУ ЗОБРАЖЕНЬ В ТЕКСТОВИХ ФАЙЛАХ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-20-1

Пилипенко П.В.
(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)

2021 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)
«____»_____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пилипенку Павлу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та дослідження методу пошуку плагіату зображень з подальшим його впровадженням в систему виявлення плагіату зображень в текстових файлах

затверджена наказом по університету від «_22_» жовтня 2021 року № 1574Ст.

2. Термін подання студентом роботи до екзаменаційної комісії _22 _листопада____ 2021 р.

3. Вихідні дані до роботи Методи визначення змісту зображення, методи виділення характерних точок, алгоритми детекторів та дескрипторів, алгоритм RANSAC, нейромережа ResNet-50, мова програмування Python, вебфреймворк Django, бібліотека OpenCV.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд сучасних методів виявлення плагіату зображень.
2. Реалізація алгоритму знаходження змісту зображення.
3. Реалізація алгоритму пошуку характерних точок зображення.
4. Створення алгоритму зручного збереження знайденої інформації у базі даних.
5. Розробка алгоритму пошуку подібних зображень серед збережених у базі даних.
6. Створення правил оцінки подібності зображень для прийняття рішення про плагіат.

7. Розробка вебзастосунку з можливістю виконувати перевірку зображень на плагіат.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність дослідження, постановка задачі, опис предметною області, виявлення змісту, виділення характерних точок, пошук відповідних пар двох зображень, відбір кращих пар, знаходження параметрів геометричних перетворень, розрахунок відсотку подібності, ілюстрація роботи застосунку.

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
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	22.10.2021	
2	Аналіз завдання, підбір літератури	22.10.21-23.10.21	
3	Аналіз літератури з досліджуваної проблеми	23.10.21-24.10.21	
4	Аналіз технічних засобів	25.10.21-26.10.21	
5	Розробка методу	27.10.21-01.11.21	
6	Програмна реалізація	01.11.21-08.11.21	
7	Оформлення пояснювальної записки	08.11.21-15.11.21	
8	Перевірка на плагіат	20.11.2021	
9	Рецензування	28.11.2021	
10	Підготовка презентації та доповіді	29.11.2021	
11	Занесення роботи в електронний архів	01.12.2021	
12	Попередній захист кваліфікаційної роботи	01.12.2021	

Дата видачі завдання 22 жовтня 2021 р.

Студент _____
(підпис)

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

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 77 с., 3 табл., 29 рис., 3 дод., 37 джерел.

КОМП'ЮТЕРНИЙ ЗІР, ВИЯВЛЕННЯ ПЛАГІАТУ, ПОДІБНІСТЬ ЗОБРАЖЕНЬ, ДЕТЕКТОР, ДЕСКРИПТОР, НЕЙРОМЕРЕЖА, RANSAC, ВЕБЗАСТОСУНОК, PYTHON, DJANGO, OPENCV.

У роботі розробляється та пропонується алгоритм виявлення плагіату зображень у текстових документах, який впроваджується у вебзастосунок. Порівняння нових зображень відбувається із зображеннями, інформація о яких збережена в базі даних. Для виявлення змісту зображення використовується нейромережа ResNet-50, а для інформативного опису характерних точок обирається дескриптор AKAZE. Розроблено алгоритм прийняття рішення про визначення зображення плагіатом. Зображення можуть відрізнятись від оригіналу геометричними перетвореннями, особлива увага приділена випадкам зміни розмірів та відсікання частини зображення. Для знаходження пар відповідних характерних точок на зображеннях, використовується метод *k*-найближчих сусідів, а хибні пари відкидаються за допомогою алгоритму RANSAC.

Розроблений алгоритм впроваджено у вебзастосунок з базою даних, організованою для швидкого пошуку відповідних зображень. Застосунок розроблений за допомогою мови програмування Python та вебфреймворку Django.

COMPUTER VISION, PLAGIARISM DETECTION, SIMILARITY SAVED, DETECTOR, DESCRIPTOR, NEURAL NETWORKS, RANSAC, WEB APPLICATION, PYTHON, DJANGO, OPENCV.

The paper develops and proposes a plagiarism detection algorithm for images stored in a text document that is embedded in a web application.
New images are compared with the images, information about which is stored in the database. To identify the content of the image, the ResNet-50 neural network is used, and the AKAZE descriptor is selected for the informative description of the characteristic points. An algorithm for making a decision on determining an image as a plagiarism has been developed. Images may differ from the original by geometric transformations, special attention is paid to cases of resizing and clipping part of the image. To find pairs of corresponding feature points in images, the *k*-nearest neighbors method is used, and false pairs are rejected using the RANSAC algorithm.

The developed algorithm is embedded in a web application with a database organized to quickly search for relevant images. The application is developed using the Python programming language and the Django web framework.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Сучасний стан проблеми виявлення плагіату 9
1.1 Проблематика пошуку зображень, що є підозрілими на плагіат 9
1.2 Огляд існуючих сервісів для пошуку плагіату зображень та виявлення плагіату зображень в текстових файлах 10
1.3 Методи порівняння зображень за змістом 11
1.3.1 Класичні методи пошуку схожих зображень 11
1.3.2 Нейромережеві методи 13
1.4 Огляд програмних можливостей для вирішення задачі пошуку зображень, що є підозрілими на плагіат 15
1.5 Постановка задачі дослідження 18
2 Розробка та дослідження методу пошуку зображень, що є підозрілими на плагіат 20
2.1 Загальний опис методу для визначення зображень, схожих на плагіат 20
2.2 Підготовка зображення перед порівнянням 21
2.2.1 Класифікація зображення за змістом на базі нейромережевого підходу 21
2.2.2 Математична модель опису зображень на основі характерних точок 23
2.3 Порівняння зображень 26
2.3.1 Відбір зображень схожих за змістом на базі нейромережевого підходу 26
2.3.2 Відбір зображень на основі аналізу характерних точок 27
2.3.3 Розробка критерію про підозру на плагіат на основі методу RANSAC 32

2.4 Алгоритм пошуку зображень, що є підозрілими на плагіат, у текстових файлах 33
2.5 Дослідження ефективності обраних критеріїв та налаштування параметрів 35
3 Розробка комп'ютерної моделі 42
3.1 Архітектура застосунку 42
3.1.1 Використання вебфреймворку для створення головних модулів системи 42
3.1.2 Проєктування бази даних 45
3.2 Програмна реалізація 46
3.3 Тестування розробленої системи 53
3.4 Аналіз результатів роботи застосунку по запропонованому алгоритму 56
3.4.1 Створення дата сету 56
3.4.2 Опис експериментів 57
3.4.3 Результати експериментів 58
3.4.4 Висновки щодо досліджень 67
Висновки 70
Перелік джерел посилання 71
Додаток А Порівняння кількості знайдених відповідностей до кількості характерних точок 75
Додаток Б Порівнянь кількості знайдених відповідностей після RANSAC та до нього 76
Додаток В Приклади зображень у колекції 77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

СУБД – система управління базами даних

API – Application Programming Interface

BSD – Berkeley Software Distribution

FED – Fast Explicit Diffusion

ORM – Object Relation Mapping

OpenCV – Open Source Computer Vision Library

MVC – Model View Controller

FED – Fast Explicit Diffusion

MLDB – Modified Local Difference Binary

RANSAC – Random Sample Consensus

ВСТУП

У часи швидкого розвитку інформаційних технологій, все більше повсякденних справ переходить у цифровий формат. Тим паче у часи складних епідеміологічних умов все більше й більше послуг переходить у online-формат. Це має свої переваги та недоліки, але сьогодні буде звернена увага на проблему плагіату. Це явище, коли людина бере чужу роботу і видає її за свою, порушуючи авторське право.

Плагіат може зустрічатись у майже всіх сферах людської життєдіяльності. Це може бути як привласнення чужого тексту, так і зображень, креслень, схем, результатів роботи програм. Для перевірки тексту на плагіат вже існує велика кількість сервісів, які навіть можуть залишити посилання на оригінальний текст. Але перевірка зображень це більш складна робота, тим паче що зображення можна змінювати за розміром, відсікати частини. Людина може з легкістю побачити подібність зображень, але для автоматичної швидкої обробки великої кількості зображень необхідно розробити алгоритм порівняння зображень і прийняття рішення про плагіат.

Ця робота присвячена розробці та дослідженню методу пошуку плагіату зображень з подальшим його впровадженням в систему виявлення плагіату зображень в текстових файлах.

1 СУЧАСНИЙ СТАН ПРОБЛЕМИ ВИЯВЛЕННЯ ПЛАГІАТУ

1.1 Проблематика пошуку зображень, що є підозрілими на плагіат

Зображення, з точки зору обчислювальної машини, це всього лише великий набір пікселів. Кожен піксель це лише точка якогось кольору. Найпростіший спосіб порівняти два зображення – це попіксельне порівняння двох зображень. Якщо зображення однакові, то такий метод спрацює. Однак одне і те ж саме зображення може відрізнятися незначними змінами кольорів, контрастністю, розмірами і навіть бути з обрізаними краями. Людське око з легкістю помітить схожість двох зображень в цих випадках, однак для машини це буде абсолютно різні зображення. Для того, щоб ефективно порівнювати зображення вже існує безліч алгоритмів [1], однак все одно виникають випадки, коли зображення можуть виглядати абсолютно по-різному, але обрані характеристики будуть збігатися.

Не варто забувати і про швидкість роботи порівняння зображень. Чим більший розмір картинки, тим більше буде потрібно часу на його обробку. З цієї причини, неможливо провести порівняння однієї картинки з усіма відомими в світі, а значить потрібно розробити критерії – які зображення можуть бути подібними, і на основі чого варто їх порівнювати, щоб пошук був максимально точним і швидким.

Зображення також можуть відрізнятися форматом даних, ступенем стиснення і різними геометричними перетвореннями. А якщо ж зображення знаходяться в текстових файлах, то перед виконанням пошуку потрібно їх витягти.

Виходячи з усього описаного вище, можемо зробити висновок, що порівнювати зображення досить важко, і результат залежить від великої кількості змінних.

1.2 Огляд існуючих сервісів для пошуку плагіату зображень та виявлення плагіату зображень в текстових файлах

Сьогодні системи пошуку зображень як ніколи розвинені. Незалежно від того, яке зображення необхідно знайти, швидше за все, за допомогою правильних ключових слів, фільтрів пошуку та інструментів воно буде знайдене. Існують сервіси зворотного пошуку, які допоможуть знайти джерело для зображення. Для пошуку по зображенням існує велика кількість сервісів [2].

TinEye – це система зворотного пошуку зображень, яка допомагає шукати зображення та знаходити, де вони з'являються в Інтернеті. Цей інструмент дозволяє шукати як за URL-адресами, так і за завантаженими зображеннями [3].

Пошук зображень Google – це, мабуть, найпростіший і найпопулярніший інструмент пошуку зображень – і є основним ресурсом для багатьох маркетологів. Це допоможе вам знайти точне зображення, перевірити авторські права, а також розмір зображення. Пошук зображень Google надзвичайно простий і абсолютно безкоштовний у використанні.

Інструмент візуального пошуку Bing був вперше представлений Microsoft у березні 2014 року. Спочатку він використовувався як система зворотного пошуку зображень. Bing Visual Search – ще один дуже простий і зручний у використанні інструмент пошуку зображень, сумісний з будь-яким браузером. Це чудовий інструмент для перевірки того, чи використовують ваші конкуренти якісь ваші зображення чи подібні зображення. Завантаживши зображення або вказав URL-адресу, з'являться результати пошуку та інформація: на яких вебсторінках присутнє зображення, подібні зображення, пов'язані пошукові запити.

Існують також сервіси, які призначені для пошуку саме плагіату зображення.

Plaghunter. Сервіс, який робить пошук зображень через сервіси Google самостійно, та повертає усі посилання на це зображення [4]. Цей сервіс також має платні послуги, які дозволяють постійно відстежувати використання ваших зображень, робить автоматичні звіти, та навіть відстежує хто використовує ваші фотографії у соціальних мережах.

Duplichecker. Цей сервіс також виконує зворотній пошук зображень та надає інформацію про зображення, що складається з об’єктів, місць, людей і т.д. Таким чином, не вийде випадково використати зображення, що юридично належить іншому сайту, блогу чи каналу. Для використання потрібно лише завантажити зображення або розмістити посилання на нього [5]. Щоб отримати широкий спектр зображень HD-якості, використовується пошук Google.

Всі існуючі сервіси шукають інформацію в мережі Інтернет по одному зображенню, та здебільшого використовують можливості пошукових сервісів, таких як Google.

1.3 Методи порівняння зображень за змістом

1.3.1 Класичні методи пошуку схожих зображень

Класичні методи пошуку використовують чіткі алгоритми аналізу змісту зображення. Це може бути такі ознаки, як текстура, форма, колір об’єктів, що представлені на зображенні. Вони описуються з використанням чітких правила розрахунку певних характеристик. Не існує єдиного чітко визначеного способу отримання рішення про подібність двох зображень. Через це виділяється якнайменше три методи пошуку [6].

Перший метод, це метод пошуку по текстурним подібностям. Він опирається на те, що зображення можуть мати однаковий просторовий розподіл кольорів, якщо вони близькі за характеристиками текстурної подібності. Крім того, значення яскравості можуть бути однакові, але

відтінки значення двох зображень можуть не збігатися [7]. Головним завданням є виділення на зображенні певних частин з однорідною структурою, але з різними характеристиками. Це можна представити як карту місцевості, на якій позначені ріка, ліс, будинок. У випадку аналізу зображень, кажучи про ці частини, мається на увазі сукупність характеристик, що описують візуальні властивості будь-яких об’єктів або поверхонь, тобто текстура зображення.

Під час розпізнавання текстур виконуються процеси сегментації і класифікації. На етапі сегментації виділяються геометричні області і задаються відповідні їм порядкові номери. На етапі класифікації здійснюється зіставлення пікселів текстур. Кількість ознак, за якими можуть відрізнятися один від одного ділянки різних типів, дуже велике. У деяких завданнях це середня яскравість, дисперсія, центрований момент, ентропія, в інших – кореляційні властивості [8].

Другим методом є метод пошуку за подібністю кольорів. Для цього будується гістограма розподілу колірних складових. Подібність зображень визначається шляхом порівняння цих гістограм двох зображень. Якщо вони однакові, або дуже схожі, то зображення будуть вважатись подібними. Звичайно бувають випадки, коли зображення схожі за кольорами, але мають суттєво інший зміст. Це є одним з головних з недоліків даного методу.

Третій метод це метод пошуку за подібністю форм. Він використовує опис геометричної форми регіонів зображення. Це відбувається шляхом виділення кордонів, тобто сегментації регіонів. Саме цей метод найчастіше використовується у завданнях автоматичного стеження за рухомими об’єктами на послідовності зображень. При спостереженні за об’єктом присутні різні негативні впливи, наприклад, перепади освітлення, шуму на послідовності зображень та інші ефекти. Також об’єкт спостереження може бути частково закритий іншим об’єктом, що також може сприяти зриву спостереження.

1.3.2 Нейромережеві методи

Нейромережеві методи, на відміну від класичних, базуються на порівнянні змісту зображень з використанням нейромереж. У таких підходах є встановлені кроки [9].

Постановка завдання і вибір архітектури нейронної мережі. На даному етапі необхідно отримати чітке розуміння про розв’язувані задачі, причини використання для цього методу нейромережевого моделювання та переваги, які він повинен дати. Під конкретну постановку задачі підбирається архітектура нейронної мережі.

Визначення кількісного і якісного складу входів і виходів. Якісний склад вхідних і вихідних змінних нейромережевої моделі визначається, головним чином, поставленими на попередньому етапі завданнями. Кількісний склад може залежати ще від природи вхідних і вихідних змінних і описуваних ними процесів і властивостей. Так, при необхідності масштабування деяких змінних число фактичних входів і виходів нейронної мережі може збільшитися в порівнянні з кількістю входів і виходів моделі.

Формування вихідної вибірки даних. Вихідна вибірка даних може бути сформована на основі практичного або обчислювального експерименту. Чим більше обсяг проведеного експерименту, тим точнішу нейромережеву модель можна отримати і використовувати на практиці. Слід також враховувати і вимоги до вибірки вихідних даних, сформульовані в попередньому розділі.

Попередня обробка і нормалізація вихідної вибірки. Даний етап визначається обраною архітектурою нейронної мережі, складом вхідних і вихідних змінних і, власне, фактичними даними, отриманими в результаті експерименту.

Поділ вихідної вибірки на навчальну і тестову складові. Наявна вихідна вибірка ділиться на навчальну і тестову підвибірки з урахуванням використовуваної архітектури нейронної мережі. Зазвичай обсяг навчальної підвибірки в кілька разів більше, ніж тестової. Перша використовується

строго для настройки вагових коефіцієнтів. Друга – для перевірки коректності налаштованої моделі. У разі обмеженого обсягу вихідної вибірки вона вся може бути задіяна для навчання, а оцінка коректності роботи мережі перевіряється на етапі практичної експлуатації.

Визначення структури нейронної мережі. Крім раніше встановленого складу вхідних і вихідних змінних, на даному етапі задається кількість прихованих шарів і нейронів у кожному шарі. Для багатьох архітектур нейронних мереж вибір структури залежить, в тому числі, від обсягу навчальних даних.

Налаштування параметрів нейронної мережі та алгоритму її навчання. На даному етапі задаються вид і параметри активаційних функцій прихованих і вихідних нейронів. Для ітераційного алгоритму вибираються умови закінчення навчання, коефіцієнт швидкості, порядок пред'явлення прикладів навчальної вибірки.

Навчання нейронної мережі. Для точних алгоритмів зазвичай здійснюється одноразовий розрахунок вагових коефіцієнтів. Для ітераційних – багаторазове повторення епох навчання. Для навчання можуть бути використані тільки приклади навчальної підвибірки.

Контрастування нейронної мережі. Якщо алгоритм навчання нейронної мережі має ітераційне підстроювання вагових коефіцієнтів на протязі всіх епох навчання, нерідкі випадки, коли деякі з ваг за абсолютним значенням настільки близькі до нуля, що не роблять ніякого практичного впливу на результати розрахунків. Проте, на такі розрахунки витрачається досить багато обчислювальних ресурсів, що помітно уповільнює процес навчання. Для вирішення цієї проблеми такі вагові коефіцієнти повністю обнуляються, тобто відповідний зв'язок між входами і нейронами або між двома нейронами виключається зі структури мережі.

Тестування нейронної мережі. Тестування нейронної мережі проводиться з використанням прикладів тестової вибірки. За результатами тестування оцінюється помилка роботи нейромережевої моделі. Можуть

виконуватися в циклі неодноразово доти, поки не буде отримана налаштована нейронна мережа, за допомогою якої можна вирішувати завдання з необхідним рівнем помилки.

Практичне використання. Навчена і протестована нейронна мережа використовується для практичного вирішення завдання по мірі надходження нових вихідних даних (значень вхідних змінних). Результати по навченої мережі можуть бути отримані практично миттєво і інтерпретовані користувачем для подальшого прийняття рішень.

Донавчання нейронної мережі. Отримані в ході практичної експлуатації нейронної мережі пари вхідних і вихідних векторів можуть використовуватися для подальшої підстроювання вагових коефіцієнтів. Це особливо важливо, якщо спочатку обсяг вибірки даних був невеликим. Частина архітектур нейронних мереж використовують принцип самоорганізації – в ході донавчання можуть не тільки змінювати значення вагових коефіцієнтів, а й змінювати свою структуру.

1.4 Огляд програмних можливостей для вирішення задачі пошуку зображень, що є підозрілими на плагіат

Для створення застосунку, спочатку треба виділити головні риси, які йому мають бути притаманні. Перш за все, він повинен бути написаний на мові програмування, яка підтримує велику кількість бібліотек для роботи із зображеннями, та легко працює з нейромережами. Наступний критерій, це наявність потужного вебфреймворку, який може підтримувати паралельну роботу з великою кількістю одночасних користувачів, та з легкістю робити складні запити до бази даних. Через це, обирається мова програмування Python, вебфреймворк Django, та бібліотеки роботи із зображеннями OpenCV, та ImageAI.

Python – це скриптова мова, що активно розвивається, яку використовують для вирішення великого обсягу найрізноманітніших проблем і завдань. Python стане в нагоді у створенні комп'ютерних та мобільних додатків, його застосовують у роботі з великим обсягом інформації, при розробці web-сайтів та інших різноманітних проєктів, використовують у машинному навчанні. Мова є повністю об'єктно-орієнтованою – все є об'єктами [10].

Python повністю динамічно-типізований, що дозволяє створювати гнучкі функції, а те, що він є інтерпретованою мовою програмування надає можливість створювати швидкі функції і одразу їх тестувати, без необхідності збирати увесь проєкт. Головний акцент зроблений на легкості розуміння та читання коду, щоб навіть люди які не писали цей код змогли прочитати програму як книжку, без великої купи супровідної документації. За замовчування Python містить бібліотеки, які допоможуть працювати з файлами, датами, великими числами та точними обчисленнями. Крім того, існує велика кількість сторонніх безкоштовних для використання бібліотек, які дозволяють не писати широко відомі рішення самостійно, а лише встановити залежність та визивати необхідні функції зі свого коду. Це стосується як вузькоспеціалізованих рішень, так і вебфреймворків. Крім того, саме Python широко використовується у машинному навчанні і містить велику кількість зручних інтерфейсів і бібліотек як для обробки зображень, так і для навчання нейромереж.

Django – це один з безкоштовниї вебфреймворків з відкритим кодом, написаний на мові програмування Python. Цей фреймворк за замовчуванням містить велику кількість готових рішень для автентифікації користувачів, панелі адміністратора. Одною з головних відмінних рис Django є широка підтримка різноманітних баз даних, і при необхідності може використовувати декілька різних СУБД одночасно. Фреймворк складається з компонентів, які можна підключати при необхідності та повторно

використовувати. Тобто, досягати одного й того самого результату з використанням меншої кількості коду [11].

Django надає ORM для швидкого доступу до бази даних, підтримки сесії та написання запитів на мові програмування Python, які потім перетворюється на запити мови SQL.

OpenCV (Open Source Computer Vision Library) – це бібліотека функцій програмування, головним чином спрямованих на комп'ютерний зір у режимі реального часу [12]. Бібліотека є безкоштовна та має відкритий код. Бібліотека пропонує загальну структуру для програм комп'ютерного зору. Крім того, OpenCV працює за ліцензією BSD, яка накладає мінімальні обмеження використання бібліотеки. Таким чином, комерційні організації можуть налаштувати код відповідно до своїх цілей. OpenCV пропонує доступ до понад 2500 алгоритмів, які використовуються для розгортання різних можливостей машинного навчання та комп'ютерного зору, таких як ідентифікація об'єктів та розпізнавання обличчя.

ImageAI – найсучасніший штучний інтелект для розпізнавання та виявлення з кількома рядками коду [13]. ImageAI – це проста у використанні бібліотека Computer Vision Python, яка дає розробникам можливість легко інтегрувати найсучасніші функції штучного інтелекту в свої нові та існуючі програми та системи. Його використовують тисячі розробників, студентів, дослідників, викладачів та експертів у корпоративних організаціях по всьому світу. ImageAI надає API для розпізнавання 1000 різних об'єктів на зображенні за допомогою попередньо навчених моделей, які були навчені на наборі даних ImageNet-1000. Надані моделі реалізацій SqueezeNet, ResNet, InceptionV3 і DenseNet.

1.5 Постановка задачі дослідження

Проаналізувавши вищесказане, можна зробити висновки, що у нас час все більш гостро стає питання захисту інтелектуальної власності та питання плагіату. З’являється все більше потужних сервісів для пошуку по зображенню та знаходження подібних. Однак, більшість сервісів використовує лише нейромережевий спосіб пошуку зображень, та одночасно працюють лише з одним зображенням. Тобто, якщо необхідно перевірити на плагіат текстовий документ з великої кількістю зображень у ньому, то вам потрібно спочатку дістати всі зображення, а потом по одному завантажити у системи пошуку, а потім самостійно прийняти рішення на основі результату пошуку: чи це плагіат, чи ні.

Об’єктом дослідження роботи є система захисту авторського права та інтелектуальної власності, автоматизація виявлення плагіату зображень.

Предметом дослідження даної роботи є комплекс практичних та теоретичних проблем побудови інтелектуальної системи прийняття рішень в задачах розпізнавання зображень.

Метою даної роботи є створення алгоритму прийняття рішення про плагіат серед зображень у текстових файлах, та його впровадження у вебзастосунок.

Саме тому було поставлене завдання розробити та дослідити метод пошуку плагіату зображень у текстових файлах, та створити вебзастосунок, який використовуючи розроблений метод буде виконувати цей пошук та зробити висновок: чи виявлено плагіат у текстовому файлі, чи ні.

Для цього необхідно вирішити такі завдання:

- оглянути сучасні методів виявлення плагіату зображень;
- реалізувати алгоритм знаходження змісту зображення;
- реалізувати алгоритм пошуку характерних точок зображення;
- створити алгоритм зручного збереження знайденої інформації у базі даних;

– розробити алгоритм пошуку подібних зображень серед збережених у базі даних;

– створити правила оцінки подібності зображень для прийняття рішення про плагіат;

– розробити вебзастосунок з можливістю виконувати перевірку зображень на плагіат.

2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ ПОШУКУ ЗОБРАЖЕНЬ, ЩО Є ПІДОЗРІЛИМИ НА ПЛАГІАТ

2.1 Загальний опис методу для визначення зображень, схожих на плагіат

Для створення повного алгоритму визначення плагіату зображень, спочатку треба пройти велику кількість кроків. Перш за все, треба мати на увазі, що зображення знаходяться у текстових файлах різних форматів, та самі зображення можуть бути у різних форматах та різних розмірів.

Для того, щоб виконувати перевірки, важливо сформувати колекцію зображень, з якими буде вестись перевірка. Щоб зробити перевірки швидшими, перед додаванням нового зображення до колекції треба також додати інформацію про зображення, набір характеристик, які будуть використовуватись при порівняння.

На етапі поповнення колекції виникає проблема, що зображень стає дуже багато, і виконувати перевірку нового зображення із кожним у колекції потребує дедалі більше часу. Через це до інформації про зображення необхідно додати легкий набір первинних характеристик, по яким можна робити швидкі запити. Найшвидший та найнадійніший засіб це відбір зображень схожих за змістом. Якщо на зображенні, що надходить на перевірку, зображено, наприклад, кота у смугу, то немає необхідності виконувати перевірку з усіма зображеннями, на яких відсутні коти у смуги. Для пошуку змісту зображення може допомоги нейромережевий підхід.

Відібравши схожі за змістом зображення, залишається встановити, чи дійсно це одне й те саме зображення, чи вони лише подібні за змістом. Для цього обирається метод аналізу характерних точок зображення [14]. Головна ідея полягає у тому, щоб знайти точки перепаду контрастності та описати векторні характеристики цих точок.

Маючи набір характерних точок одного зображення, та набір точок іншого зображення, їх необхідно порівняти. Для кожної точки одного зображення необхідно віднайти відповідну точку на іншому зображенні, а потім розробити критерії відкидання хибних пар. Після цього залишається лише проаналізувати результат та прийняти рішення: чи є це зображення плагіатом, чи ні.

2.2 Підготовка зображення перед порівнянням

2.2.1 Класифікація зображення за змістом на базі нейромережевого підходу

Перш за все, треба створити теги змісту зображення, щоб далі порівнювати тільки із зображеннями схожими за змістом. Для цього використовується нейромережевий підхід, а саме модель ResNet50.

ResNet50 – це варіант моделі ResNet, яка має 48 шарів згортки разом із 1 шаром MaxPool і 1 Average Pool. Він має $3{,}8\times10^{9}$ операцій з плаваючою комою [15]. Завдяки структурі, представленої ResNets, стало можливим навчання надглибоких нейронних мереж, і це означає, що мережа і може містити сотні або тисячі шарів і при цьому досягати чудової продуктивності.

Більшість проблем перепадів точності було додатково вирішено шляхом взяття більш дрібної моделі та глибокої моделі, яка була побудована з шарами дрібної моделі, і додавання до неї шарів ідентичності, і, відповідно, більш глибока модель не мала б спричиняти більшу помилку навчання, ніж її аналог, оскільки додані шари були лише шарами ідентичності.

Отже, архітектура ResNet50 містить наступний елемент:

Згортка із розміром ядра 7×7 та 64 різних ядер з кроком розміру 2, що дає 1 шар.

Максимальний пул з також розміром кроку 2.

У наступній згортці є ядро 1×1,64, а потім ядро 3×3,64 і, нарешті, ядро 1×1,256. Ці три шари повторюються загалом 3 рази, що дає 9 шарів на цьому кроці.

Далі ядро 1×1,128, потім ядро 3×3,128 і, нарешті, ядро 1×1,512, цей крок повторюється 4 рази, даючи 12 шарів на цьому кроці.

Після цього є ядро 1×1,256 і ще два ядра з 3×3,256 і 1×1,1024, і це повторюється 6 разів, що дає 18 шарів.

А потім знову ядро 1×1512 з ще двома 3×3,512 і 1×1,2048, і це повторилося 3 рази, що дало 9 шарів.

Після цього створюємо середній пул і закінчуємо його повністю з'єднаним шаром, що містить 1000 вузлів, і в кінці функцією softmax, так що це дає 1 шар.

Таким чином, у підсумку це дає 1 + 9 + 12 + 18 + 9 + 1 = 50 шарів глибокої згорткової мережі.

Цю архітектуру, та нейромережовий підхід у цілому, можна використовувати для завдань комп'ютерного зору, таких як класифікація зображень, локалізація об'єктів, виявлення різноманітних об'єктів [16].

Ця модель дуже зручна для нашої задачі виявлення змісту зображення. Вона дуже швидка, та одні й ті ж самі зображення будуть отримувати одні й ті ж самі теги, що дуже допоможе при первинному відборі подібних зображень.

Через це, кожне зображення перед перевіркою опрацьовується вже натренованою нейромережою. Результатами цього кроку є отримання п'яти тегів, які описують, що саме присутнє на зображенні. Теги відсортовані по ймовірності наявності.

2.2.2 Математична модель опису зображень на основі характерних точок

На наступному етапі необхідно виділити характерні точки зображення та описати їх векторні характеристики. Для цього обирається детектор AKAZE.

Алгоритм Accelerated-KAZE (AKAZE) у 2013 році [17], який також заснований на нелінійній дифузійній фільтрації, як KAZE, але його нелінійні масштабні простори будуються з використанням обчислювальної ефективної структури під назвою Fast Explicit Diffusion (FED). Детектор AKAZE заснований на визначнику матриці Гессе. Якість інваріантності обертання покращено за допомогою фільтрів Шарра. Максимуми відгуків детектора в просторових місцях вибираються як характерні точки. Дескриптор AKAZE заснований на алгоритмі Modified Local Difference Binary (MLDB), який також є високоефективним. Функції AKAZE інваріантні до масштабу, обертання, обмежено працюють з афінними перетвореннями та знаходять більше відповідностей у різних масштабах через нелінійні масштабні простори [18].

Алгоритм використовує FED схеми для побудови нелінійного масштабного простору з урахуванням анізотропної дифузії. Щоб прискорити побудову простору нелінійного масштабу, вбудовується схема FED в пірамідальну структуру. Пірамідальна стратегія та схеми FED дозволяють швидко побудувати простір у нелінійному масштабі, що підходить для надійного виявлення та опису характерних точок [19].

Спочатку визначається набір часів еволюції, з яких можна побудувати масштабний нелінійний простір. Масштабний простір дискретизовано в серії з O октав та S підрівнів. Набір октав та підрівнів ідентифікується дискретним октавним індексом o та підрівнем s. Індекси октави та підрівня зіставляються з їхньою відповідною шкалою σ (пікселі):

$$\sigma_i\left(o,s\right) = 2^{o+s/S},\ o \in \left[0...O-1\right],\ s \in \left[0...S-1\right],\ i \in [0,...M], \qquad (2.1)$$

де M – загальна кількість відфільтрованих зображень.

Тепер потрібно перетворити набір дискретних рівнів масштабу в одиницях пікселів σ_i в одиниці часу, оскільки нелінійна дифузійна фільтрація діє в одиницях часу. Використовуємо відображення $\sigma_i -> t_i$ [20] для перетворення одиниць пікселів у час (2.2).

$$t_i = \frac{1}{2}\sigma_i^2,\ i = \{0...M\}. \qquad (2.2)$$

Крім того, вхідне зображення може бути згорнуто за допомогою гаусса стандартного відхилення σ_0, щоб зменшити шум і можливі артефакти. Зі згладженого вхідного зображення автоматично обчислюємо коефіцієнт контрастності λ як 70% процентів градієнта h_i.

Враховуючи вхідне зображення та коефіцієнт контрасту, розпочинається використання схеми FED. Береться $M-1$ зовнішніх циклів FED і для кожного циклу обчислюється мінімальна кількість внутрішніх кроків n. У випадку 2D зображень максимальний розмір кроку, який не порушує умови стабільності, становить $\tau_{\max} = 25$, враховуючи розмір сітки 1 піксель для похідних зображень. Вважається, що кожен зовнішній цикл FED охоплює час циклу $T = t_{1+1} - t_i$. Час циклу FED θ_n охоплює лише дискретний набір значень. Щоб допустити довільний час циклу T, необхідно обчислити мінімальну довжину циклу n з $\theta_n \geq T$, а потім помножити розміри тимчасових кроків τ_j на коефіцієнт $q = T / \theta_n$ q.

Щоб пришвидшити обчислення простору нелінійної шкали, вставляється схема FED в пірамідальний підхід від тонкого до грубого. Це пов’язано з тим, що в таких програмах, як зменшення шуму зображення, фарбування або варіаційний оптичний потік, зазвичай цікавить стійкий стан

еволюції $t \to \infty$. Щоб якомога швидше досягти стійкого стану, Cascadic FED поширює підхід від грубого до тонкого. Однак у нашому випадку важливе отримання набору відфільтрованих зображень для побудови простору нелінійного масштабу, з якого можна виявляти та описувати особливості.

Як тільки досягається останній підрівень в кожній октаві, зменшується дискретизація зображення в 2 рази, використовуючи маску згладжування $\frac{1}{4},\frac{1}{2},\frac{1}{4}$, і використовується це зменшене зображення як початкове зображення для наступного циклу FED у наступній октаві. Після зменшення дискретизації зображення потрібно також змінити параметр контрасту λ. Маска згладжування зменшує контраст ідеального кроку на 25%, і тому параметр контрасту потрібно помножити на 0,75.

Обчислюємо визначник гессиана L^i для кожного з відфільтрованих зображень у нелінійному просторі. Набір диференціальних мультимасштабних операторів нормується щодо використання нормованого масштабного коефіцієнта, який враховує октаву кожного конкретного зображення в нелінійному масштабному просторі, тобто $\sigma_{i,norm} = \sigma_i / 2^{o^i}$:

$$L^i_{Hessian} = \sigma^2_{i,norm}(L^i_{xx}L^i_{yy} - L^i_{xy}L^i_{xy}). \qquad (2.3)$$

Для обчислення похідних другого порядку використовується з’єднані фільтри Шарра з розміром кроку $\sigma_{i,norm}$. Фільтри Шарра наближають інваріантність обертання краще, ніж інші фільтри або диференціювання центральних відмінностей [21]. Спочатку шукаємо максимуми реакції детектора в просторовому розташуванні. На кожному рівні еволюції i перевіряємо, що реакція детектора вища за попередньо визначений поріг, і це максимум у вікні 3×3 пікселі. Це робиться для того, щоб швидко відкидати відповіді, які не мають максимуму. Потім для кожного з максимумів

потенціалу перевіряємо, що відповідь є максимумом щодо інших ключових точок рівня $i+1$ та $i-1$, відповідно, безпосередньо над i та безпосередньо під ним у вікні розміром $\sigma_i \times \sigma_i$ пікселів. Нарешті, двовимірне положення ключової точки оцінюється з субпіксельною точністю шляхом підгонки 2D квадратичної функції до визначника реакції Гессе в околиці 3×3 пікселів і знаходження її максимуму.

Використовуючи такий алгоритм, знаходимо характерні точки зображення та обчислюємо їх векторні характеристики.

2.3 Порівняння зображень

2.3.1 Відбір зображень схожих за змістом на базі нейромережевого підходу

Першим кроком порівняння зображень є етап відбору схожих за змістом зображень. Для цього використаємо збережені результати роботи нейромережевого кроку. Кожне зображення містить по п’ять тегів, відсортовані по ймовірності присутності. Так як зображення можуть відрізнятись певними геометричними перетвореннями, то використовувати перевірку по всі п’яти тегам не доцільно. Через це, відбираємо для перевірки тільки ті зображення, у яких збігається перший тег, та хоча б які ще два. Наступні порівняння починаються з тих, у яких повністю збіглись всі п’ять тегів, якщо такі є, та йдуть по зменшенню кількості збігів. Якщо ж таких зображень не було знайдено, то скоріш за все це нове зображення і у колекції ще немає даних про таких набір тегів.

2.3.2 Відбір зображень на основі аналізу характерних точок

2.3.2.1 Пошук відповідних дескрипторів на зображеннях

Розраховані векторні характеристики для обох зображень надають змогу знайти відповідності між характерними точками зображень. Для цього використовується метод *k*-ближчих сусідів.

Метод найближчих сусідів – найпростіший метричний класифікатор, заснований на оцінюванні подібності об’єктів. Об’єкту присвоюється той клас, який мають найближчі до нього об’єкти з відповідної вибірки.

Детектор AKAZE бінарний, але результатом його роботи є вектор характеристик для кожної точки. Значення дистанції для кожного атрибуту може відрізнятись і не бути коректним. Саме тому отримані дані підлягають нормалізації. Існує два основних способи нормалізації даних: мінімакс-нормалізація і *Z*-нормалізація [22]. Виконаємо мінімакс-нормалізацію:

$$x' = (x - \min[X]) / (\max[X] - \min[X]), \tag{2.4}$$

де x – значення атрибуту.

Після нормалізації всі значення будуть лежати в діапазоні від 0 до 1.

Так як розраховані дескриптори є векторними ознаками, то для їх порівняння можливо розрахувати Евклідову відстань:

$$D = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}, \tag{2.5}$$

де n – кількість атрибутів.

Нехай задана вибірка пар, $X^m = \{(x_1, y_1), ..., (x_m, y_m)\}$, та на безлічі об’єктів задана функція відстані D. Чим більше значення цієї функції, тим менш схожими є два об’єкти x, y. Для довільного об’єкта u розташуємо об’єкти вибірки y_i у порядку зростання відстані до u:

$$D(u, y_{1;u}) \le D(u, y_{2;u}) \le ... \le D(u, y_{m;u}), \quad (2.6)$$

де через $y_{i;u}$ позначається той об'єкт вибірки, який є *i*-м сусідом об'єкта u.

У даному випадку необхідно виконати порівняння з усіма обчисленими дескрипторами та обирається дві найближчі пари, тобто $k = 2$.

Незважаючи на те, що на цьому кроці були відібрані кращі пари для кожної характерної точки першого зображення (першим зображенням в парі вважається зображення з найменшою кількістю характерних точок), це не означає що зображення є подібними та навіть не означає, що значення дескрипторів у парах є дуже схожими. Для подальшої роботи з парами точок, необхідно відібрати тільки такі пари, значення відстані між двома ближніми сусідами, не перевищую заданий поріг. Задамо поріг подібності відстаней між двома знайденими відповідностями для кожної точки у 75% [23]. Таким чином, обираються тільки ті пари характерних точок двох зображень, для котрих значення різниці відстаней двох кращих пар не перевищує заданий поріг.

Якщо використовувати тільки значення одного найближчого сусіда, то кількість знайдених пар буде завжди дорівнювати кількості характерних точок меншого зображення, що сприяє появі великої похибки обчислення.

Результатом цього кроку є формування пар характерних точок двох зображень, де дескрипторам характерної точки першого зображення відповідає найближче значення дескрипторів характерної точки другого зображення. Кількість сформованих пар лежить у інтервалі $[0, N]$, де N – кількість характерних точок меншого зображення.

2.3.2.2 Усунення хибних відповідностей

Зображення, що порівнюються, можуть відрізнятись геометричними перетвореннями, такими як зміна масштабу, поворот, відсікання частин. Через це можуть сформуватись хибні парі характерних точок, які необхідно відфільтрувати. На цьому кроці маємо пари характерних точок двох зображень які пройшли перший поріг подібності, та значення таких пар не менше ніж 4 для можливості подальшої роботи над зображеннями [24].

Для встановлення відповідностей між зображеннями, треба побудувати модель геометричних перетворень [25]. Для цього будується матриця гомографії. Вона має вигляд (2.7):

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}. \tag{2.7}$$

Нехай (x_1, y_1) – відповідні точки першого зображення, а (x_2, y_2) – другого. Таким чином матриця Гомографії приймає вигляд (2.8):

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}. \tag{2.8}$$

Дане рівняння справедливо для всіх пар характерних точок.

Отримані параметри геометричного перетворення надають можливість використати алгоритм RANSAC.

Random sample consensus (RANSAC) є ітераційним методом для оцінки параметрів математичної моделі з набору спостережуваних даних, що містять шуми, коли шуми не впливають на значення оцінок. Тому це також може

бути інтерпретоване як метод виявлення шумів [26]. Це недетермінований алгоритм в тому сенсі, що він дає розумний результат лише з певною вірогідністю, при цьому ймовірність зростає, чим більше дозволено ітерацій.

Основне припущення полягає в тому, що дані складаються з «відповідних даних» (inliers), тобто даних, розподіл яких може бути пояснено деяким набором параметрів моделі, хоча вони можуть бути предметом шуму, і «не відповідних даних» (outliers), які є даними, що не відповідають моделі. Вони можуть надходити, наприклад, від екстремальних значень шуму, від помилкових вимірювань або неправильних гіпотез щодо інтерпретації даних. RANSAC також передбачає, що, зважаючи на (як правило, невеликий) набір inliers, існує процедура, яка може оцінити параметри моделі, яка оптимально пояснює або відповідає цим даним.

Алгоритм RANSAC – це технологія навчання для оцінки параметрів моделі шляхом випадкової вибірки спостережуваних даних. Враховуючи набір даних, елементи даних яких містять як відповідні, так і не відповідні дані, RANSAC використовує схему голосування для пошуку оптимального результату. Елементи даних у наборі даних використовуються для голосування за одну або кілька моделей. Реалізація цієї схеми голосування ґрунтується на двох припущеннях: про те, що елементи шуму не будуть голосувати послідовно за будь-яку єдину модель (декілька не відповідних даних), і є достатньо можливостей для узгодження хорошої моделі (мало даних, що відсутні). Алгоритм RANSAC по суті складається з двох етапів, які повторюються ітеративно.

На першому кроці із вхідного набору даних випадково обирається мінімальний набір даних. Примірна модель та відповідні параметри моделі обчислюються з використанням лише елементів цього набору зразків. Потужність вибіркової підмножини – це найменше значення, що достатньо для визначення параметрів моделі.

На другому кроці алгоритм перевіряє, які елементи всього набору даних узгоджуються з моделлю, ініційованою розрахунковими параметрами

моделі, отриманими з першого кроку. Елемент даних буде розглядатися як не відповідний, якщо він не відповідає підібраній моделі, ініційованій набором оцінених параметрів моделі в межах деякого порогу помилки, який визначає максимальне відхилення, яке можна віднести до впливу шуму.

Набір inliers, отриманий для підібраної моделі, називається набором консенсусу. Алгоритм RANSAC повторює два етапи, поки отриманий консенсус, встановлений у певній ітерації, не буде мати достатню кількість відповідних даних.

Вхід до алгоритму RANSAC – це набір спостережуваних значень даних, спосіб пристосування якоїсь моделі до спостережень та деякі параметри довіри. RANSAC досягає своєї мети шляхом повторення наступних кроків:

– обирається випадкова підмножина вихідних даних. Вона називається гіпотетичними відповідними даними;

– модель підходить до набору гіпотетичних відповідних даних;

– всі інші дані потім тестуються на встановлену модель. Ті точки, які добре відповідають оціночній моделі, згідно з деякою функцією втрат, що враховуються для моделі, вважаються частиною набору консенсусу;

– оціночна модель є досить хорошою, якщо досить багато балів було віднесено до складу набору консенсусу;

– після цього модель може бути вдосконалена шляхом її повторної оцінки за допомогою всіх членів набору консенсусу.

Ця процедура повторюється фіксовану кількість разів, кожен раз створюючи або модель, яку відкидають, оскільки занадто мало балів є частиною набору консенсусу, або вдосконалену модель разом із відповідним розміром набору консенсусу. В останньому випадку зберігається вдосконалена модель, якщо її набір консенсусу більший, ніж збережена раніше модель.

Таким чином, отримується набір пар важливих точок і відкидаються усі хибні.

2.3.3 Розробка критерію про підозру на плагіат на основі методу RANSAC

Якщо після використання методу RANSAC не було відкинуто жодної пари, то значить над зображенням не проводилось жодних геометричних перетворень. А так як на цьому кроці вже було знайдено достатню кількість відповідностей, то можемо цілком впевнено вважати таке зображення плагіатом. У випадку, коли знайдених відповідних пар недостатньо для побудови матриці гомографій, то скоріш за все зображення різні, або видозмінені настільки, що вже майже неможливо встановити оригінальний вигляд.

На виході роботи алгоритму отримується набір пар характерних точок, який порівнюється з кількістю відібраних пар до виконання алгоритму RANSAC. Нехай N_0 – кількість відповідних пар до RANSAC, а N_1 – кількість відповідних пар після. Тоді можемо розрахувати відсоток подібності між зображеннями (2.9):

$$P = (N_1 / N_0) * 100\% . \quad (2.9)$$

Прийняття рішення щодо визнання зображенням плагіатом базується на отриманому відсотку подібності. Якщо він більший за заданий поріг, то вважається, що дане зображення з високою вірогідністю є плагіатом. Чим більше наближення до 100%, тим більша вірогідність. Для ідентичних зображень це значення завжди дорівнює 100%.

2.4 Алгоритм пошуку зображень, що є підозрілими на плагіат, у текстових файлах

Розглянувши всі кроки, необхідні для перевірки зображень, можна об’єднати їх в єдиний алгоритм та встановити критерії для прийняття рішення про плагіат.

Все починається з підготовки зображень, які будуть порівнюватись. Щоб уникнути випадків плагіату за зміною кольорів зображення та пришвидшити роботу програми, зображення треба перевести у градієнт сірого. Наступним кроком буде виявлення змісту зображення, щоб створити першочергове уявлення з якими саме зображеннями потрібно порівнювати нове зображення користувача. Виділяється п’ять тегів, відсортовані по імовірності появлення. Далі, за допомогою детектора AKAZE необхідно виділити всі характерні точки разом з набором векторних характеристик цих точок. Якщо це перше зображення у колекції, воно заноситься до неї. Наступні зображення будуть мати додаткові кроки. Щоб уникнути порівняння кожного нового зображення з кожним у колекції, спочатку відбувається відбір зображень за змістом. Якщо перший, та ще будь які два теги співпадають, то таке зображення можна використовувати для порівняння. Якщо таких зображень не знайдено, то вихідне зображення додається до колекції. Порівняння, а саме пошук відповідностей між зображеннями, починаються з зображень, які мають найбільшу кількість спільних тегів. Для цього використовується метод *k*-найближчих сусідів та обираються дві найближчі пари. Розраховується Евклідова відстань та відбираються тільки ті пари, для яких подібність відстаней між двома знайденими відповідностями не менше 75%. Тепер треба прийняти рішення, чи варто переходити на наступний крок, чи зображення вже можна вважати не подібними. Розглянемо відношення кількості знайдених пар до кількості характерних точок. Якщо знайдене відношення більше ніж перший заданий поріг, то переходимо до наступного кроку, а якщо менше – до наступного

зображення. Так як зображення можуть відрізнятись геометричними перетвореннями, то необхідно отримати параметри геометричних перетворень методом RANSAC, та усунути хибні пари. Після цього залишається оцінити відсоткове значення спільних пар, що залишились, тобто кількість пар до алгоритму, та кількість після. Якщо отримане відношення більше за другий встановлений поріг, то зображення вважаються плагіатом. При значенні у 100% зображення вважаються ідентичними, а зменшення відсотку показує, що над зображенням проводились певні геометричні перетворення. Спрощений алгоритм можна побачити на рисунку 2.1.

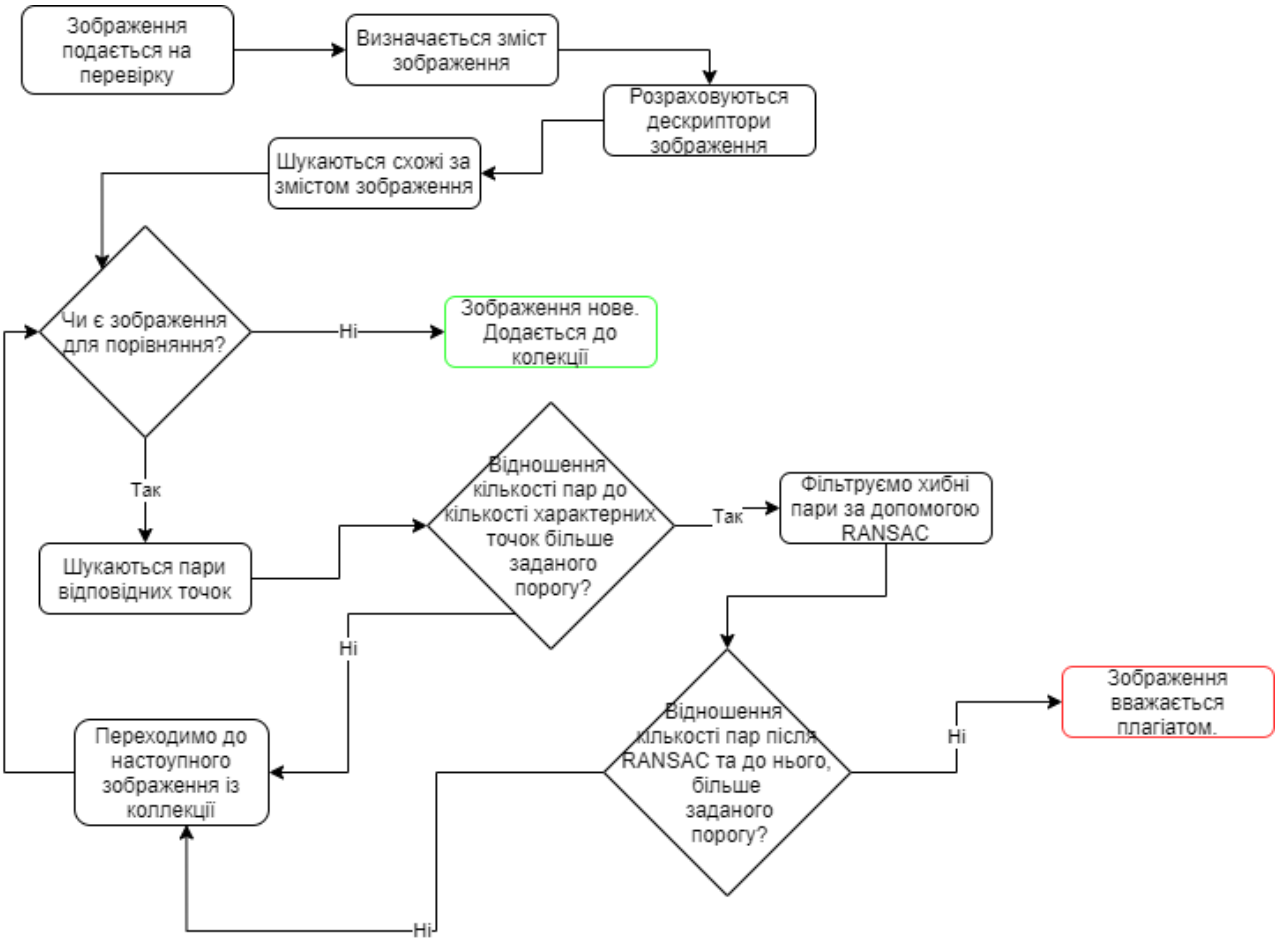

Рисунок 2.1 – Схема алгоритму прийняття рішення про плагіат

2.5 Дослідження ефективності обраних критеріїв та налаштування параметрів

Для того, щоб дати відповіді на запитання, чи потрібно переходити до наступного кроку порівнянь, та яке саме значення міри подібності вважати плагіатом, проведемо дослідження. Зауважимо, що найбільш поширеними перетворенням зображення у текстових документах вважаємо зміну масштабу та відсікання частин. Для кращого розміщення зображення у документах весь час доводиться змінювати розміри зображення, тож частіше за все зображення в документах та в колекції будуть різного розміру. Через це візьмемо якесь зображення, та створимо його копії зі рівномірним зменшеннями масштабу, починаючи 25% відсотками оригінального масштабу, та закінчуючи 200%. Зображення більшого масштабу, здебільшого, не зможуть гарно поміститись у документі, а ще менші зображення буде важко розглядіти та використовувати.

Такими чином було створено 36 копій зображень – включаючи й копію зі 100% масштабу, тобто, оригінал. Перш за все, знаходимо залежність кількості знайдених характерних точок від відсотку масштабу. Для цього використовується дескриптор AKAZE.

Для того щоб уникнути випадкових значень та мати усереднену картину, цей, та подальші досліди, проводиться з десятьма різними за початковими розмірами, змістом, формами зображеннями, та відображається середнє значення. Результати пошуку характерних точок можна побачити на рисунку 2.2. Отриманий графік є майже прямою лінією, і ілюструє, що зі збільшенням масштабу, збільшується й кількість знайдених характерних точок. Кількість характерних точок зображення може бути різною, в залежності від використаного дескриптора, та може трохи відрізнятись для схожих зображень, у випадку якщо вони були стиснені для зменшення ваги. Одні й ті ж самі зображення з використанням одного й того ж самого дескриптора завжди мають однакову кількість характерних точок.

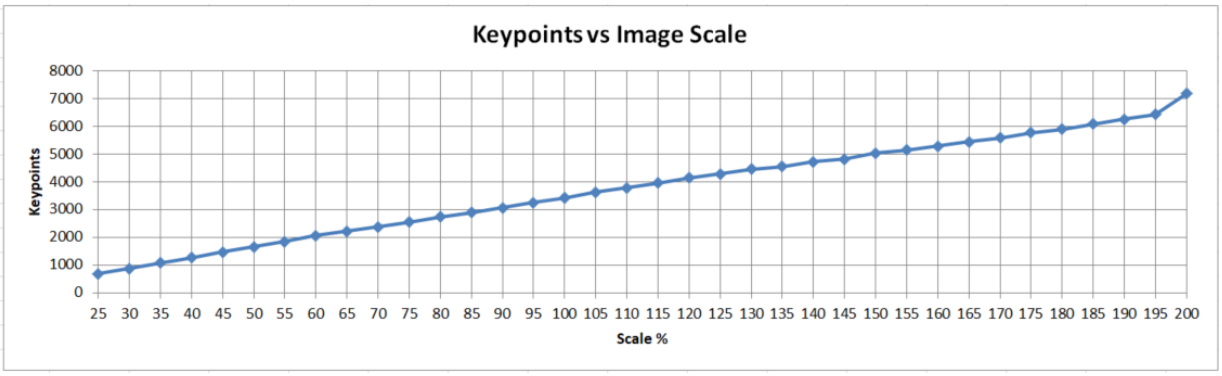

Рисунок 2.2 – Графік залежності кількості характерних точок від масштабу зображення

Тепер застосуємо метод *k*-ближніх сусідів для знаходження кількості відповідних пар характерних точок на оригінальному (масштаб 100%) зображенні, та зображеннях зі зміненим масштабом. Середні значення представлені на рисунку 2.3.

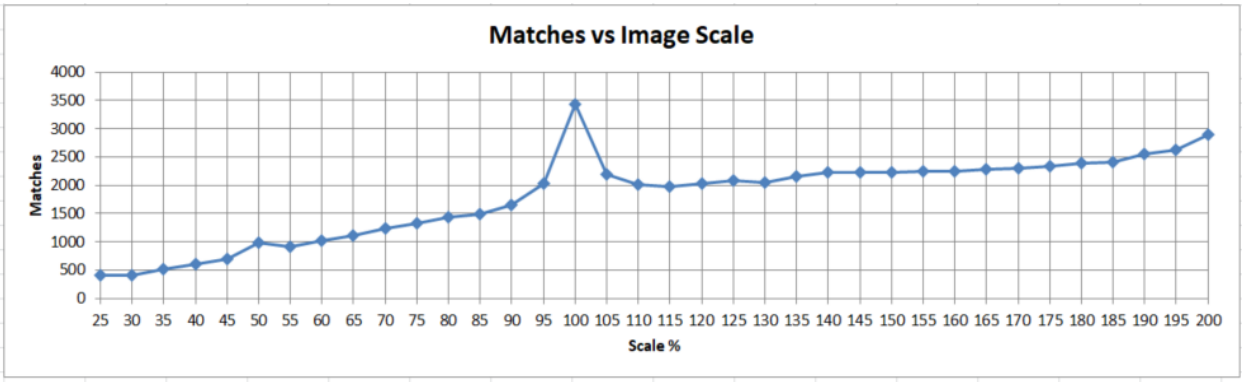

Рисунок 2.3 – Графік залежності кількості знайдених відповідностей від масштабу зображення

Отриманий графік показує, що чим ближче показник масштабу до оригіналу, тим більше відповідних пар було знайдено. Можна побачити, що для інших значень не можна встановити якусь певну відповідність. Кількість знайдених пар може як збільшитись, так і зменшитись, у порівнянні з результатами наступного за масштабом зображення. Наявність такої непередбачуваності можна пояснити обраним методом відбору пар. Встановивши поріг для значень дистанції між двома кращими точками у

75%, маємо невелику вірогідність як розглядати хибні точки, так і пропустити деякі потрібні [27]. Розглянемо відношення кількості знайдених пар до кількості характерних точок, для тих же самих зображень, що відрізняються масштабом (рис. 2.4).

Отриманий графік дозволяє побачити, що відсоток знайдених відповідностей в середньому не сягає нижче 40%. Розглядаючи створені дата сети окремо, було встановлено, що мінімальне значення відповідності лежить в інтервалі [0,38, 0,42]. Як і було встановлено до цього максимальне значення у 100% досягається тільки при порівнянні двох однакових зображень.

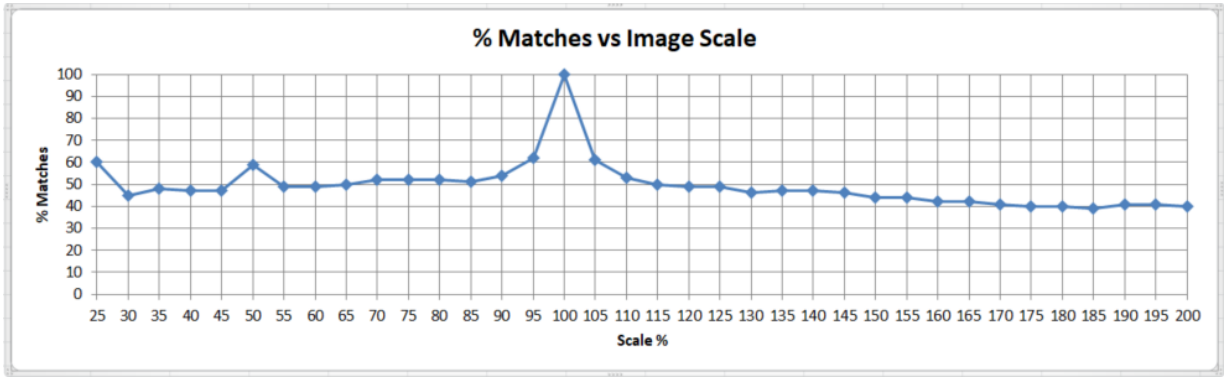

Рисунок 2.4 – Графік залежності відсотку знайдених відповідностей від масштабу зображення

Наступним кроком є розрахунок кількості пар, що залишаються після роботи алгоритму RANSAC. Середні значення для зображень із різним масштабом у порівнянні з оригіналом можна побачити на рисунку 2.5.

Отриманий графік дуже схожий на графік відношення кількості знайдених пар до відсотку масштабу зображень (рис. 2.3). Це дає змогу судити, що кількість знайдених пар у зображенні з плагіатом після використання алгоритму RANSAC залишається майже без змін.

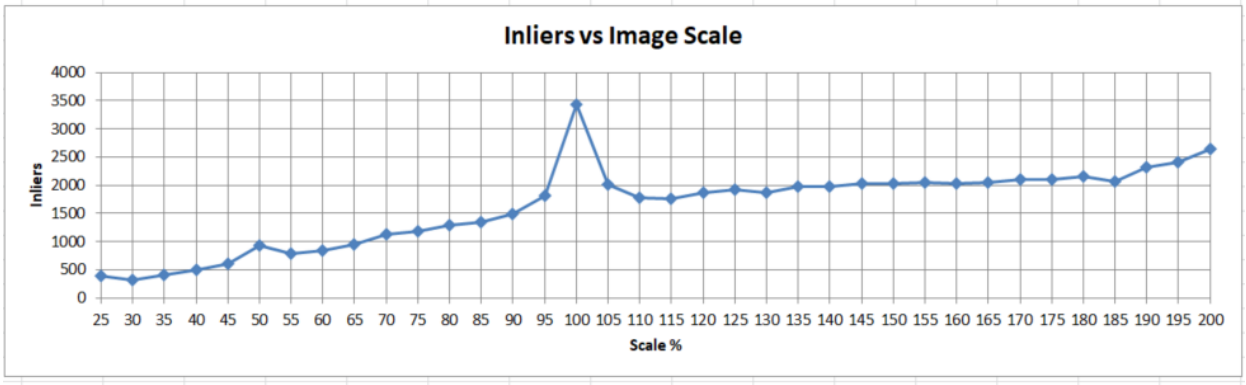

Рисунок 2.5 – Графік залежності знайдених відповідностей після RANSAC від масштабу зображення

Крім того, кількість знайдених пар при порівнянні оригіналу з оригіналом продовжує залишатись тією самою, що підтверджує доцільність використання даного алгоритму.

Для дослідження відсотку плагіату, побудуємо ще один графік. У цей раз розрахуємо відсоток подібності між зображеннями (2.8). Результат відношення відсотку подібності до відсотку масштабу зображення можна побачити на рисунку 2.6.

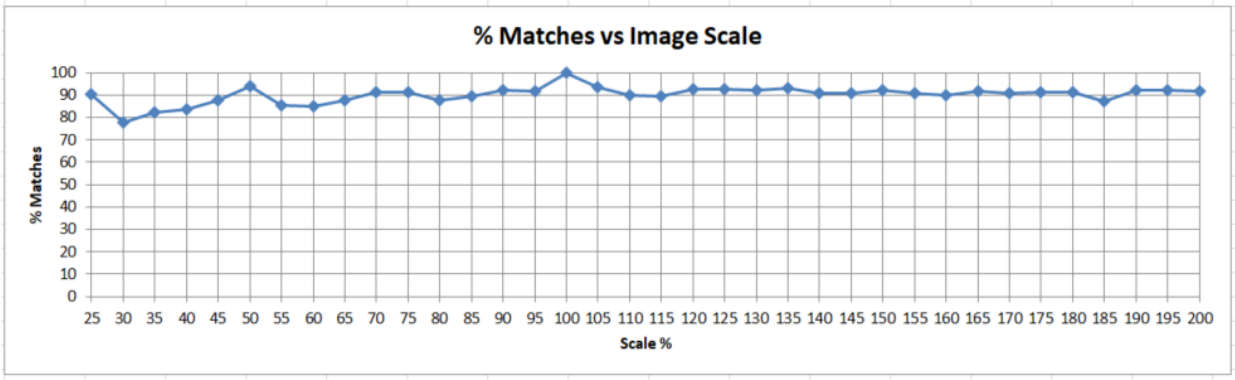

Рисунок 2.6 – Графік залежності відсотку подібності від масштабу зображення

В цей час також використовувались 10 різних зображень та 36 копій кожного з різницею тільки у масштабі. На графіку представлені середні значення для всіх десяти зображень. Як можна побачити, значення відсотку

подібності не сягає нижче 75%, та дорівнює 100% у випадку порівняння двох однакових зображень. Розглядаючи значення всіх результатів по одинці було встановлено, що найменше значення подібності знаходиться у діапазоні [0,76, 0,78].

Таким чином можна зробити висновок, що зображення можна вважати плагіатом, якщо відсоток подібності є вищим за 75%. Крім того, було встановлено, що відношення між кількістю знайдених пар та кількістю характерних точок повинно бути більше 38%.

Для того щоб встановити, чи мають отримані значення дійсне застосування при порівнянні двох зображень, також необхідно дослідити значення відсотку знайдених пар та відсотку подібності при порівнянні двох зображень, що не являються плагіатом. Для цього візьмемо ті ж самі 10 різних оригінальних зображень, та колекцію з 50 відмінними зображеннями. Зробимо незалежні порівняння кожного оригінального зображення із кожним зображенням із колекції. Середні значення порівняння зобразимо у виглядів діаграм. Перша діаграма – відношення кількості характерних точок до кількості знайдених пар (рис. 2.7). Створена гістограма складається з стовбців, що представляють унікальні зображення, які використовувались у порівняннях. Для наочності представлено тільки 19 зображень з найбільшою кількістю знайдених пар. Кожен стовбець складається з двох чисел: кількості характерних точок (зелений колір, підпис всередині), та кількості створених пар (червоний колір, підписи зверху).

На відміну від значень отриманих для однакових зображень, що відрізняються лише масштабом (рис. 2.4), можна побачити, що для відмінних зображень кількість створених пар дуже низька у відсотковому відношенні до кількості знайдених характерних точок. Відсоток відповідності знайдених пар до кількості характерних точок не перевищує 5% (додаток А).

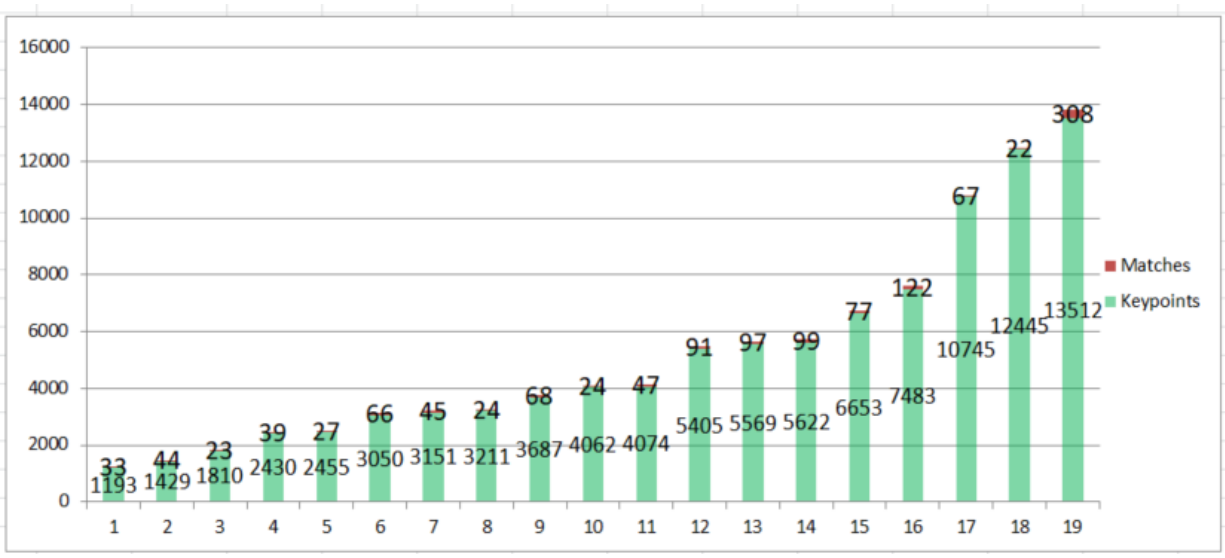

Рисунок 2.7 – Діаграма порівняння кількості знайдених характерних точок і кількості створених пар

Це дає підстави вважати, що поріг у 38% для відношення кількості знайдених пар до кількості знайдених характерних точок є надійним і може використовуватися для припинення порівняння зображень, якщо він не був подоланий.

Залишається перевірити який відсоток подібності будуть мати не подібні зображення. Для цього використаємо ті ж самі правила та зображення, як і в попередньому дослідженні. Будемо вважати, що всі зображення пройшли попередній поріг. Результати представлені на рисунку 2.8.

Отримана діаграма наглядно ілюструє велике зменшення кількості відповідних пар після алгоритму RANSAC для різних зображень, на відміну від дуже малих змін для зображень, що вважаються плагіатом. Міра подібності жодного із зображень не перевищує 45% (додаток Б). Таким чином, обраний поріг в 75% можна вважати надійним показником для виявлення плагіату.

Крім того, опираючись на результати попереднього дослідження з цими ж самими зображенням, у підрахунки відсотку відношення знайдених пар до кількості характерних точок, можна зробити висновок, що жодне із

цих зображень навіть не буде розглядатись як плагіат, та для них не буде будуватись матриця гомографії та виконуватись алгоритм RANSAC.

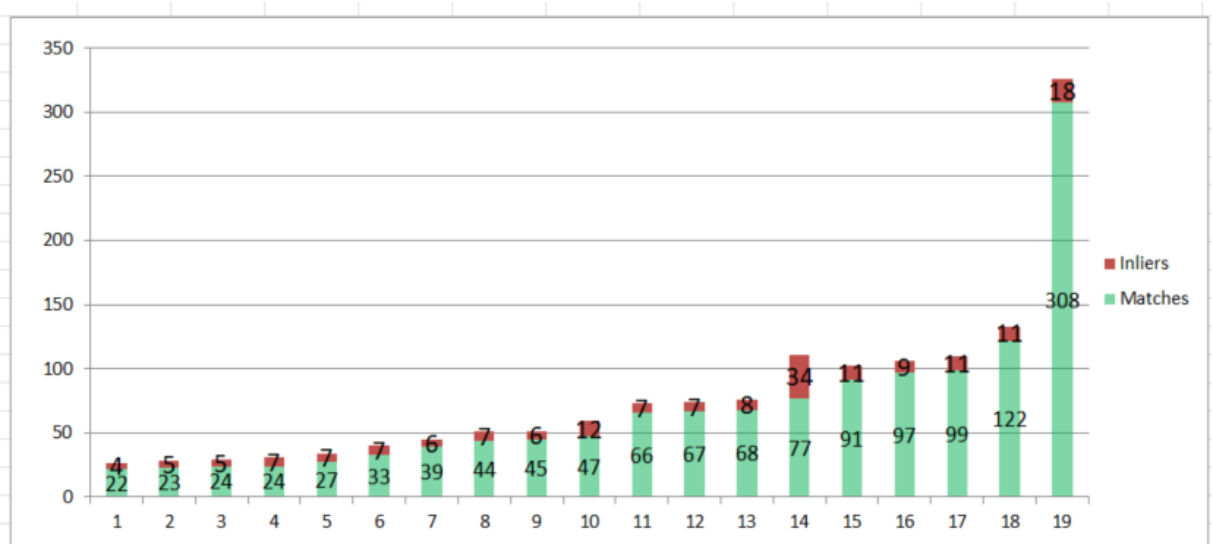

Рисунок 2.8 – Діаграма порівняння кількості створених пар, та їх кількості після RANSAC

3 РОЗРОБКА КОМП'ЮТЕРНОЇ МОДЕЛІ

3.1 Архітектура застосунку

3.1.1 Використання вебфреймворку для створення головних модулів системи

Перед власне створенням вебзастосунку, перш за все потрібно розробити архітектуру застосунку з використанням запроваджених властивостей фреймворку. Обраний фреймворк Django використовує шаблон проєктування Модель-вигляд-контролер (Model-view-controller) [28]. Це така схема поділу даних програми, інтерфейсу користувача і керуючої логіки на три окремих компоненти: модель, вигляд і контролер – таким чином, що модифікація кожного компонента може здійснюватися незалежно. Для спрощеного розуміння такої архітектури, можна виділити клієнтську та серверну сторонку (рис. 3.1).

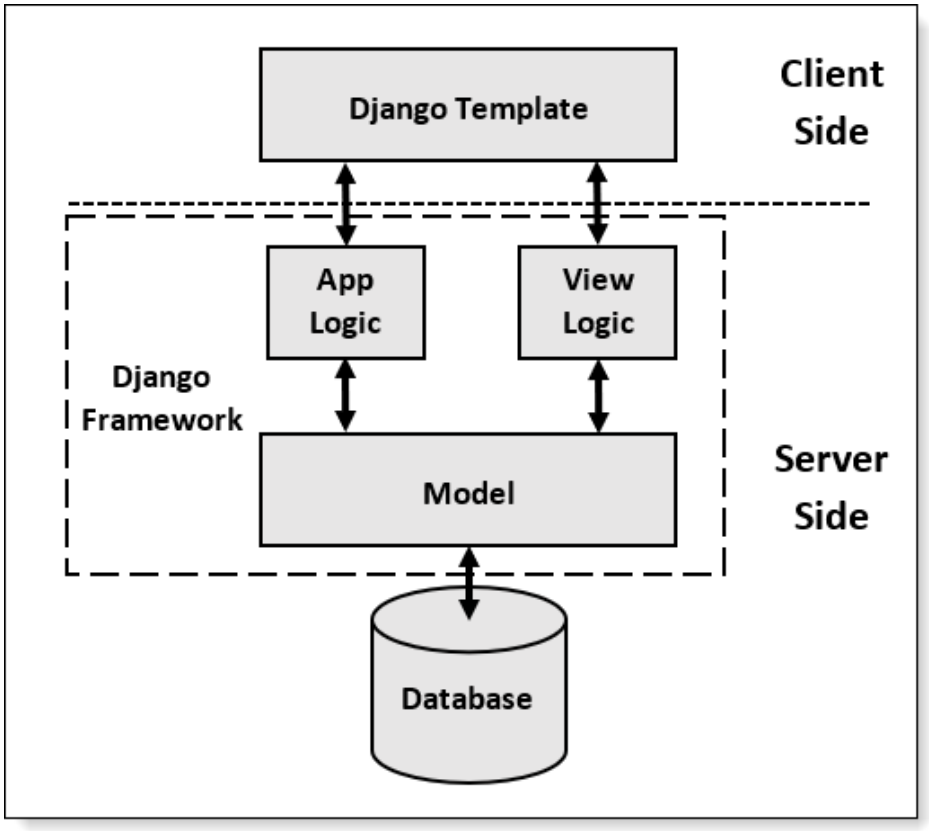

Рисунок 3.1 – Загальна архітектура застосунку

Користувач бачить вже готові HTML сторінки, але спочатку вони мають вигляд шаблонів. Такий шаблон може містити всі необхідні стилі, розмітку, але головний контент потрапляє у шаблон з серверної частини. Django Templates дозволяють навіть додавати до шаблонів цикли операторів, оператори умови та інші частини коду, які під час генерації перетворюються на повноцінну сторінку. Зможемо використати такий підхід для відображенні у циклі всіх знайдених зображень, що є підозрілими на плагіат, а також створити умови для відображення лише тексту, що плагіат не було знайдено.

На серверній частини необхідно створити декілька функцій для обробки запитів за певними адресами. Ці функції будуть приймати файли користувачів, виконувати необхідну обробку та віддавати у відповідь згенеровану HMTL-сторінку.

Знайдені та оброблені зображення, а саме 5 тегів змісту, набір характерних точок та набір дескрипторів додається до моделі зображення. Ця модель є представленням таблиці у базі даних через класс на мові програмування Python. А за допомогою Django ORM [29] виконується зв’язок між кодом програми та базою даних.

Базовими необхідними функціями, які повинен мати вебзастосунок, є: сторінка для завантаження файлу із зображеннями, сторінка з результатами перевірки, сторінки адміністратора з можливістю подивитись існуючі зображення та очистити колекцію, при необхідності, та сторінка авторизації.

Для створення сторінки завантаження файлу також використовуються властивості фреймворку, а саме гнучкі форми. На рівні сервера треба створити класс форми, та описати типи даних. Також, можна задати правила валідації цих даних. Для нашого випадку, це буде форма з одним обов’язковим полем «Файл». Ця форма передається до шаблону сторінки і фреймворк самостійно перетворить її у звичний для користувача вигляд, з кнопкую вибору файла у файловій системі та кнопкою підтвердження. Для цієї сторінки треба створити унікальний URL, та створити функцію обробки для цього URL. Дані, відправлені користувачем необхідно валідувати. Для

цього використовуються серіалайзери. Якщо форма містить невалідні дані, то необхідно повернути користувачу помилку. Інакше, використати функції пошуку зображень, підозрілих на плагіат, та відобразити результати пошуку на новій сторінці.

Сторінка з результати перевірки повинна містити текст про результат перевірки. Якщо підозрілих на плагіат зображень не було знайдено, тоді необхідно відобразити відповідний текст. У іншому випадку, необхідно відобразити усі знайдені зображення у вигляді: зображення з файлу користувача, оригінальне зображення, розрахований відсоток подібності. Django Templates повністю задовольняють потреби для створення цієї сторінки, так як єдине, що потрібно відправити до шаблону сторінки, це список знайдених зображень та необхідні дані для відображення. Використовується функція умови, і використовується динамічне відображення в залежності від результату перевірки.

Сторінки адміністратора, та сторінку авторизації немає необхідності створювати власноруч, так як Django містить їх за замовчуванням [30].

Схему можливих суттєвих дій, які повинні мати можливість робити користувач та система, представлені у вигляді use-case діаграми (рис. 3.2).

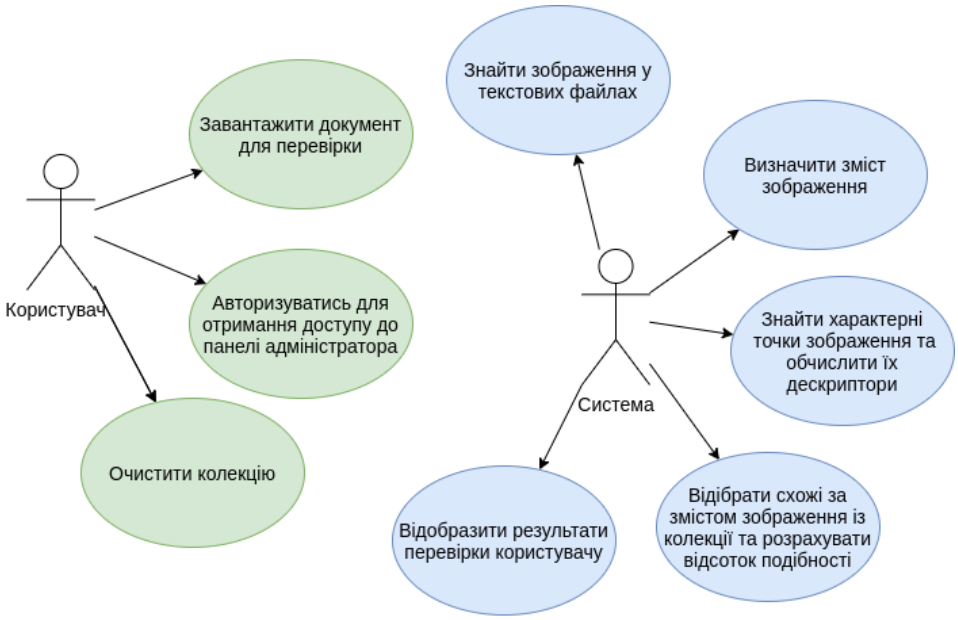

Рисунок 3.2 – Use-Case діаграма застосунку

3.1.2 Проєктування бази даних

Django фреймворк підтримує роботу з великою кількість реляційних та нереляційних систем управління базами даних. Серед необхідних властивостей для вирішення поставлених задач важливо, щоб обрана СУБД підтримувала дані у JSON форматі, та пошукові запити потребували якомога менше часу. Авжеж, СУБД повинна бути безкоштовною та легкою для локального використання на різноманітних операційних системах. Таких варіантів лише два: реляційна СУБД PostgreSQL та нереляційна MongoDB. нереляційні бази можуть з легкістю знаходити необхідні дані за певними ключами та зберігати великі обсяги даних у незвичних для зберігання форматах [31]. Але з оглядом на те, що PostgreSQL може дуже швидко працювати по індексам, а індексами у нашому випадку будуть виступати звичайні строки, та у подальшому можуть потребуватись додаткові зв’язки з такими таблицями як Користувач, Колекція, то перевага віддається саме цій реляційній СУБД.

Для реалізації мінімального життєздатного продукту необхідно створити таблицю Зображення та Користувача. Обраний фреймворк створює необхідні для авторизації таблиці самостійно.

Таблиця Зображення повинна містити назву, дескриптори, характерні точки, інформацію про завантаження та 5 тегів, по яким створюються індекси.

Таким чином, було створена схема бази даних застосунку (рис. 3.3).

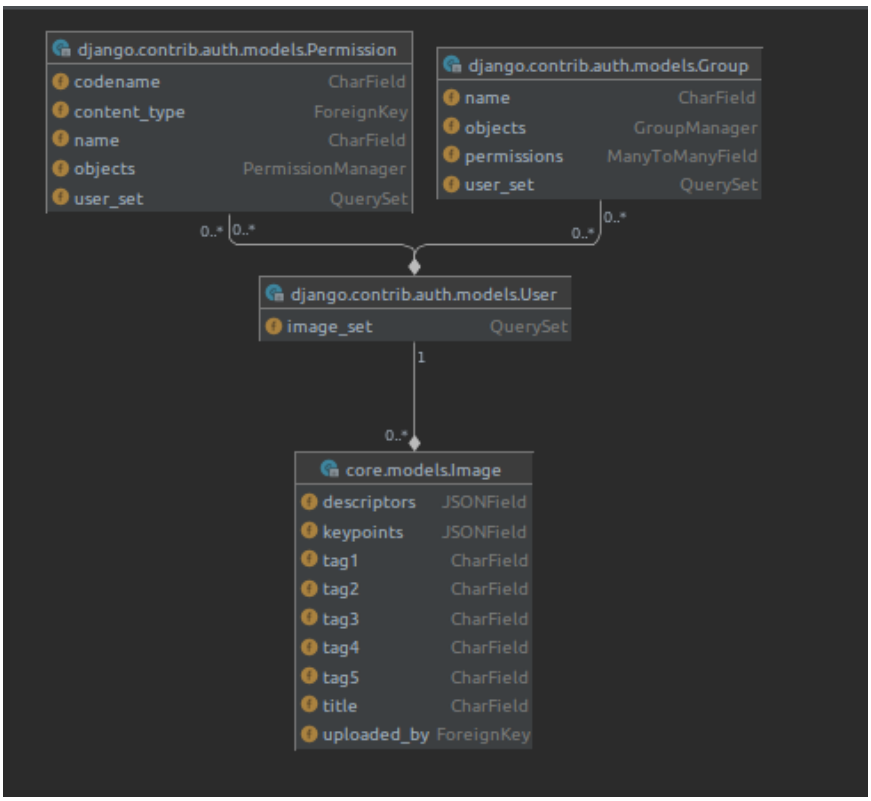

Рисунок 3.3 – Схема бази даних

3.2 Програмна реалізація

Так як для створення застосунку використовується фреймворк, то необхідно притримуватись правил побудови та структури нав’язаної обраним фреймворком. Створення та тестування вебзастосунку буде відбуватись на локальній машині на операційній системі Ubuntu 18. Застосунок може бути побудований й на інших операційних системах, але легше всього це зробити на Linux-подібних. Створення локального вебсерверу не дуже відрізняється від створення глобального серверу, при необхідності розмістити вебзастосунок у мережі Інтернет для надання доступу до нього великої кількості користувачів. У випадку з розміщенням вебзастосунку у мережі Інтернет, необхідно налаштувати вебсервер, наприклад Nginx, який бере на себе обов’язки обробляти запити користувачів, що надходять на певний ip-адрес, передавати їх на обробку фреймворку, та повертати відповідь до користувача. Розглянемо випадок створення локального сервера.

Перш за все, щоб створити проєкт на мові програмування Python, треба мати мову Python у системі. Для створення даного вебзастосунку використовується версія 3.7, яка за замовчуванням встановлена в ОС Ubuntu 18.

Для того щоб встановлювати необхідні для проєкту бібліотеки, створюється віртуальне оточення. Можна сказати, що таким чином виділяється окреме місце, куди будуть ставитись усі необхідні для цього проєкту бібліотеки, замість того щоб ставити їх глобально у систему. Таким чином у системі можуть з легкістю знаходитись декілька проєктів, які можуть потребувати одну й ту саму бібліотеку, але різну її версію.

Наступним кроком є встановлення Django всередині створеного для цього проєкту віртуального оточення. Для цього використовується pip [32]. Це дуже зручний менеджер пакетів, який надає можливість швидко встановлювати необхідні пакети, фреймоворки, бібліотеки, останньої або вказаної версії. Для створення вебзастосунку буде використовуватися Django версії 2.2, тому необхідно ввести команду:

pip install django=2.2

Таким чином, встановлюється як Django, так і все залежності. На цьому етапі маємо все необхідне щоб почати проєкт. Для необхідно перейти до директорії, де буде розміщено проєкт, активувати створене віртуальне оточення, та ввести команду:

django-admin startproject imagesearch.

У цій команді слово imagesearch є назвою проєкту, і може бути довільною. Після виконання цієї команди, створюється папка із вказаною назвою, та певні файли (рис. 3.4).

```
imagesearch/
    manage.py
    imagesearch/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Рисунок 3.4 – Базова структура створеного проєкту

Більшість подальших команд будуть використовуватись з використанням файлу manage.py – керуючого файлу, який приводить в дію велику кількість необхідних команд розробника.

Так як проєкт складається із незалежних компонентів, то необхідно створити програмний засіб який буде містити увесь код, необхідний для подальшої роботи із зображеннями. Створюється застосунок командою:

python manage.py startapp core.

Слово core може бути замінено на будь яке інше, це є назвою застосунку. Структуру створеної програми можна побачити на рисунку 3.5.

```
core/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Рисунок 3.5 – Структура створеного застосунку

Структура проєкту вже майже створена повністю, залишається додати папку static, в якій будуть знаходиться папки styles, scripts, images. Це

статичні дані, які повинні бути доступні з будь-якого місця у проєкті. Django дозволяє створити цю папку за межами проєкту, навіть на іншому диску, вказавши шлях до неї у налаштуванні. У випадку, коли в проєкті використовується велика кількість зображень, яка постійно збільшується то краще розмістити їх на окремому гнучкому диску, та не збільшувати розмір проєкту в цілому. Для вебзастосунку, що розроблюється, це також може бути гарною ідеєю та може бути запроваджено у майбутньому для оптимізації роботи застосунку. Наразі створимо цю папку всередині проєкту.

Останнім пунктом в створені структури застосунку залишається створення папки templates всередині папки core. Це папка, яка буде містити усі шаблони HTML-сторінок, які будуть використовуватись для відображення наповнення.

Створена структура дозволяє перейти до наступного кроку – створення моделей даних. Моделі представлені у виглядів класів, та відображають структуру таблиць БД. Модель користувача вже є в системі за замовчуванням та не потребує в модифікуванні.

Створюється модель таблиці Image (рис. 3.6).

```
class Image(models.Model):
    title = models.CharField(max_length=255)
    uploaded_by = models.ForeignKey(User, blank=True, null=True,
                                    on_delete=models.SET_NULL)
    keypoints = JSONField()
    descriptors = JSONField()
    tag1 = models.CharField(max_length=255, blank=True, null=True)
    tag2 = models.CharField(max_length=255, blank=True, null=True)
    tag3 = models.CharField(max_length=255, blank=True, null=True)
    tag4 = models.CharField(max_length=255, blank=True, null=True)
    tag5 = models.CharField(max_length=255, blank=True, null=True)

    def __str__(self):
        return self.title
```

Рисунок 3.6 – Модель таблиці Image

Дана модель буде використовуватись для полегшення запитів до БД. Не потрібно власноруч створювати SQL запити, Django ORM надає можливість взаємодіяти з даними як з класом, який має велику кількість зручних методів для фільтрації та відбору необхідних значень із таблиці або об’єднання таблиць. Створені запити автоматично конвертуються в SQL скрипти, які повертають результати із бази у вигляді класу набору даних, який дозволяє створювати додаткові фільтрації, сортування, якщо це потрібно.

Для того щоб створена модель змогла виповнювати усі зазначені вище дії, необхідно створити зв’язок з БД. Для цього у файлі налаштування є параметр DATABASES. Його значеннями є список словників, кожен з яких містить відомості про базу даних. У нашому випадку буде використовуватись лише одна база. Заповнюється відомості про тип БД, місце знаходження, назву, порт доступу. За замовчанням використовується SQLite база, що генерується автоматично. Вона не підходить для нашої моделі, яка містить JSON поля, з якими необхідно часто та швидко працювати. Спочатку треба створити пусту базу даних у PostgreSQL та дати їй назву, яку треба записати у налаштуваннях проєкту.

Коли з налаштуваннями закінчено, настає час внесення змін вже до самої БД. Для цього виконується команда:

python manage.py makemigrations

Результатом роботи команди є створення скрипта, який містить перелік змін, що необхідно внести в БД, та є варіантом опису даних (DDL), що будуть інтерпретовані у набір SQL команд, таких як Create Table. Для прийняття всіх створених міграцій, виконується наступна команда:

python manage.py migrate

Після її виконання зміни, що були відображені у моделях, прийнялись і для відповідних таблиць у БД.

На цьому етапі застосунок має готову структуру проєкту та бази даних. Можна переходити до створення шаблонів сторінок.

Першим шаблоном є основним, від якого будуть наслідуватись всі інші сторінки. Він повинен мати всі основні елементи HTML-сторінки, підключені стилі та скрипти. Для кращого подальшого розміщення елементів, використовується бібліотека bootstrap. Крім того, необхідно виділити декілька блоків, які в подальшому будуть змінені у подальших сторінках. Це текст заголовку сторінки, та контент всередині body, який йде після навігації та загальної структури вигляду сторінки. Необхідно пам’ятати, що сторінку поповнення колекції повинні бачити тільки адміністратори, тож шаблон повинен містити перевірки для відображення цього пункту у навігації, його треба відображати тільки якщо користувач, а саме об’єкт моделі користувача, містить позитивне значення у полі is_superuser.

Інші сторінки наслідуються від базовою, та містять форму для завантаження документу, у випадку із сторінками пошуку на плагіат та поповнення колекції, та блоки із зображеннями, що відображаються у циклі, для сторінки результатів.

Наступним кроком є написання алгоритму порівняння зображень. Він був детально описаний у другому розділі цієї роботи, тож відмітимо тільки що певні кроки, такі як знаходження характерних точок то обчислення дескрипторів за допомогою AKAZE, пошук *k*-ближніх сусідів, побудова матриці гомографії та алгоритм RANSAC присутні у бібліотеці OpenCV, та не потребують додаткової роботи над створенням чи модифікуванням цих алгоритмів. Для роботи з нейромережею використовуються бібліотека ImageAi. Насправді, це не єдина єдина бібліотека, вона є лише зручною обгорткою над великою кількістю інших, які автоматично встановлюються по замовчуванню.

Щоб написаний код зміг запуститись без перешкод, спочатку треба встановити всі бібліотеки. А щоб не встановлювати всі по одній, створюється файл requirements.txt до якого вносяться тільки головні бібліотеки з чітко зазначеними версіями (рис. 3.7).

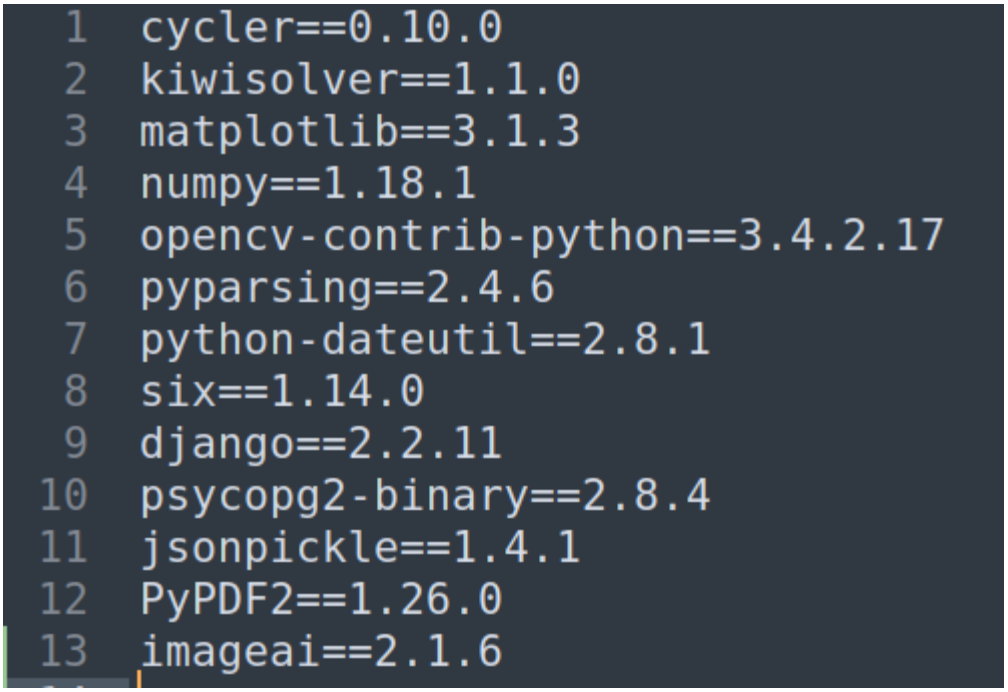

Рисунок 3.7 – Зміст файлу requirements.txt

Для виконання встановлення пакетів з цього файлу використовується команда:

pip3 install -r requirements.txt

Залишається зв’язати створений код із шаблонами сторінок. Для цього використовуються функції перегляду (view), які отримують запит, обробляють його, створюють контекст, та генерують вебсторінку на базі існуючого шаблону, наповнюючи його змістом. Для зв’язування функції перегляду та кінцевої адреси, яку відкриває користувач, використовуються шляхи (path), які й складаються з пари: шлях, обробник.

Створення застосунку можна вважати завершеним. Залишається його запустити, щоб мати доступ до створених сторінок. Для цього використовується наступна команда:

python manage.py runserver:7000

Ця команда запускаю сервер та надає доступ до нього на вказаному порту. У даному випадку сервер працює на 7000 порту, хоча це значення може з легкістю бути змінено.

3.3 Тестування розробленої системи

Розроблений застосунок складається з головної сторінки, на якій пропонується здійснити перевірку на плагіат. Вона містить кнопку, яка дозволяє обрати текстовий файл. При натисненні на неї відкривається вікно, яке надає змогу обрати тільки файли з розширенням doc, docx та pdf (рис. 3.8).

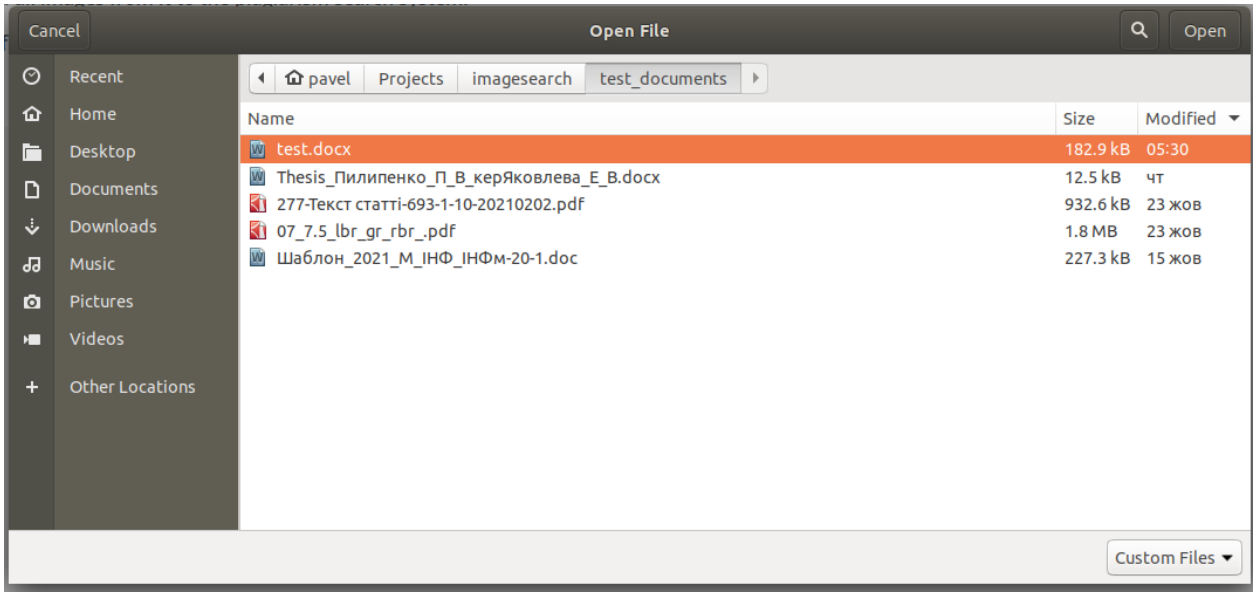

Рисунок 3.8 – Вибір текстового файлу

Тепер необхідно перевірити роботу пошуку подібності зображень. Виберемо файл, в якому міститься зображення із колекції, та одне нове, оригінальне. Очікується, що програма віднайде подібне зображення та виведе його для користувача. Після завантаження файлу, отримуємо результат (рис. 3.9).

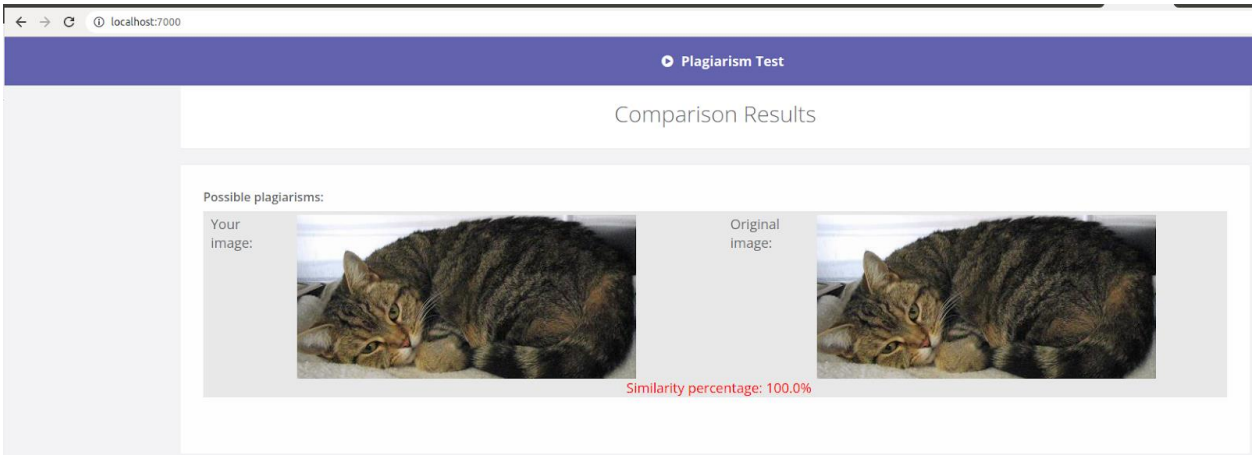

Рисунок 3.9 – Результат перевірки

Як і очікувалось, зображення було знайдено, і відображене тільки подібне. Необхідно ще перевірити, що відобразиться на екрані, якщо зображення не будуть знайдені зовсім. Завантажимо файл із оригінальними зображеннями, яких ще не має у базі.

Отримуємо результат перевірки (рис. 3.10).

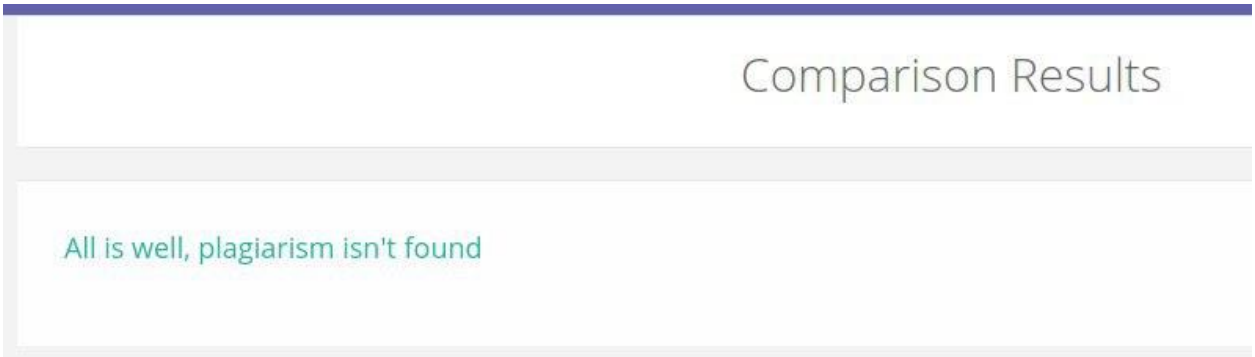

Рисунок 3.10 – Відображення повідомлення про результат перевірки

Тобто, зображень підозрілих на плагіат не було знайдено, і користувач був проінформований про це.

Перевіримо випадки, коли вихідне зображення відрізняється розмірами. Для цього створимо зменшену та збільшену копію одного й того ж самого зображення, яке вже є в колекції. Очікується, що обидва зображення будуть відображені як ті, що є підозрілими на плагіат (рис. 3.11).

Рисунок 3.11 – Перевірка зображень, що відрізняються масштабом

Як можна побачити, при збільшенні зображення відсоток подібності залишається близький до 100, а при зменшенні відсоток нижчий. Це пов’язано з тим, що при зменшенні зображення зменшується кількість характерних точок, тому після фільтрацій навіть декілька хибно відкинутих пар може значно зменшити відсоток подібності.

Залишається перевірити випадок знаходження плагіату в дуже популярному способі модифікації зображень, а саме відсіканні частини.

Збережемо до колекції оригінальне зображення, створимо обрізану копію и відправимо на перевірку. Результат можна побачити на рисунку 3.12.

Рисунок 3.12 – Результат перевірки обрізаного зображення

Отриманий результат підтверджує ефективність обраних методів перевірки зображень. Також, можна сказати що програма була створена без помилок і готова до використання.

3.4 Аналіз результатів роботи застосунку по запропонованому алгоритму

3.4.1 Створення дата сету

Для аналізу результатів роботи, спочатку треба створити датасет, на якому буде відбуватись перевірка. Для того, щоб охопити якомога більше критеріїв ефективності роботи, датасет повинен містить декілька сотень зображень різних категорій. Для формування такого датасету використовуються джерела, в яких можна знайти вже класифіковані датасети [33]. Заповнимо колекцію із 100 зображень котів, 100 зображень собак, 100 зображень квіток, 100 зображень автомобілів, та 100 зображень різних графіків, креслень, діаграм.

Приклади зображень можна побачити у додатку В.

Створена колекція містить зображення різних класів, різних фігур та дозволить виявити як розроблений алгоритм буде відкидати зображення не

відповідних класів та чи буде він робити зі схожими за виглядом графіками, які присутні у великій кількості робіт з технічних дисциплін.

3.4.2 Опис експериментів

Експерименти повинні дослідити різноманітні етапи перевірки. Опис експериментів представлений у таблиці 3.1.

Таблиця 3.1 – Опис експериментів

№	Назва	Опис
1	Відбір зображень за змістом	Завантажимо всі 500 оригінальних зображень у колекцію. Візьмемо 10 зображень із цієї колекції, по 2 зображення кожного змісту, та виконаємо пошук подібних за змістом зображень. Програма повинна не виконувати пошук в інших за змістом зображеннях та знайти оригінальні.
2	Відбір модифікованих зображень за змістом	Повторимо експеримент №1, але 10 зображень для перевірки підвергнемо дії певних геометричних перетворень, таких як зміна масштабу та відсікання частини. Програма повинна все також правильно встановити зміст та відкинути інші класи.
3	Швидкість відбору зображень за змістом	Візьмемо одне зображення із колекції, та виконаємо пошук подібних за змістом з різною кількістю зображень у колекції. Поступово збільшуючи кількість зображень різних класів, швидкість пошуку не повинна сильно збільшуватись.

Продовження таблиці 3.1

№	Назва	Опис
4	Виявлення однакових зображень	Візьмемо інші 10 зображень із колекції та виконаємо повний цикл перевірки. Очікується, що всі 10 зображень будуть знайдені із значенням подібності в 100%.
5	Виявлення зображень, відмінних масштабом	Візьмемо інші 10 зображень, та створимо копії: 50%, 75%, 150%, 200% відсотків масштабу. Сумарно, 40 зображень для перевірки. Програма повинна встановити, що всі 40 зображень є підозрілими на плагіат.
6	Виявлення зображень, відмінних відсіканням частин	Візьмемо ще 10 зображень, та створимо їх копії: 10 із невеликим відсіканням частин (звичайне видалення зайвих частин фону), та 10 з більш значними змінами. Додаток Б. Програма повинна встановити, що всі 20 зображень є підозрілими на плагіат.
7	Знаходження оригінальних зображень	Спочатку завантажимо лише половину датасету, 250 зображень. Далі завантажимо ще 250 оригінальних зображень з повною перевіркою на плагіат. Система повинна встановити, що всі зображення є унікальними.

3.4.3 Результати експериментів

3.4.3.1 Відбір зображень за змістом

Цей експеримент повинен показати, наскільки ефективно працює алгоритму відбору схожих за змістом зображень на першому кроці роботи

програми. Цей алгоритм повинен пришвидшити роботу програми, та не робити зайві перевірки. Авжеж, на цьому кроці повинно бути знайдено хоча б одне подібне зображення, бо відбувається пошук зображень, які точно є у колекції. Результати перевірки можна побачити на рисунку 3.13.

Так як для кожного зображення у колекції було 100 зображень того ж класу, наприклад для зображення кота було 100 зображень інших котів, то очікувалось, що перевірка на плагіат буде відбуватись тільки серед цих ста зображень, а не усіх п’ятисот зображень у колекції.

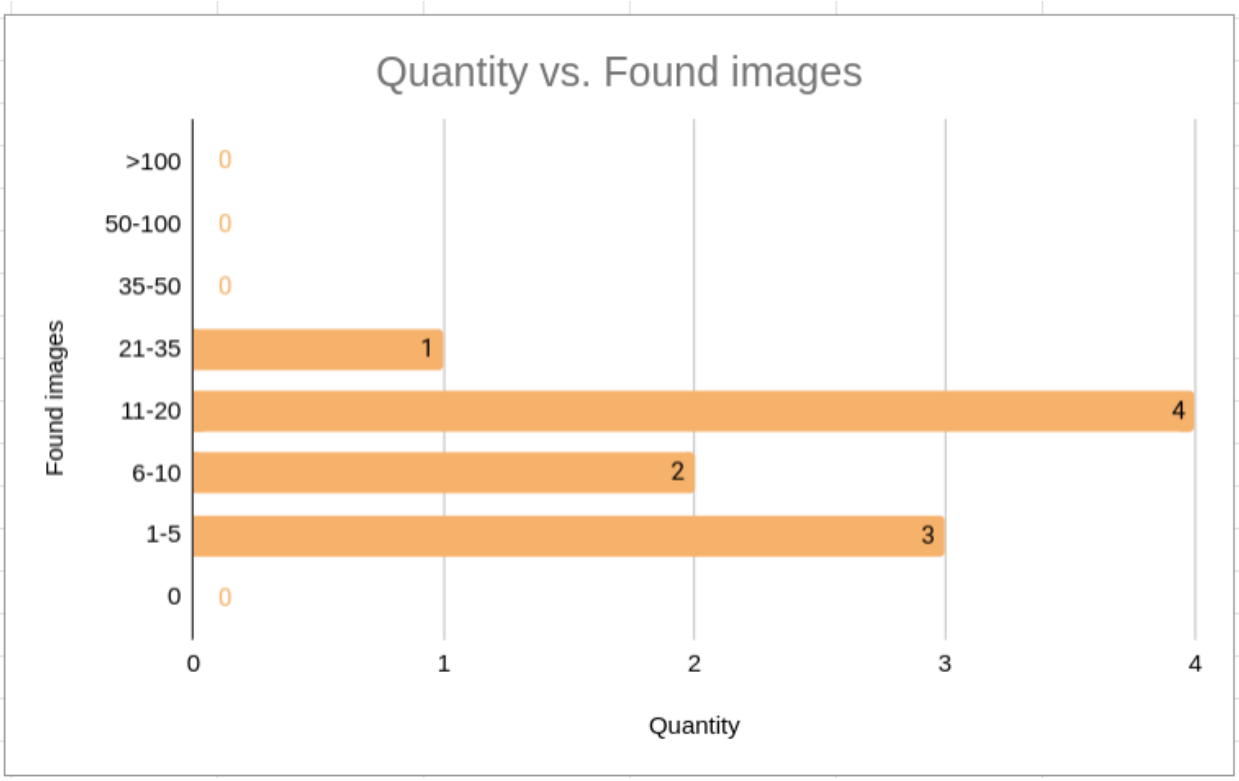

Рисунок 3.13 – Діаграма відношення кількості зображень до кількості знайдених для них подібних зображень

Як можна побачити, не було випадків, коли зображення зовсім не знайшло собі подібних, та не було випадків, коли для на плагіат перевірялись зовсім різні за змістом зображення. Навпаки, завдяки модифікації логіки та пошуку не тільки за одним тегом, а й ще за двома спільними, кількість зображень для перевірки завжди набагато менша загальної кількості

зображень навіть одного класу. Через це можна зробити висновок, що алгоритм працює правильно.

3.4.3.2 Відбір модифікованих зображень за змістом

Так як зображення можуть відрізнятись геометричними перетвореннями, то існує ризик що зображення отримає інший набір тегів. Таким чином, швидкість пошуку подібних зображень може бути зниженим, а може і зовсім не буде знайдено оригінальне зображення. Було проведено експеримент, результати якого можна побачити на рисунку 3.14.

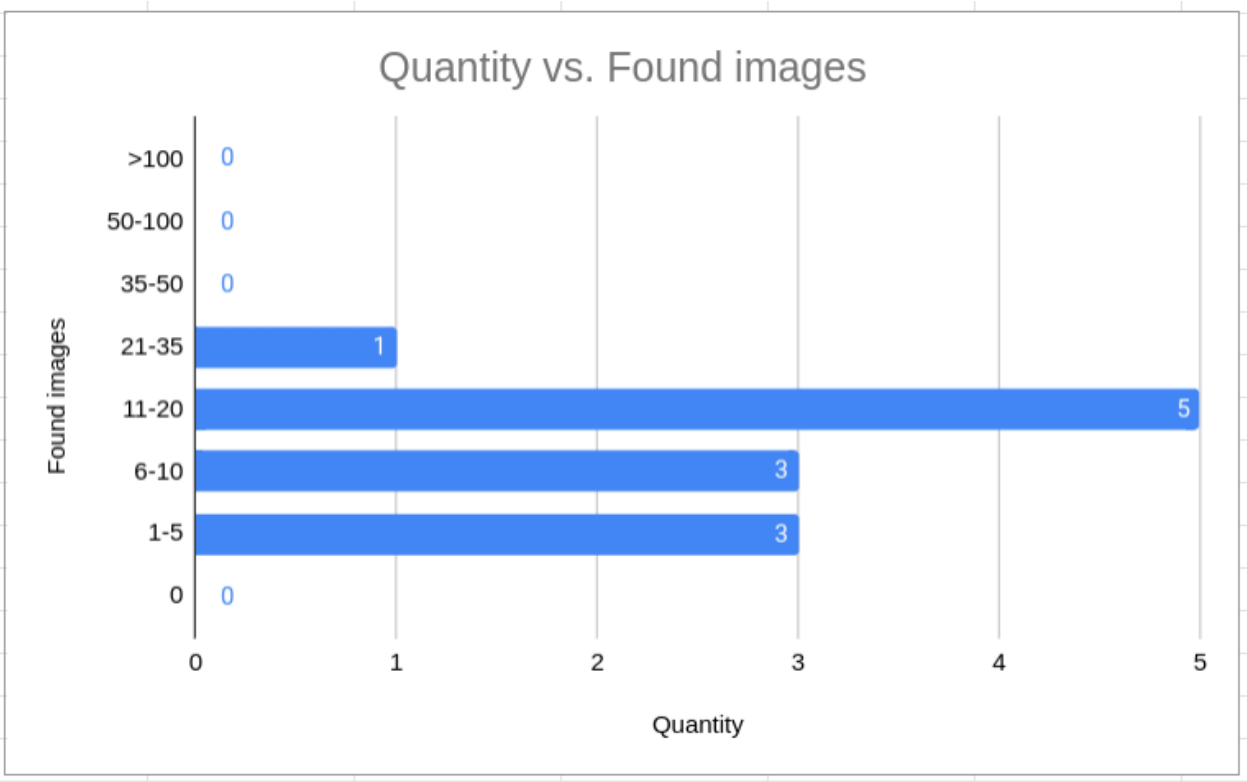

Рисунок 3.14 – Діаграма відношення кількості зображень до кількості знайдених для них подібних зображень для перетворених зображень

Аналізуючи результати, можна впевнено сказати, що великих змін в кількості знайдених подібних за змістом зображень не відбулось. Тільки в одному випадку кількість знайдених подібних зображень збільшилось, і це

випадку зменшення масштабу зображення. Таке зображення отримало трохи видозмінений набір тегів, але навіть у цьому випадку загальна кількість тегів все же значно менша за кількість зображень одного класу.

Таким чином, алгоритм пошуку зображень за змістом підтвердив свою ефективність.

3.4.3.3 Швидкість відбору зображень за змістом

Одним із головних елементів програми є пошук тільки відповідних зображень. Вже було встановлено, що відбираються тільки схожі за змістом зображення, та їх кількість зазвичай не більше 50% відсотків зображених одного класу. Але необхідно перевірити, наскільки пошук буде виконуватись довше із збільшенням кількості зображень у колекції. Результати експерименту представлені на рисунку 3.15.

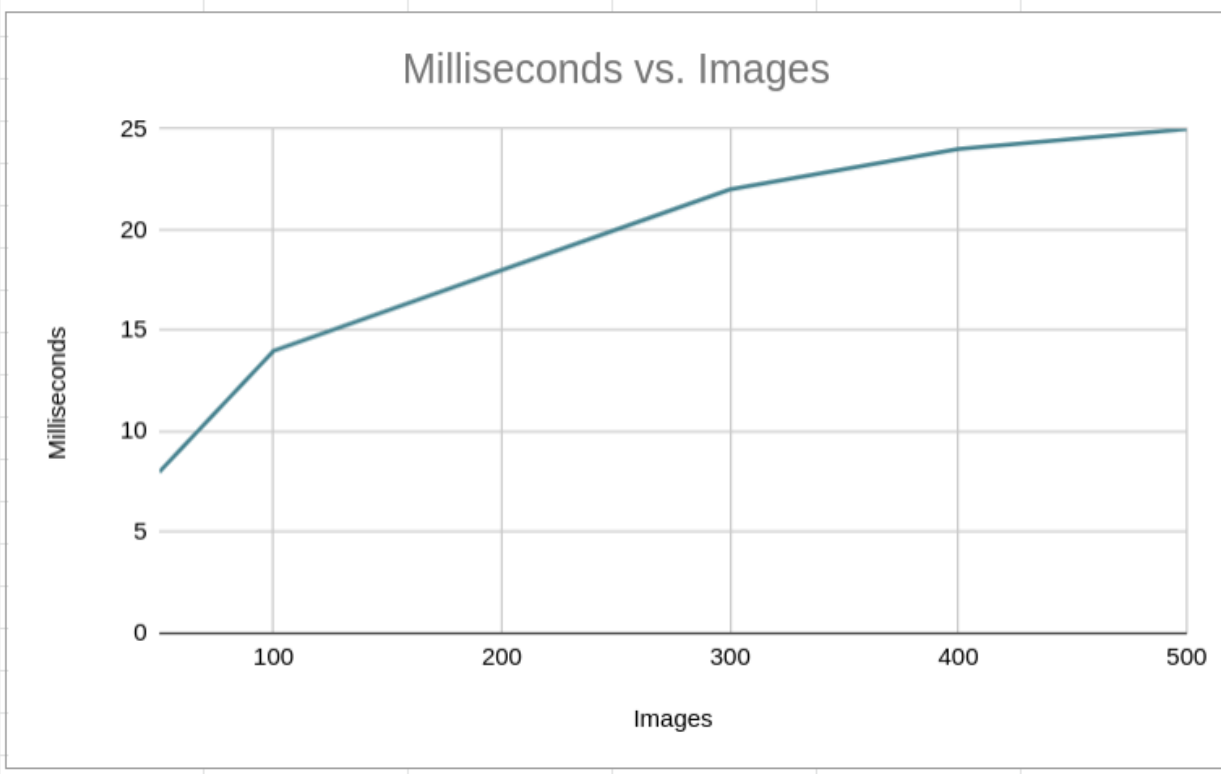

Рисунок 3.15 – Залежність швидкості відбору подібних зображень

Для більшої точності цей експеримент був повторений 5 разів для різних зображень, та на графіку представлені вже середні значення. Можна побачити, що дійсно, зі збільшенням кількості зображень і збільшується час на пошук подібних зображень. Але цей час вимірюється у мілісекундах, через створені індекси по кожному з тегів. Таким чином, навіть поповнюючи колекцію різноманітними зображенням з іншими тегами, це мало впливає на пошук. На графіку добре видно, що збільшивши кількість зображень в десять разів, витрачений час на пошук збільшився лише в три рази. Це підтверджує ефективність використання зазначеного підходу.

3.4.3.4 Виявлення однакових зображень

У цьому експерименті необхідно перевірити виявлення плагіату у випадках, коли зображення, що перевіряються, ідентичні зображенням у колекції. Використовуємо 10 зображень із різних класів, та повну колекцію із 500 зображень. Результат перевірки зображено на рисунку 3.16.

Було знайдено всі 10 зображень, та значення подібності для кожного з них визначено як 100%, що є повністю коректним результатом роботи програми.

Цей експеримент надає підставу зробити висновок про правильність відбору зображень при пошуку за змістом, так і про алгоритм порівняння зображень у цілому.

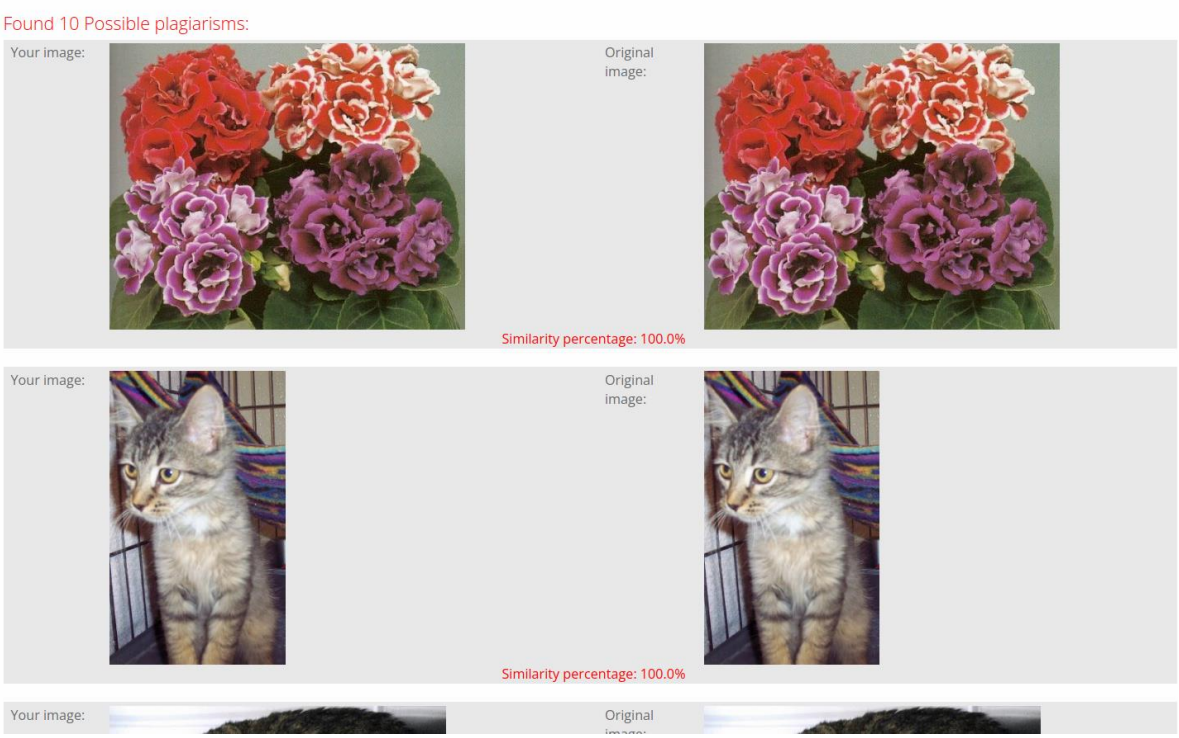

Рисунок 3.16 – Результати пошуку однакових зображень

3.4.3.5 Виявлення зображень, відмінних масштабом

Даний експеримент є дуже суттєвим, так як зображення досить часто можуть бути змінені у масштабі під час спроб плагіату. Експеримент проведено на 40 зображеннях відмінних масштабом, а результати можна побачити на рисунку 3.17.

Експеримент наглядно ілюструє головний недолік створеної системи. При роботі із зображеннями маленького масштабу, кількість характерних точок зменшується, кількість відповідних знайдених пар також зменшується, через що відбуваються випадки не знаходження плагіату зображень.

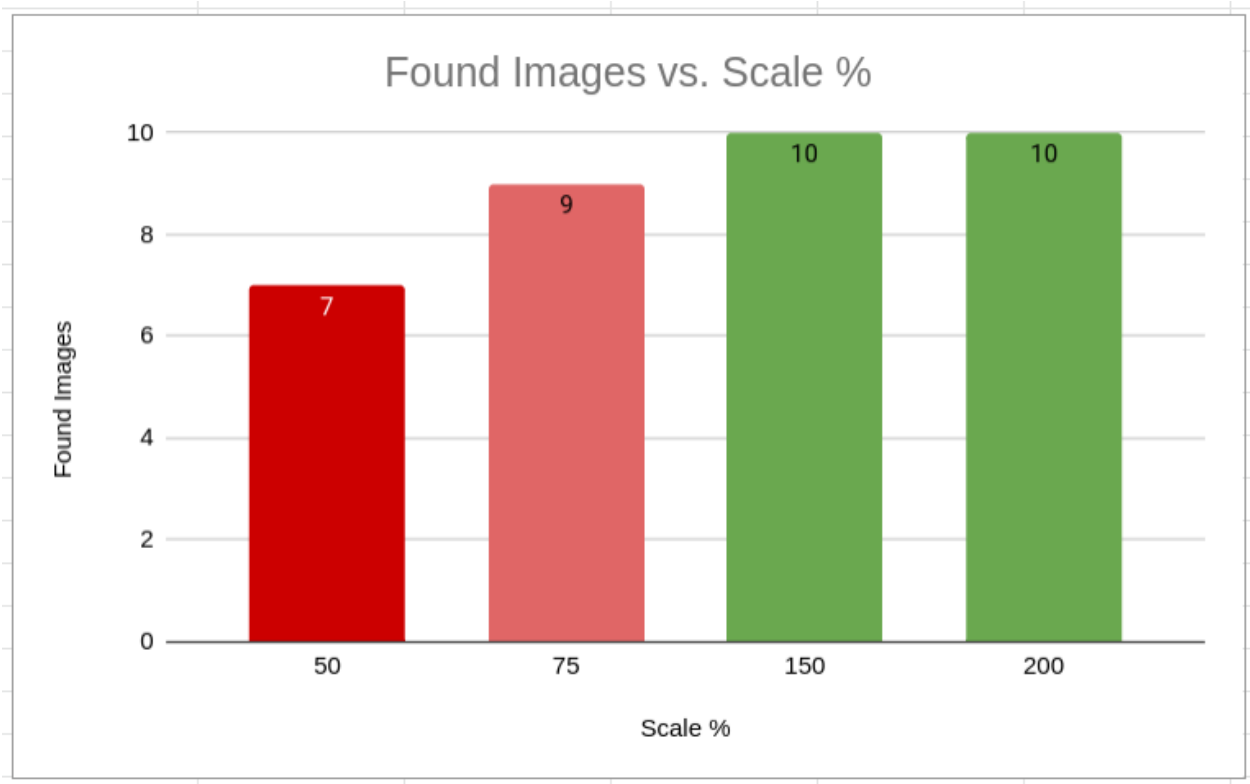

Рисунок 3.17 – Діаграма відношення кількості знайдених зображень до відсотку масштабу

Також, це залежить від оригінального розміру зображення. Не були знайдені саме ті зображення, які в оригінальному вигляді мали розмір 100×100 пікселів (рис. 3.18).

(а) (б)

Рисунок 3.18 – Приклад оригінального та видозміненого зображення: (а) оригінальне; (б) видозмінене

В той же час, всі зображення зі збільшеним масштабом були з легкістю знайдені та виявлені як ті, що є підозрілими на плагіат. Таким чином, були виявлено слабку сторону систему, та підтверджено, що зображення з достатньою кількістю ознак будуть знаходитись без суттєвих проблем.

3.4.3.6 Виявлення зображень, відмінних відсіканням частин

Ще одним популярним методом плагіату є відсікання частини зображення. Для перевірки результату виявлення плагіату зображень, що видозмінені таким метод, візьмемо 20 копій зображень та зробимо різну ступінь відсікання частин зображення. Отримані копії відправимо на перевірку у систему. Результати можна побачити на рисунку 3.19.

Можна побачити, що якщо відсікати лише фон зображення, то всі зображення знаходяться без перешкод. Але, якщо зображення втрачає суттєву кількість характерних точок (рис. 3.20), то оригінальне зображення може бути не знайдене.

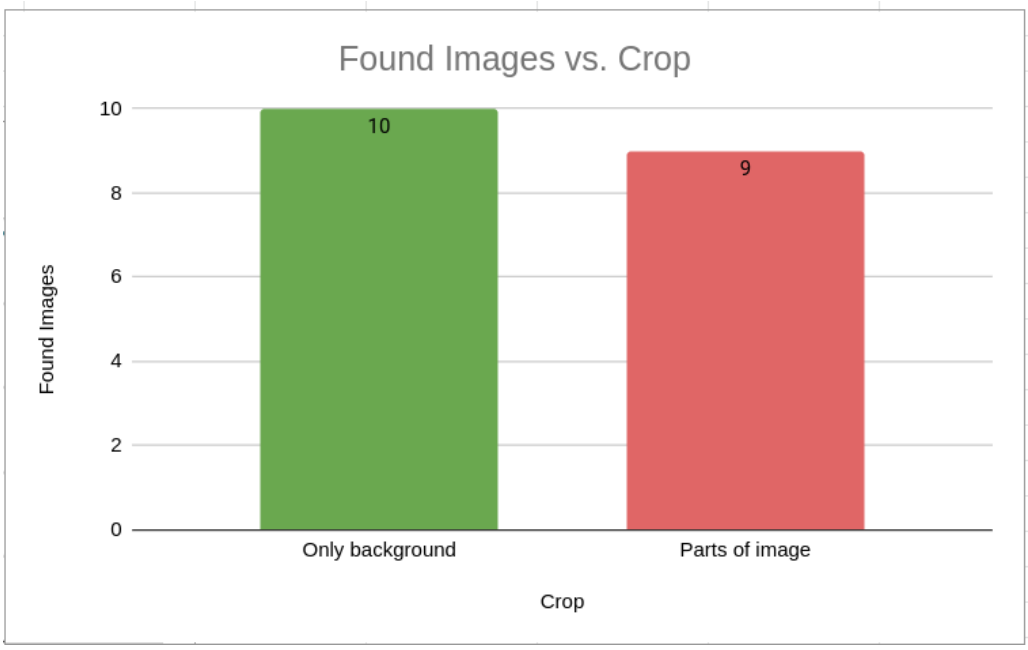

Рисунок 3.19 – Діаграма відношення кількості знайдених зображень до виду відсікання частин зображень

У цілому можна вважати, що система добре справляється з випадками відсікання частин та зможе вказати зображення, що є підозрілими на плагіат.

(а) (б)

Рисунок 3.20 – Приклад відсікання частини зображення, який не змогла встановити система:
(а) оригінал; (б) видозмінене зображення

3.4.3.7 Знаходження оригінальних зображень

Залишається перевірити на великій вибірці, чи буде знайдений плагіат у випадках, коли він не повинен бути знайденим. А саме, коли до колекції додається ще 250 оригінальних зображень (рис. 3.21).

Серед усіх зображень не було знайдено плагіату. Більшість зображень не знайшли собі відповідних пар, і тільки для однієї пари зображень була встановлена відповідність у ~32%. Всі інші створені пари набрали менший відсоток подібності. А так як для прийняття рішення, про підозру на плагіат необхідно, щоб відсоток подібності був більший за 70%, то можна зробити висновок, що в цьому експерименті точність роботи програми дорівнює 100%.

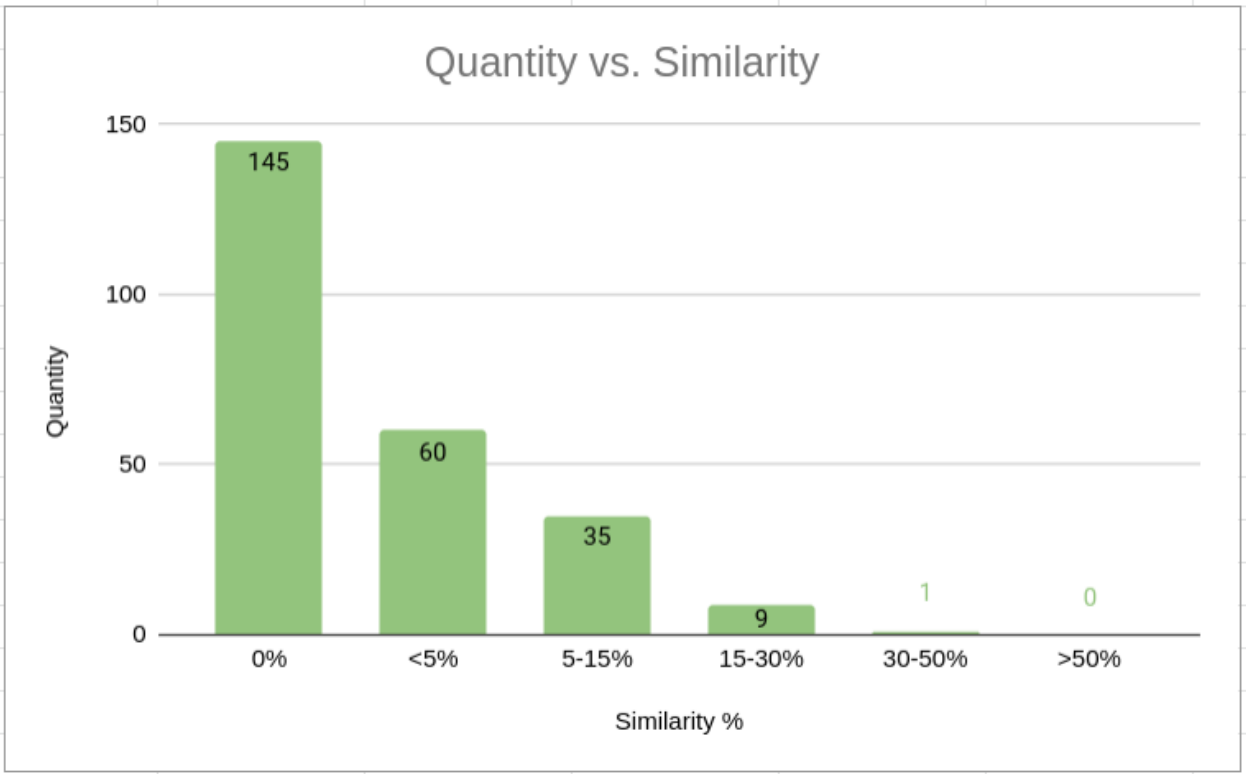

Рисунок 3.21 – Діаграма відношення кількості зображень до встановленого для них відсотку подібності для оригінальних зображень

Проведені експерименти доводять доцільність використання запропонованого алгоритму, хоча й існують випадку, коли систему може помилитись.

3.4.4 Висновки щодо досліджень

Для того, щоб чітко встановити ефективність розробленого методу, надамо оцінки за трьома метриками. Використаємо дані з експериментів, описаних вище.

Перша метрика це *Accuracy*. Ця метрика показує частку зображень, для яких система прийняла правильне рішення. Визначається:

$$Accurace = \frac{P}{N}, \tag{3.1}$$

де P – кількість зображень для яких система прийняла правильне рішення;

N – розмір вибірки.

У нашому випадку, розмір вибірки це сумарна кількість зображень, яка перевірялась у ході експериментів. Будемо використовувати тільки ті експерименти, які оцінювали повний цикл роботи програми, тобто N = 320.

Серед цих зображень для 315 було встановлено правильний результат.

Тобто *Accuracy* = 98,4%

Розглянемо наступну метрику – *Precision*. Це частка зображень, що дійсно належать даному класу, щодо всіх документів які система віднесла до цього класу. Разом з тим, розглянемо ще одну метрику – *Recall* – це частка знайдених системою зображень, що належать класу, щодо всіх документів цього класу у вибірці.

Для обчислення даних метрик використовується такі показники:

– TP – кількість істинно-позитивних рішень;

– TN – кількість істинно-негативних рішень;

– FP – кількість хибно-позитивних рішень;

– FN – кількість хибно-негативних рішень.

Precision розраховується:

$$Precission = \frac{TP}{TP + FP} \ . \qquad (3.2)$$

У нашому випадку, істинно-позитивним рішенням (TP) буде рішення, що зображення є плагіатом, коли воно дійсно є плагіатом (на основі нашої експертної думки). Рішення буде вважатись хибно-позитивним (FP) у випадку, коли зображенням є унікальним, а система вважає, що це плагіат. Істинно-негативне рішення (TN) – це рішення, коли система правильно вважає, що зображення є оригінальним, і воно дійсно є оригінальним. Хибно-негативним рішенням (FN) є рішення, коли зображення є плагіатом, а система вважає, що воно унікальне.

Для розрахунку *Recall*, використовується:

$$Recall = \frac{TP}{TP + FN}. \tag{3.3}$$

Заповнюємо таблицю 3.2 відповідно до результатів, отриманих у експериментах.

Таблиця 3.2 – Проміжні оцінки роботи системи

N	**P**	**TP**	**TN**	**FP**	**FN**
320	315	70	250	0	5

Розрахуємо метрики *Precision* та *Recall*. Результати всіх трьох метрик представлені у таблиці 3.3.

Таблиця 3.3 – Оцінка роботи системи

Accuracy	**Precision**	**Recall**
98,4%	100%	93,3%

Отримані результати показують, що розроблений метод є ефективним та доцільним у використанні для пошуку зображень, що є підозрілими на плагіат. Оригінальні зображення не будуть хибно визначені, як подібні, а повністю ідентичні зображення завжди будуть знайдені. В той же час, зображення, що відрізняються геометричними перетвореннями, з більшою ймовірністю будуть знайдені, але існують випадки, такі як відсікання великої кількості суттєвої інформації зображення або дуже сильне зменшення оригінального зображення, в яких система може не побачити відповідностей з оригіналом.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений та досліджений метод пошуку плагіату зображень у текстових файлах. У процесі його створення були вирішені такі задачі:

– проведення аналізу сучасних методів виявлення плагіату зображень;

– реалізація алгоритму знаходження змісту зображення;

– реалізація алгоритму пошуку характерних точок зображення;

– створення алгоритму зручного збереження знайденої інформації у базі даних;

– розроблення алгоритму пошуку подібних зображень серед збережених у базі даних;

– створення правил оцінки подібності зображень для прийняття рішення про підозру на плагіат;

– розроблення вебзастосунку з можливістю виконувати перевірку зображень на плагіат.

Дослідження показало, що розроблений метод та обраний підхід мають велику ефективність у пошуку зображень, що є підозрілими на плагіат, навіть у випадках виконання певних геометричних перетворень над зображеннями.

Результати досліджень були апробовані у вигляді 4-х тез доповідей на 12-тій Міжнародній науково-практичної конференції «Free and open source software» [34], 25-ому Міжнародному молодіжному форуму «Радіоелектроніка і молодь в XXI столітті» [35], VII та VIII міжнародних науково-технічних конференціях НТУ «ХПІ» «Інформатика, управління та штучний інтелект» [36, 37].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

2. Sevilla, A. (2021). Best plagiarism checkers of 2021. TechRadar. URL: https://www.techradar.com/best/plagiarism-checkers (дата звернення: 02.11.2021).

3. TinEye reverse image search. URL: https://tineye.com/ (дата звернення: 02.11.2021).

4. Plaghunter: Image Plagiarism Checker. URL: https://www.plaghunter.com/en/ (дата звернення: 02.11.2021).

5. Reverse Image Search – Find Similar Images. URL: https://www.duplichecker.com/reverse-image-search.php (дата звернення: 02.11.2021)

6. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

7. Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.

8. Ayyadevara, V. K., & Reddy, Y. (2020). Modern Computer Vision with PyTorch: Explore deep learning concepts and implement over 50 real-world image applications. Packt Publishing Ltd.

9. Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA: Determination press.

10. Van Rossum, G. (2007, June). Python Programming Language. In USENIX annual technical conference (Vol. 41, p. 36).

11. Vincent, W. S. (2020). Django for Beginners: Build websites with Python and Django. WelcomeToCode.

12. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. « O'Reilly Media, Inc.».

13. ImageAI. ImageAI 2.0.3. URL: http://www.imageai.org/ (дата звернення: 02.11.2021).

14. Гороховатський, В. О., Пупченко, Д. В., & Солодченко, К. Г. (2018). Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення.

15. Wu, Z., Shen, C., & Van Den Hengel, A. (2019). Wider or deeper: Revisiting the resnet model for visual recognition. Pattern Recognition, 90, 119-133.

16. Kovtunenko, A., Yakovleva, O., Liubchenko, V., & Yanholenko, O. (2020). ДОСЛІДЖЕННЯ СУМІСНОГО ВИКОРИСТАННЯ МАТЕМАТИЧНОЇ МОРФОЛОГІЇ ТА ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ЦІННИКІВ. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, (1 (3)), 24-31.

17. Alcantarilla, P. F., & Solutions, T. (2011). Fast explicit diffusion for accelerated features in nonlinear scale spaces. IEEE Trans. Patt. Anal. Mach. Intell, 34(7), 1281-1298.

18. 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. 4(4), 89-101.

19. Siddig, A., Guo, Z., Zhou, Z., & Wu, B. (2018). An image denoising model based on a fourth-order nonlinear partial differential equation. Computers & Mathematics with Applications, 76(5), 1056-1074.

20. Alcantarilla, P. F., Bartoli, A., & Davison, A. J. (2012, October). KAZE features. In European conference on computer vision (pp. 214-227). Springer, Berlin, Heidelberg.

21. Weickert, J., & Scharr, H. (2002). A scheme for coherence-enhancing diffusion filtering with optimized rotation invariance. Journal of Visual Communication and Image Representation, 13(1-2), 103-118.

22. Любченко, В. А., Яковлева, Е. В., & Передрий, Е. О. (2008). Нормализация перспективных преобразований проєктивно искаженных изображений. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, (26), 174-184.

23. Feature Matching. (n.d.). URL: https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html (дата звернення: 02.11.2021).

24. García, G. B., Suarez, O. D., Aranda, J. L. E., Tercero, J. S., Gracia, I. S., & Enano, N. V. (2015). Learning image processing with OpenCV. Birmingham: Packt Publishing.

25. Fan, R., Wang, H., Cai, P., Wu, J., Bocus, J., Qiao, L., & Liu, M. (2021). Learning collision-free space detection from stereo images: Homography matrix brings better data augmentation. IEEE/ASME Transactions on Mechatronics.

26. Derpanis, K. G. (2010). Overview of the RANSAC Algorithm. Image Rochester NY, 4(1), 2-3.

27. Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

28. Forcier, J., Bissex, P., & Chun, W. J. (2008). Python web development with Django. Addison-Wesley Professional.

29. McLaughlin K. An introduction to the django orm. URL: https://opensource.com/article/17/11/django-orm (дата звернення: 03.11.2021).

30. Rubio, D. (2017). Django admin Management. In Beginning Django (pp. 495-547). Apress, Berkeley, CA.

31. Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. ACM Computing Surveys (CSUR), 51(2), 1-43.

32. Pip. PyPI. URL: https://pypi.org/project/pip/ (дата звернення: 03.11.2021).

33. Find open datasets and Machine Learning Projects. Kaggle. URL: https://www.kaggle.com/datasets?search=image (дата звернення: 03.11.2021).

34. Пилипенко П.В., Яковлева О.В. (2020). Використання Django фреймворку для створення системи пошуку зображень у текстових документах користувачів. Матеріали XII-ої міжнародної науково-практичної конференції «Free and open source software»

35. Пилипенко П.В., Яковлева О.В. (2021). Пошук подібних зображень у колекції на основі аналізу характерних точок. Матеріали 25-го міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті».

36. Пилипенко П.В., Яковлева О.В. (2020). Розробка алгоритму обчислення відсотку подібності двох зображень для використання у системі пошуку плагіату. Матеріали VII міжнародної науково-технічної конференції НТУ «ХПІ» «Інформатика, управління та штучний інтелект».

37. Пилипенко П.В., Яковлева О.В. (2021). Development of the method for searching plagiarism-suspicious images in text files. Матеріали VIII міжнародної науково-технічної конференції НТУ «ХПІ» «Інформатика, управління та штучний інтелект»..