Прикарпатський національний університет імені Василя Стефаника
Факультет математики та інформатики
Кафедра комп’ютерних наук та інформаційних систем

ВИПУСКНА МАГІСТЕРСЬКА РОБОТА
Тема «СИСТЕМА ЛОКАЛІЗАЦІЇ ОБ’ЄКТІВ НА ЗОБРАЖЕННІ»

Виконав: студент 2 курсу
гр. КНМ-1
ОР магістр
Мельник О.С.
Спеціальність: 122 Комп’ютерні науки
Науковий керівник:
к.т.н., доц. Превисокова Н.В.
Рецензент:
к.т.н., доц. Ровінський В.А.

Івано-Франківськ, 2022

ЗМІСТ

ВСТУП

Актуальність. Область застосування локалізації об’єктів на зображеннях значно зросла завдяки розвитку технологій протягом останніх років. Локалізація відіграє важливу роль для виділення об’єктів в медицині, військовій розвідці, розпізнаванні обличь та відбитків пальців, відеоспостереженні, комп’ютерному зорі та інших напрямах.

Багато технічних, наукових напрямків орієнтуються на розвиток систем, в яких інформацію подано зображенням. При обробці зображень виникає ряд технічних і технологічних проблем. З найбільшою кількістю проблем зустрічаються під час обробки і розпізнавання зображень.

У зв’язку з проблемами і розвитком технологій локалізація об’єктів на зображеннях набуває все більшого використання в різних напрямках, а програми, які дозволяють локалізувати різноманітні об’єкти використовують в собі лише окремі методи локалізації та не завжди надають користувачам можливість отримати цікавлячий їх результат. Саме для розв’язання цих проблем постає необхідність у створенні системи локалізації об’єктів на зображенні.

Об’єкт досліджень – система локалізації об’єктів на зображенні.

Предмет дослідження – методи та програмна реалізація системи локалізації об’єктів на зображенні.

Мета і завдання дослідження. Метою магістерської роботи є узагальнення та систематизація знань в напрямку цифрової обробки зображень та розробка програмної реалізації системи яка буде локалізувати об’єкти на зображеннях, шляхом використання різних алгоритмів та швидкодії їх обробки.

Для досягнення мети необхідно виконати задачі:

1) проаналізувати методи цифрової обробки зображень;
2) здійснити аналіз методів локалізації зображень;
3) обґрунтувати вибір програмних засобів для реалізації програми;
4) розробити структурну схему системи локалізації об’єктів на зображенні;

5) здійснити програмну реалізацію системи локалізації об’єктів на зображенні.

Методи дослідження. Для досягнення мети дослідження застосовувались: методи локалізації шляхом бінаризації зображення, розмивання Гауса, методи локалізації при використанні бібліотек машинного навчання PyTorch, YOLO.

Наукова новизна. Новизна роботи полягає в аналізі існуючих алгоритмів локалізації об’єктів на зображенні, та порівнянні характеристик існуючих згорткових нейронних мереж, для визначення самої точної та швидкої серед них.

Структура роботи. Робота складається з вступу, 3 розділів, висновків, 28 використаних джерел, містить 36 рисунків, 4 таблиці та 1 додаток.

1. АНАЛІЗ МЕТОДІВ ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

До основних галузей застосування цифрової обробки зображень відносяться:

– підвищення якості зображень для кращого візуального сприйняття;
– обробка зображень для їх збереження;
– локалізація об’єктів військового та медичного характеру.

Під обробкою цифрових зображень розуміють реалізацію комп’ютерних алгоритмів для вирішення завдань поставлених перед користувачем. Цифрове зображення складається з скінченного числа елементів, кожен з яких розташований в конкретному місці та приймає певне значення. Ці елементи називають пікселями.

«Зір – один з головних органів почуттів, тому зорові образи грають велику роль в людському сприйнятті. Однак, на відміну від людей, які сприймають електромагнітне випромінювання у видимому діапазоні, машинна обробка зображень охоплює практично весь електромагнітний спектр [2]». Завдяки цьому цифрова обробка зображень охоплює широкі та різноманітні області застосування.

Незважаючи на відсутність меж на шляху від обробки зображень до машинного зору їх розрізняють на процеси низького, середнього і високого рівня.

Процеси низького рівня стосуються тільки примітивних операцій:

– попередньої обробки для зменшення шуму;
– зміни контрасту або поліпшення зображень;

Характерним для низького рівня є присутність зображення на початку та наприкінці процесу.

Обробка зображень на середньому рівні охоплює такі завдання, як:

– сегментація;
– опис об'єктів і їх стиснення;
– класифікація об’єктів.

Характерним для середнього рівня є наявність зображення лише на початку процесу. Кінцевим результатом є ознаки і атрибути даного зображення, як приклад, лінії контурів, ознаки конкретних об’єктів.

Високий рівень обробки включає в себе розуміння розпізнаних об'єктів при аналізі зображення і в пізнавальних функціях, які заведено пов'язувати із комп’ютерним зором [2, с.23].

1.1 Стадії цифрової обробки зображень

З наведеного можна зробити висновок, що цифрова обробка зображень містить процеси з зображеннями на вході та виході, а також процеси, які отримують ознаки із зображень, включаючи розпізнавання об’єктів. Відносно цього всі стадії цифрової обробки були зведені в схему, зображену на рисунку 1.1 [2].

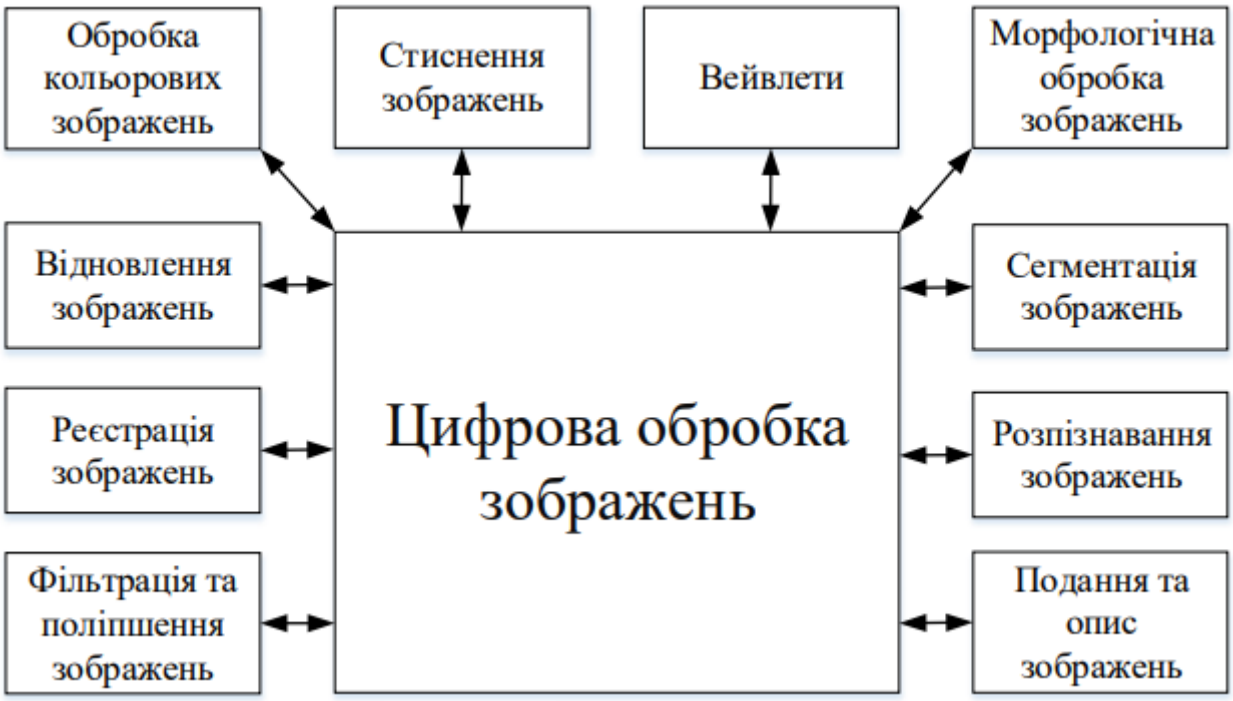

Рисунок 1.1 – Стадії цифрової обробки зображень

Обробка зображення починається з реєстрації зображення. При наявності цифрового зображення єдиним завдання реєстрації буде його масштабування. Якщо зображення немає цифрового варіанту, під час реєстрації можна зіштовхнутись з такими проблемами як: геометричне спотворення, дисперсія, надлишкова яскравість, хроматична дисторсія [8].

Поліпшення зображення – процес обробки зображення, під час якого задаються умови, спрямовані на збільшення якості зображення. Поліпшення зображень базується на тому, що людина вважає ліпшим результатом.

Відновлення зображення також пов’язано з покращенням зображення. На відміну від поліпшення використовуються математичні або імовірнісні моделі спотворення зображень [2, с.51]

У зв’язку з поширенням Інтернету обробка кольорових зображень набула важливості. Використовуються різноманітні моделі та види цифрового перетворення кольору. Також колір може бути використаний для передачі ознак зображення.

Вейвлет-перетворення утворює фундамент для подання зображень в декількох масштабах водночас. Вони застосовуються для фільтрації, аналізу стану, стискання та попередньої обробки зображень.

Стиснення використовується для зменшення об’єму пам’яті, необхідного для збереження зображення або зменшення розміру каналу, необхідного для передачі. «Майже кожен користувач знайомий з стисненням зображень, оскільки воно використовується в різних розширеннях графічних файлів [2]».

Морфологічна обробка використовується разом з інструментами для вилучення компонентів з зображень, щоб використати зазначені компоненти для подання і опису форми. Морфологічна обробки перша з стадій цифрової обробки зображень середнього рівню.

Сегментація поділяє зображення на складові частини та об’єкти. Сегментація вважається самою складною задачею цифрової обробки зображень. Занадто детальна сегментація призведе до довгого процесу вирішення завдання, вимагаючи ідентифікації об'єктів окремо. Проте недостатньо детальна сегментація призведе до помилок під кінець обробки. Чим точніша сегментація, тим більший шанс на правильне розпізнавання об’єкту [2].

Результатом сегментації є інформація про пікселі, яку треба обробити. Цим займається сегмент подання та опису. Дані передаються в вигляді кордонів областей, які відокремлюють одну область від іншої. Іншим варіантом подачі даних

є точки самих областей. При будь-якому з випадків потрібно обробити дані в форму, зручну для обробки комп’ютером. Подання у вигляді кордонів зручне при увазі на зовнішні характеристики форми, наприклад кути і вигини. Подання у вигляді областей є кращим при акцентуванні на внутрішніх властивостях об'єктів, таких як текстури або форми скелета. У деяких додатках ці уявлення доповнюють один одного [2].

Завдяки даним сегментації виконується наступна стадія обробки зображень – локалізація, яка аналізує отримані дані по різноманітним параметром і визначає чи є на зображенні об’єкти. Окрім сегментації для локалізації зображень використовують нейронні мережі, які навчають визначати об’єкти окремих категорій.

Фінальною стадією обробки цифрового зображення вважається розпізнавання, під час якого локалізованому об’єкту присвоюється ідентифікатор на підставі його опису. На рисунку 1.2 показано приклад розпізнавання.

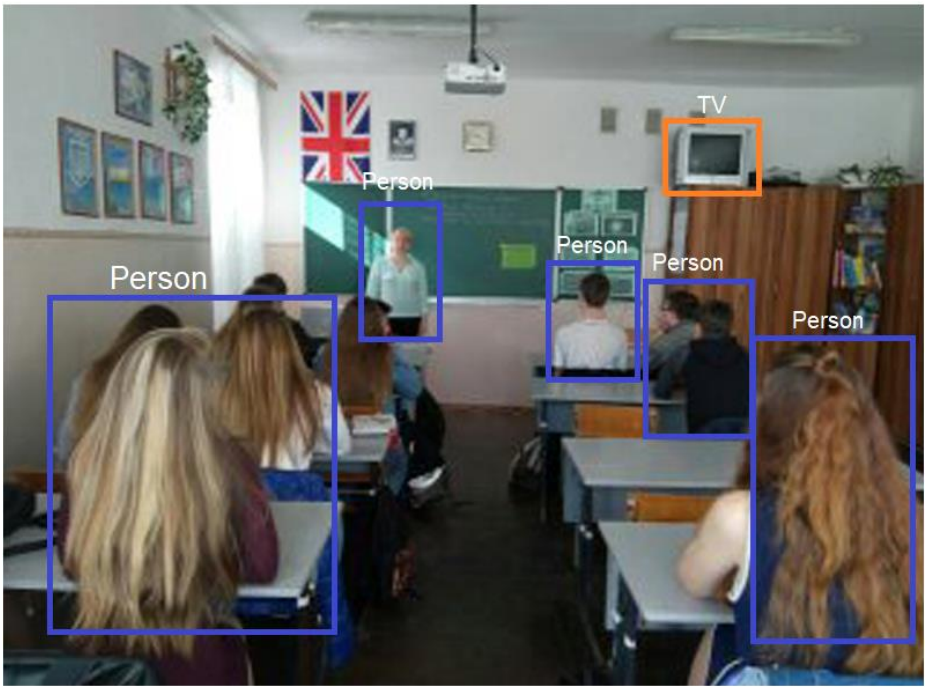

Рисунок 1.2 – Розпізнавання об’єктів на фотографії

В багатьох випадках підчас обробки зображення використовують одну або дві стадії, оскільки звичайні зображення рідко досягають обробки середнім рівнем

процесів, і зазвичай закінчуються стисненням для збереження більшої кількості зображень на носії.

1.2 Класифікація методів цифрової обробки зображень

«Наприкінці XX століття на місце аналогових методів обробки прийшли цифрові. Вони більш точні, надійніші і простіші для реалізації у порівнянні з аналоговими. Для їх використання необхідне спеціальне обладнання, таке як процесори з конвеєрною обробкою і багатопроцесорні системи [7]».

Основними задачами для практичного використання методів цифрової обробки зображень вважаються:

1. Фільтрація та покращення візуального сприйняття. З зображення намагаються прибрати шуми, зробити зображення більш контрастним та визначити контури.

2. При реставрації картин, старих фільмів та фотографій виникає інше завдання цифрової обробки – відновлення відсутніх ділянок зображень. Завдяки подібності наявних ділянок виконується відновлення і покращення всього зображення.

3. Виявлення і ідентифікація об’єктів. На зображенні потрібно визначити об’єкти, які нас цікавлять, і прокласифікувати, якщо зображення містить об’єкти різних типів. Прикладами такої задачі виступають зчитування номерів машин, пошук об’єктів, які повністю відрізняються від свого оточення, як приклад пневмонія, військова техніка на супутникових знімках.

4. Під час послідовної зйомки окремі об’єкти можуть рухатись на зображенні, завдяки чому однакові елементи можуть знаходитись в різних місцях. Ця проблема називається геометричною трансформацією зображення. Така трансформація може бути проблемою, наприклад, пацієнт почав дихати під час рентгену, що призведе до перешкод при визначенні діагнозу, або взагалі спотворенню рентгену. Або швидкість руху хмар, яка дозволить визначити швидкість вітру. Зазначені приклади потребують від нас виконання ще одної важливої задачі – поєднання елементів, виконання якої дозволить збільшити точності прогнозування.

5. Стиснення зображення. Якісні зображення займають великий обсяг і потребують високої швидкості передачі даних, у зв’язку з чим збільшуються вимоги до накопичувачів і каналів передачі.

Також за призначенням методи цифрової обробки зображень використовуються в задачах відновлення та задачах обробки в прикладних і наукових цілях [5].

Задачі для відновлення зображень:

1) кольорова корекція;
2) порівняння зображень;
3) інтерполяція і згладжування;
4) редагування та ретушування;
5) поділ зображень на області;
6) обертання і масштабування;
7) комбінування зображень;
8) компенсація втрати різкості.

Задачі в прикладних та наукових цілях:

1) розпізнавання тексту;
2) машинний зір;
3) обробка медичних зображень;
4) ідентифікація людини;
5) обробка супутникових зображень;
6) автоматичне керування машиною.

Ще одним критерієм для класифікації методів обробки зображень вважають кількість пікселів, яка використовується в одному кроці перетворення:

1) Попіксельні методи, які змінюють значення пікселя $a\ (x, y)$ в значення $b\ (x, y)$, не звертаючи уваги на сусідні пікселі.

2) Локальні методи, які для обчислення $b\ (x, y)$ використовують значення пікселів навколо $a\ (x, y)$.

3) Глобальні методи визначають $b\ (x, y)$ на основі всіх значень зображення $A\ (x, y)$.

1.3 Класифікація просторових методів цифрової корекції зображень

Підходи щодо розв'язку задач поліпшення та відновлення структури цифрового зображення поділяють на просторові та частотні. Просторові методи базуються навколо маніпуляцій з пікселями, а частотні використовують фільтрацію сигналу за перетворенням Фур'є.

Просторова обробка застосовується, якщо єдиним джерелом викривлення є адитивний шум. Частотна ж може використовуватись для нечітких зображень з дефектами освітлення, також вона враховує шум [4].

Просторова обробка зображення описується виразом:

$$g(x,y) = T[f(x,y)] \tag{1.1}$$

За формулою (1.1) обчислено оброблене зображення $g(x,y)$, де $f(x,y)$ – початкове зображення, а T – оператор обробки.

На рисунку 1.3 вказана класифікація методів обробки в просторових областях.

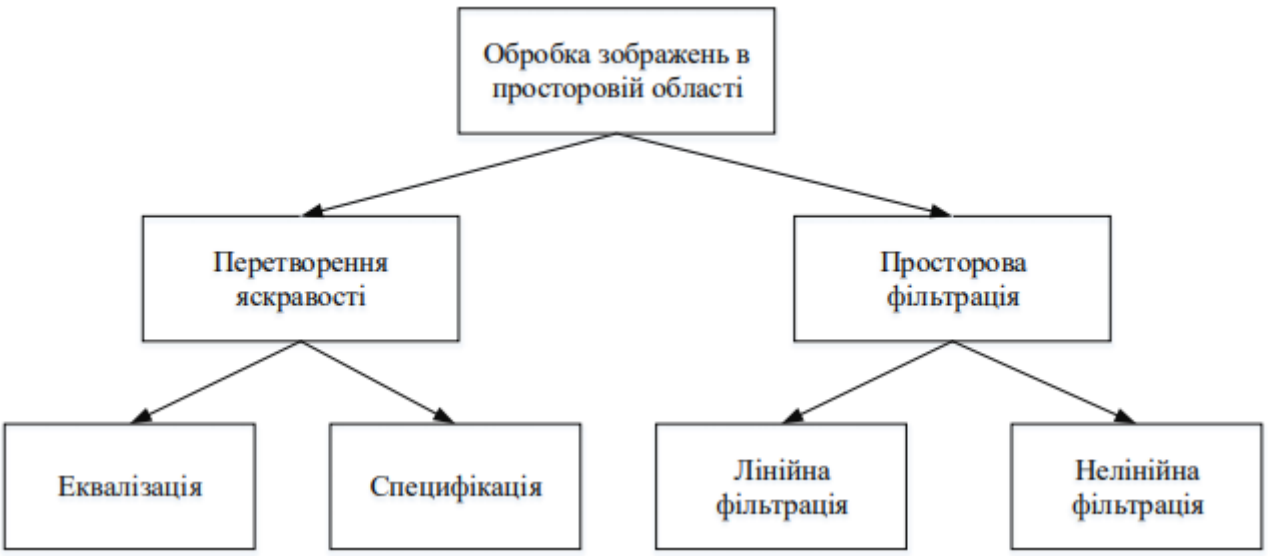

Рисунок 1.3 – Класифікація просторових методів обробки зображень

Яскравість і контраст – суб’єктивні властивості зображення, які сприймає око людини. Яскравість визначає наскільки колір пікселів відрізняється від чорного, а контраст вказує до якого кольорового діапазону відносяться пікселі. Чим більший діапазон, тим більший контраст має зображення.

Для виконання перетворення яскравості застосовуються функції аргументами яких є кольори пікселів початкового зображення. В залежності від кількості пікселів до яких буде виконано перетворення можна буде отримати більш яскраве, або більш похмуре зображення в залежності від збільшення або зменшення кольорового діапазону пікселів.

До фільтрації відносять лінійні та нелінійні фільтри, при використанні яких виконується обробка зображення відносно яскравості пікселів навколо точки.

Під час лінійних операцій для отримання добутку в позиції *(x, y)* значення в околі пікселя множиться на коефіцієнт. Якщо використана область має розмірність *m* * *n*, то коефіцієнти у вигляді матриці називають фільтром [1].

На відміну від лінійних операцій нелінійна фільтрація заснована на нелінійних функціях, які застосовуються до пікселів околу.

Найбільш використовуваними фільтрами при обробці зображень вважаються: градаційні, арифметико-логічні, згладжуючі, відновлюючі та фільтри зміни яскравості.

1.4 Аналіз методів цифрової обробки зображень

Еквалізація гістограми зображення. Метод використовується, якщо зображення однотонне. Спочатку будується гістограма яскравості, по вісі абсцис вказують значення яскравості, а по вісі ординат – число пікселів з таким значенням яскравості [7].

Алгоритм побудови гістограми:

– створюємо масив з 0, з діапазоном [0,…,255] – вісь абсцис;

– визначаємо колірні канали кожного пікселя, отримані значення мають належати до діапазону масиву – вісь ординат.

Кількість пікселів на кожному колірному каналі утворюють стовпчики шуканої нами гістограми. Далі виконується еквалізація гістограми, яка коректує яскравість окремих пікселів до середніх значень.

На рисунку 1.4 можна побачити результат виконання еквалізації гістограми. Перевагою цього методу є легка автоматизація, при якій не потрібно введення додаткових параметрів для поліпшення фотографій.

Рисунок 1.4 – Зверху: початкове зображення Odense. Знизу: зображення після обробки еквалізацією гістограми

Виділення контурів. Метод виділення контурів використовується лише при наявності шумів по всій поверхні об’єкту. Не зважаючи на назву методу, контури зазвичай і так добре видимі, а його задача полягає в зменшенні контурної різкості між об’єктами зображення.

Один з варіантів використання методу полягає в побудові контурної маски за допомогою аналізу фрагментів зображення та їх кореляції для подальшого знаходження різких змін кольору і освітлення. Також для виконання можуть використовуватись такі фільтри: інверсні, Вінера, Байєса. Їх задача полягає в пошуку аналогії між динамікою зображень і фізичними операціями.

Якщо зображення має динамічний фон, даний метод краще не використовувати, оскільки буде отримано велику кількість контурних об’єктів, які будуть накладатись один на інший [9].

Також використовується детектор контурів Кенні, який використовує чотири фільтри для виявлення направлення контурів, після чого шукає першу похідну від Гауссіана і отримує значення, сприятливі до використання.

Якщо зображення було отримано при поганому освітленні, до нього використовують алгоритм Retinex. Даний алгоритм імітує біологічні процеси ока під час розрізнення кольорів при поганому освітленні [10]. Також даний алгоритм використовують в астрономії та медицині.

Зображення виражається як добуток частот освітлення і об’єкта:

$$L(i,j) = G * I(i,j) \tag{1.2}$$

За формулою(1.2) обчислено освітленість L , за рахунок добутку фільтру Гауса G та , вхідного об’єкту I. Після визначення освітленості зображення та виконання алгоритму потрібно виконати відновлення зображення, яке виконується за формулою(1.3):

$$I'(i,j) = \sum_k \log I(i,j) - \log g_k(i,j) * I(i,j) \tag{1.3}$$

В результаті отримано числове значення обробленого зображення I', завдяки добутку значень вхідного об’єкту I з різницею логарифмів вхідного зображення, та обробленого алгоритмом g.

Результати роботи даного алгоритму можна побачити на рисунку 1.5.

Рисунок 1.5 – Результат роботи алгоритму Retinex

Фільтр різкості зображення реалізується використанням ядра згортки. Елементи зображення отримують нові значення за рахунок елементів, примикаючих до даного. Кількість примикаючих елементів визначається квадратною матрицею, розмірність якої рівна розміру ядра згортки, з центром в даному елементі.

Ядро згортки має непарний розмір вікна, щоб був лише 1 головний елемент. Фільтрація виконується під час руху вікна фільтра по зображенню. Під час цього кожен елемент отримує ваговий множник, який використовують в ваговій функції. В кожній позиції вікна функція множиться на значення сусідніх до позиції пікселів вихідного зображення і результати сумуються. Отриману суму називають відгуком фільтру і присвоюється пікселю, який має обрану позицію функції.

Також під час використання фільтру різкості зображення треба враховувати граничні пікселі. В результаті створюється зображення, де в центрі знаходиться вхідне зображення, а краї заповнюються пікселями з тимчасового зображення. Рисунок 1.6 наводить приклад застосування даного фільтру.

Рисунок 1.6 – Обробка зображення фільтром для підвищення різкості, зліва оригінальне зображення, справа оброблене фільтром

1.5 Висновок

В даному розділі магістерської кваліфікаційної роботи досліджено і описано стадії цифрової обробки зображень.

Розглянуто класифікацію методів обробки зображень. Визначено типові задачі для відтворення зображень:

1. кольорова корекція;
2. порівняння зображень;
3. інтерполяція і згладжування;
4. редагування та ретушування;
5. поділ зображень на області;
6. обертання і масштабування;
7. комбінування зображень;
8. компенсація втрати різкості.

Розглянуто класифікацію просторових методів обробки. Визначено важливість методів трансформації яскравості та фільтрації щодо просторових зображень.

Під час аналізу були розглянуті такі методи: еквалізація гістограми, виділення контурів, алгоритм Retinex, фільтр різкості.

2. МЕТОДИ ЛОКАЛІЗАЦІЇ ОБ’ЄКТІВ НА ЗОБРАЖЕННІ

Для локалізації об’єктів застосовують різні методи цифрової обробки зображень. Задача локалізації полягає в визначенні області об’єкту на зображеннях в залежності від заданих параметрів.

Результатом локалізації об’єктів є їх координати, по яким будуються границі, або фігури, для візуального сприйняття розташування об'єкта.

Методи обробки різняться в залежності від типу зображення локалізованого об'єкта (об'єкт може бути представлений точковим зображенням, групою точок, у вигляді квадрату або збільшення масштабу зображення для показу лише об’єкту) [6].

При виборі значень параметрів локалізації, виходячи з змінності умов отримання зображень, з метою виключення можливості втрати області об'єкту, значення параметрів задаються максимальним діапазоном можливих значень. Виходячи з цього, не виключена ймовірність локалізації помилкових областей об'єктів. На кожному етапі локалізації можливе використання різних параметрів об'єктів, тому методів локалізації по якомусь одному набору ознак у чистому вигляді не існує.

Рисунок 2.1 класифікує методи цифрової обробки які використовуються при локалізації об’єктів на зображеннях.

Рисунок 2.1– Класифікація методів для локалізації об’єктів.

При використанні методів кореляційного аналізу задача полягає в виявленні пікселів об’єктів, які будуть визначені якщо між їх значенням яскравості визначається стійкий кореляційний зв’язок. Кореляційні методи мають велику обчислювальну складність у зв’язку з масштабуванням шуканого об’єкту.

Аналіз гістограми полягає в побудові розподілу властивостей пікселів зображення. Завдяки гістограмі визначаються порогові значення яскравості, всі пікселі які знаходяться в межах порогів, виділяються на зображенні, як шукані області.

При роботі з методами сегментації зображення розбивають на окремі сегменти, кожен з яких містить власні характеристики, завдяки чому об’єкти відокремлюються від фону і виділяються контурами.

Суть структурних методів полягає у формуванні на основі непохідних елементів зображення складових структурних елементів об'єкта. У якості складових структур служать елементи зображення об'єкта, складені з відрізків прямих ліній: проекції прямокутних паралелепіпедів на площину (у простішому випадку шукають також паралельні лінії, прямокутники і паралелограми), перебування яких необхідно, наприклад, для виявлення будівель і доріг на зображеннях [3].

2.1 Бінаризація зображень

Як було сказано раніше, сегментація – частина середнього рівня процесу цифрової обробки, завдання якої полягає в поділі зображення на складові його області або об'єкти. Ступінь деталізації, до якої виконується такий поділ, залежить від розв'язуваної задачі, тобто, сегментацію слід припинити, коли об'єкти, що цікавлять або області виявлено. Наприклад, в задачі автоматизованого контролю складання вузлів радіоелектронної апаратури інтерес представляє аналіз зображень виробів, що виготовляються з метою виявлення певних дефектів, таких як відсутність компонентів або розрив контактних доріжок на платі. Тому не має сенсу проводити сегментацію дрібніше того рівня деталізації, який необхідний для виявлення подібних дефектів [2, с.799].

Одним з найпростіших прикладів сегментації є бінаризація. При бінаризації створюються чорні ділянки, які визначають об’єкти, і білі ділянки, які відповідають за сегменти фону. В процесі до скалярного зображення використовується один глобальний поріг:

$$J(x,y) = \begin{cases} 0 \text{ якщо } I(x,y) < T \\ 1 \text{ в протилежному випадку} \end{cases} \quad (2.1)$$

За формулою (2.1) обчислено бінарне зображення відносно глобального порогу T. Величину глобального порогу можна визначити, застосувавши стратегію оптимізації націлену на зменшення кількості артефактів.

Для локалізації об’єктів на зазначеному зображенні слід виконати пошук контурів, які будуть виявленні за рахунок позицій на зображенні де буде виконуватись перехід з чорної ділянки до світлої і навпаки (рис. 2.2).

Рисунок 2.2 – Пошук контурів на бінарному зображенні.

2.2 Розмивання Гауса та контури Кенні

Оператор усереднення Гауса вважається оптимальним для згладжування зображення. Шаблон для оператора Гауса має значення, задані співвідношенням

Гауса. Функція Гауса g в координатах x, y контролюється дисперсією σ*σ відповідно до:

$$g(x,y) = e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \quad (2.2)$$

Формула 2.2 дає спосіб обчислення коефіцієнтів для шаблону Гауса, який потім згортається разом із зображенням. Наслідки вибору шаблонів Гауса різних розмірів показані на рисунку 2.3 [17].

Рисунок 2.3 – Застосування шаблонів Гаусу різних розмірів

Дивлячись на рисунок 2.3 можна прийти до висновку, що фільтр Гауса може запропонувати покращену продуктивність, оскільки з зменшенням кількості шумів, кількість особливостей залишається незмінною [18].

Під час локалізації об’єктів саме розмиття Гауса не дозволить нам дізнатись де саме знаходяться контури об’єктів, і з цим допоможе алгоритм Кенні.

Алгоритм Кенні – оператор виявлення контурів, який використовується для виявлення широкого спектру контурів, в основі якого лежать такі цілі [17]:

1. Оптимальне виявлення без помилкових відповідей.
2. Хороша локалізація з мінімальною відстанню між виявленою та справжньою позицією контуру.
3. Наявність можливості визначення самої точної локації об’єкту серед декількох варіантів.

Алгоритм Кенні самим першим продемонстрував користь використання розмиття Гауса для пошуку границь об’єктів, оскільки завдяки йому зменшувалась кількість шумів на зображенні що і дозволило зменшити кількість помилкових відповідей.

Також заміною гауссового фільтру можуть виступати курвлети, оцінки градієнту, адаптивний фільтр [17].

2.3. Класифікація нейронних мереж в машинному навчанні.

Машинне навчання – це одна з галузей штучного інтелекту, задача якої полягає в наданні комп’ютеру здатності навчатись, щоб поступово покращити продуктивність в якійсь задачі [19].

До основних підходів навчання нейронної мережі відносять:

1. Контрольоване навчання
2. Навчання без контролю
3. Навчання з підкріпленням

До основних напрямків застосування розроблених нейронних мереж відносять [21]:

1. Класифікацію. Машина повинна згенерувати модель яка буде розподіляти небачені раніше об’єкти до різних класів. Одним з самих розповсюджених прикладів такого застосування є сортування електронних листів на 2 категорії: спам, не спам.

2. Регресію. Прогнозування на основі вибірки об’єктів з різними ознаками. На виході має вийти дійсне число. Прикладами можуть виступати: ціна квартири, вартість цінного паперу після півроку, очікуваний дохід магазину на наступний місяць, якість вина при сліпому тестуванні.

3. Кластеризація. Модель працює за тим же принципом що і при класифікації, однак групи не відомі заздалегідь, тому машина повинна буде сама визначити на які групи розподілити об’єкти.

4. Зменшення розмірності. Зведення великої кількості ознак до меншої для зручності їх подальшої візуалізації, наприклад, стиснення даних.

5. Виявлення аномалій від стандартних випадків. На практиці таким завданням є, наприклад, виявлення шахрайських дій з банківськими картами.

Завдяки розвитку технологій машинне навчання отримало окрему галузь – глибинне навчання.

До основних завдань нейронних мереж глибинного навчання відносять [19]:

1. Автоматичне розпізнавання мовлення
2. Розпізнавання зображень.
3. Пошук нових ліків
4. Системи рекомендації
5. Біоінформатику.

Завдяки швидкому розповсюдженню інтернету використання нейронних мереж для зазначених вище цілей набуло великого розвитку, оскільки збільшилась кількість галузей в яких використовуються цифрові зображення.

PyTorch це бібліотека машинного навчання з відкритим кодом, яка використовується для розробки та тренування нейронних мереж.

Більшість популярних фреймворків глибинного навчання використовують статичні обчислювальні графіки, PyTorch використовує динамічні обчислення, що забезпечує більшу гнучкість у створенні складних архітектур [20].

Pytorch використовує основні концепції Python, такі як: класи, структури та умовні цикли, завдяки чому він більш інтуїтивний для розуміння. Це робить його набагато простішим, ніж інші фреймворки, такі як TensorFlow, які привносять власний стиль програмування.

На основі PyTorch побудовано ряд застосунків для глибокого навчання [20]:

1. Tesla Autopilot
2. Uber`s Pyro
3. Hugging Face`s Transformers
4. PyTorch Lightning
5. Catalyst

2.4. Аналіз методу згорткової нейронної мережі.

Метод згорткової нейронної мережі – один з класів глибинних штучних нейронних мереж прямого поширення. В згорткових нейронних мережах виокристовують різновид перцептронів, розроблений таким чином, щоб зменшити обсяг попередьного оброблення. Основним напрямок використання даного методу є розпізнавання різних об’єктів з високою ефективністю [25].

Існує багато згорткових нейронних мереж, до найпоширеніших з них відносяться:

1. R-CNN – метод пошуку об’єктів, який працює як класифікатор зображень. На вхід мережі зображення подається розбитим на сегменти різного розміру. Кожному сегменту мережа робить передбачення, через це метод працює дуже повільно, оскільки одне зображення може проходити через огляд більше тисячі разів [26].

2. Fast R-CNN – Поліпшена версія R-CNN, яка сама визначає сегменти вхідного зображення. В усіх інших моментах не відрізняється від R-CNN, оскільки досить довго виконує передбачення.

3. SSD – переглядає зображення лише раз, а в основі використовує класифікацію VGG16, яка може визначити об’єкти 1000 різних класів, однак розмір вихідної мережі дуже великий [27].

4. YOLO. Так само як і мережа SSD не розбиває зображення на сегменти, а переглядає його лише раз, після чого одразу розробляє припущення, який саме з об’єктів бази даних підходить під зазначене припущення та визначає його [22].

Навчання згорткових нейронних мереж проходить за принципом який наведено на рисунку 2.4. На вхід подається вибірка зображень пов’язаних з об’єктами, які в майбутньому мережа має визначати. Після чого визначають ознаки об’єкту за якими в подальшому буде виконуватись класифікація об’єктів, та побудова гіпотез який саме об’єкт знаходиться на зображенні.

Рисунок 2.4 – Структурна схема навчання нейронної мережі.

Чим коректніше мережа визначає клас об’єкту тим більшим буде його кофіцієнт, однак не завжди мережі вдається коректно визначити тип об’єкту, що досліджується у зв’язку з чим може статись деградація мережі.

Для дослідження використовувалась вибірка з 1000 зображень тварин, людей, та різних об’єктів набору даних Pascal VOC2012 [28], половина з яких використовувалась для навчання нейронних мереж, а інша для порівняння швидкості та точності прогнозів.

При навчанні мережі YOLO, перші 10 ітерації давали близько 62% некоректності визначення об’єктів, оскільки база даних тільки запам’ятовувала ознаки об’єктів. Після 100 ітерації відсоток некоректності впав до 39%, і лише близько 500 ітерації мережа некоректно визначала об’єкт в 2-3% випадків.

В середньому після 500 ітерацій відсоток некоректності всіх досліджених алгоритмів складав від 1 до 5 (табл. 2.1).

Таблиця 2.1 – Відсотки некоректності мереж при навчанні.

Назва	Кількість ітерацій		
	10	100	500
30Hz DPM	79%	27%	4.9%
YOLO	62%	39%	2.4%
Fastest DPM	74%	26%	4.5%

Продовження Таблиці 2.1

R-CNN	68%	27%	3.2%
Fast R-CNN	59%	24%	1.4%
SSD512	64%	37%	2.7%

Після навчання мереж було виконано порівняння їх точності та швидкості на другій частині зображень. Згідно таблиці 2.2 видно, що підхід використаний в YOLO дозволив збільшити точність та швидкість локалізації та визначення об’єктів в 2-3 рази в порівнянні з іншими нейронними мережами. Нейронні мережі Fast R-CNN та SSD мають більшу точність прогнозу, однак програють по швидкості. Також слід зазначити, що серед нейронних мереж з якими проводилось порівняння лише одна працює в режимі реального часу, так само як і мережа YOLO.

Таблиця 2.2 – Результати порівняння швидкості локалізації різних нейронних мереж.

Назва мережі	Середня точність прогнозу	Кадри в секунду	Час обробки зображення в секундах.
30Hz DPM	25.1	30	0.503
YOLO	64.3	45	0.286
Fastest DPM	30.4	15	0.894
R-CNN	52.1	6	1.624
Fast R-CNN	71.0	0.5	3.320
SSD512	76.2	19	0.670

Після виконаного дослідження було вирішено використати в програмній реалізації системи локалізації об’єктів на зображенні нейронну мережу YOLO. До плюсів алгоритму YOLO слід зазначити:

1. Швидкість
2. Точність локалізації
3. Якісне узагальнення результатів роботи алгоритму
4. Відкритий вихідний код

YOLO використовується в таких напрямках як:

1. Медицина. В хірургії алгоритм допомагає локалізувати рух органів в режимі реального часу, або перевірити чи дійсно органи одного пацієнта підходять іншому.
2. Агрокультура. Алгоритм дозволяє роботам визначити чи дозрів той чи інший плід, що дозволяє автоматизувати процес збору помідорів та деяких інших культур.
3. Охоронне спостереження. YOLO алгоритм використовується в деяких країнах для оцінки порушення соціальної дистанції між людьми, щоб захистити людей від зараження Covid-19.
4. Безпілотні автомобілі. Інтеграція YOLO важлива для автономних транспортних засобів, оскільки вони повинні правильно визначати смуги руху та всі навколишні об’єкти та пішоходів для підвищення безпеки дорожнього руху.

Оскільки YOLO має відкритий код, його розвитком займаються досі, і з моменту створення нейронної мережі було розроблено декілька версій, які використовуються в задачах різної складності. Після їх порівняння було вирішено використати YOLOv3, який було розроблено для більш коректної локалізації об’єктів, та точності локалізації незалежно від розмірів вхідного зображення.[24].

Версії після v3 більше фокусуються на розпізнаванні об’єктів та мають більшу вхідну базу, завдяки більш розвинутим шляхам навчання мережі. Рисунок 2.5 демонструє з якою швидкістю та точністю розпізнаються об’єкти при тестуванні машинним навчанням TensorRT [23].

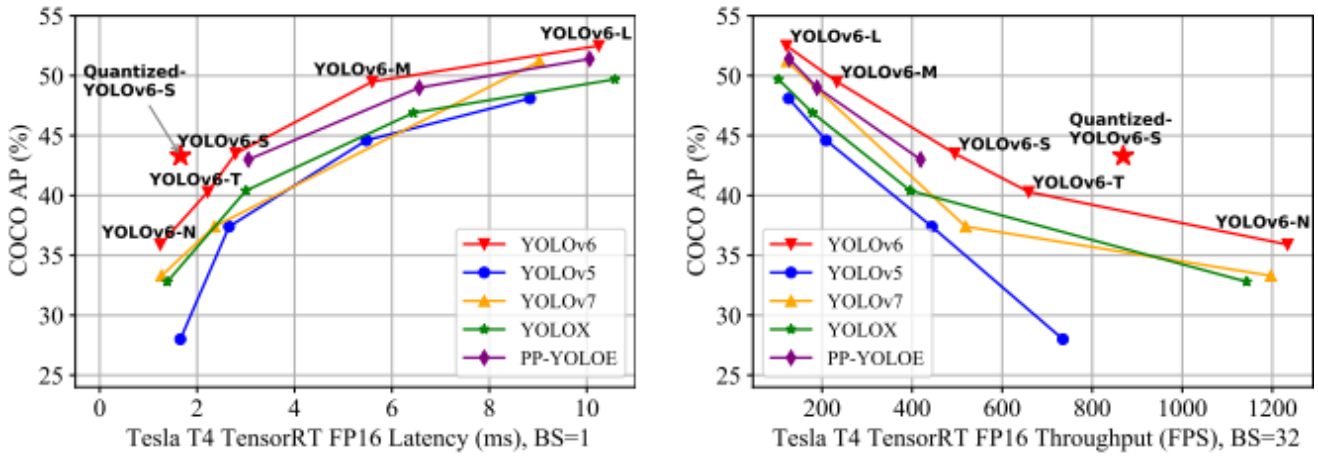

Рисунок 2.5 – Порівняння YOLOv6 з попередніми версіями за допомогою тестування з TensorRT 7(квантова модель з TensorRT 8), вісь у демонструє точність

2.5. Висновок

В даному розділі магістерської кваліфікаційної роботи розглянуто класифікацію методів локалізації об’єктів на зображенні. Визначено такі типові методи як:

– Кореляційного аналізу.
– Аналізу гістограми зображень.
– Методи сегментації.
– Структурні методи.

Розглянуто класифікацію нейронних мереж в машинному навчанні. Досліджено бібліотеку машинного навчання PyTorch. Визначено основні напрямки застосування нейронних мереж, такі як:

– Класифікацію.
– Регресію.
– Кластеризація.
– Зменшення розмірності.
– Виявлення аномалій від стандартних випадків.

Проаналізовано яким чином працюють нейронні мережі, та виконано порівняння точності локалізації та швидкодії вже існуючих згорткових нейронних мереж, таких як:

1. 30Hz DPM
2. YOLO
3. Fastest DPM
4. R-CNN
5. Fast R-CNN
6. SSD512

Після виконаних досліджень до програмної реалізації системи локалізації об’єктів на зображенні було вирішено реалізувати такі методи:

1) Бінаризація порогових значень. При виконанні локалізації даним алгоритмом будуть отримані контури, які виявляються за рахунок координат зображення, де виконується перехід з чорної ділянки до світлої і навпаки.

2) розмивання Гауса. Після оброблення зображення кількість шумів на зображенні зменшується, що дозволяє використати алгоритм Кенні для більш точного визначення границь об’єктів.

3) YOLO. Алгоритм в основі якого лежить нейронна мережа дозволяє швидко локалізувати об’єкти які відносяться до відомих мережі класів, однак взагалі не локалізує об’єкти за межами зазначених класів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЛОКАЛІЗАЦІЇ ОБ’ЄКТІВ

3.1. Програмні засоби та інструменти

Обґрунтування вибору бібліотеки OpenCV.

OpenCV (Open Source Computer Vision Library), бібліотека розроблена на мові C/C++. Вона була розроблена для вирішення задач, пов’язаних з обробкою і аналізом зображень та відео завдяки використанню комп’ютерного зору. Остання загальнодоступна версія – 4.6.0 опублікована 12.06.2022 року [11].

До переваг OpenCV слід зазначити:

– відкритий вихідний код;

– мультиплатформенність – OpenCV підтримує такі оперативні системи: Windows, Linux, Mac, Android, iOS;

– обширна документаційна база, яка підтримується від версії 3.0.0 в онлайн режимі;

– велика кількість оптимізованих алгоритмів, що прискорюють виконання задач;

– завдяки орієнтації бібліотеки на вирішення проблем обробки і аналізу зображень використання бібліотеки полегшує роботу з методами комп’ютерного зору.

Серед інших бібліотек які працюють з комп’ютерним зором слід зазначити Matlab, який можна вважати головним опонентом OpenCV. Однак на відміну від OpenCV в Matlab [12]:

– закритий вихідний код;

– підтримуються лише комп’ютерні операційні системи, що не задовольняє умов мультиплатформенності в нашому часі;

– ціна ліцензії занадто велика (в залежності від обсягу функції від 29 до 2350 доларів);

– оскільки Matlab орієнтований на роботу в математичному просторі, він не в усіх випадках може надати спектр алгоритмів які можуть знадобитись при роботі з комп'ютерним зором.

Усі методи OpenCV згруповані в модулі, основні наведено на таблиці 3.1

Таблиця 3.1 – Основні модулі OpenCV версії 4.6.0

Модуль	Опис
core	Основна функціональність
imgproc	Обробка зображень
imgcodecs	Читання та запис зображень
videoio	Читання та запис відеофайлів
highgui	Графічний інтерфейс високого рівня
video	Аналіз відео
calib3d	Калібрування камери та 3D-реконструкція
features2d	2D функціонал
objdetect	Виявлення об'єктів
dnn	Глибинне машинне навчання
ml	Машинне навчання
flann	Кластеризація та пошук у багатовимірних просторах
photo	Аналіз зображень
stitching	Зшивання зображень
gapi	швидка та портативної обробки зображень

Обґрунтування вибору мови програмування.

Python це зручна для користувача мова, з якою легко працювати, однак в порівнянні з C/C++ при роботі з Python ми втрачаємо швидкодію алгоритмів, оскільки Python працює повільніше через використання гнучкості типів даних.

Тому програмісти, які використовують OpenCV, розробили пакет OpenCV-Python, який є оболонкою для оригінальної реалізації на C/C++, що дозволяє нам писати код із інтенсивними обчисленнями на C/C++ в середині Python, з подальшим його використанням як окремих модулів Python. Завдяки цьому

швидкість алгоритмів не падає, оскільки код написаний в оригінальному коді C/C++ (оскільки це фактичний код C++, який працює у фоновому режимі).

Для розробки графічного інтерфейсу було використано бібліотеку PyQt.

Qt – інструментарій який дозволяє розробляти зручний мультиплатформенний застосунок на мові програмування C++.PyQt – варіант даного інструментарію на мові Python.

В ньому можна створити весь графічний інтерфейс програми після чого згенерувати код на цікавлячий вас мові програмування [13].

На відміну від інших бібліотек графічного інтерфейсу для Python PyQt має більш обширний функціонал і можливість налаштування стилів об’єктів що дозволяє привносити в інтерфейс новизну, яку інші графічні інтерфейси не можуть привнести у зв’язку з застарілими або фіксованими виглядами об’єктів.

3.2. Проєктування програмних засобів системи локалізації об’єктів на зображенні за методологією SADT

Методологія SADT – методологія аналізу та проектування системи.

Разом з стандартом IDEF розглядає систему на різних рівнях. IDEF0 розглядає всю систему як одне ціле. Створена контекстна діаграма для даної системи, містить 1 задачу і описує всю систему на абстрактному рівні [14].

Рисунок 3.1 демонструє контекстну діаграму системи локалізації об’єктів на зображенні.

Головною задачею системи є локалізація об’єктів (Object localization).

В ролі вхідних даних виступають зображення (Input image).

Вхідні дані в систему додає користувач (User).

Після виконання обробки методами локалізації (Localization methods), користувач може зберегти оброблене зображення (Output image).

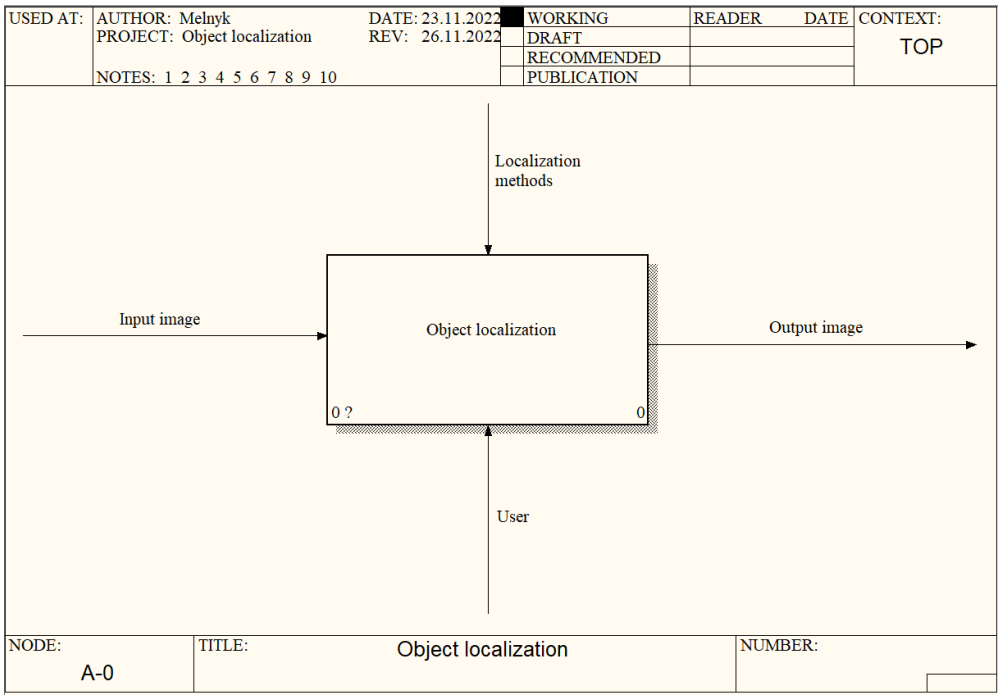

Рисунок 3.1 – Контекстна діаграма системи локалізації об’єктів

Після контекстної діаграми створюється діаграма декомпозиції, яка розбиває систему на підсистеми, які описуються окремо. Рисунок 3.2 демонструє діаграму декомпозиції.

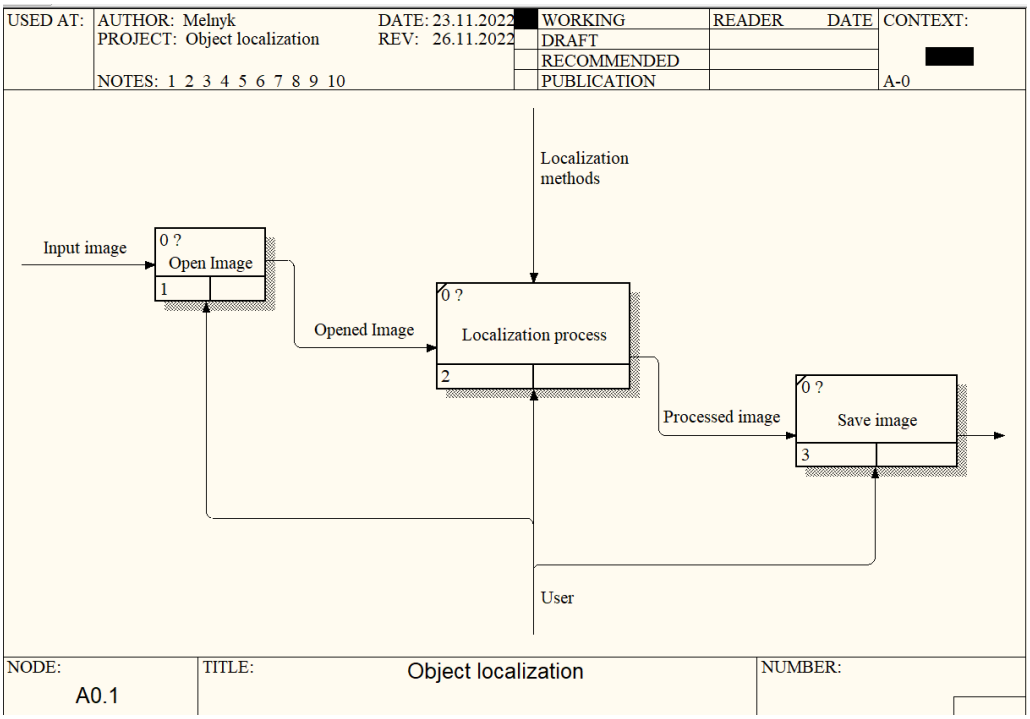

Рисунок 3.2 – Діаграма декомпозиції блоку локалізації об’єктів

Процес локалізації зображення складається з:

– відкриття зображення (Open Image);

– обробки зображення методами локалізації об’єктів (Localization process);

– збереження зображення(Save image).

Діаграма потоків даних (DFD) використовується для показу, як процеси перетворюють вхідні дані на вихідні і визначає відносини між процесами [15]. Контекстна діаграма DFD продемонстрована на рисунку 3.3. Вона складається з задачі локалізації об’єктів на зображенні і вхідних та вихідних даних, в ролі яких виступає зображення.

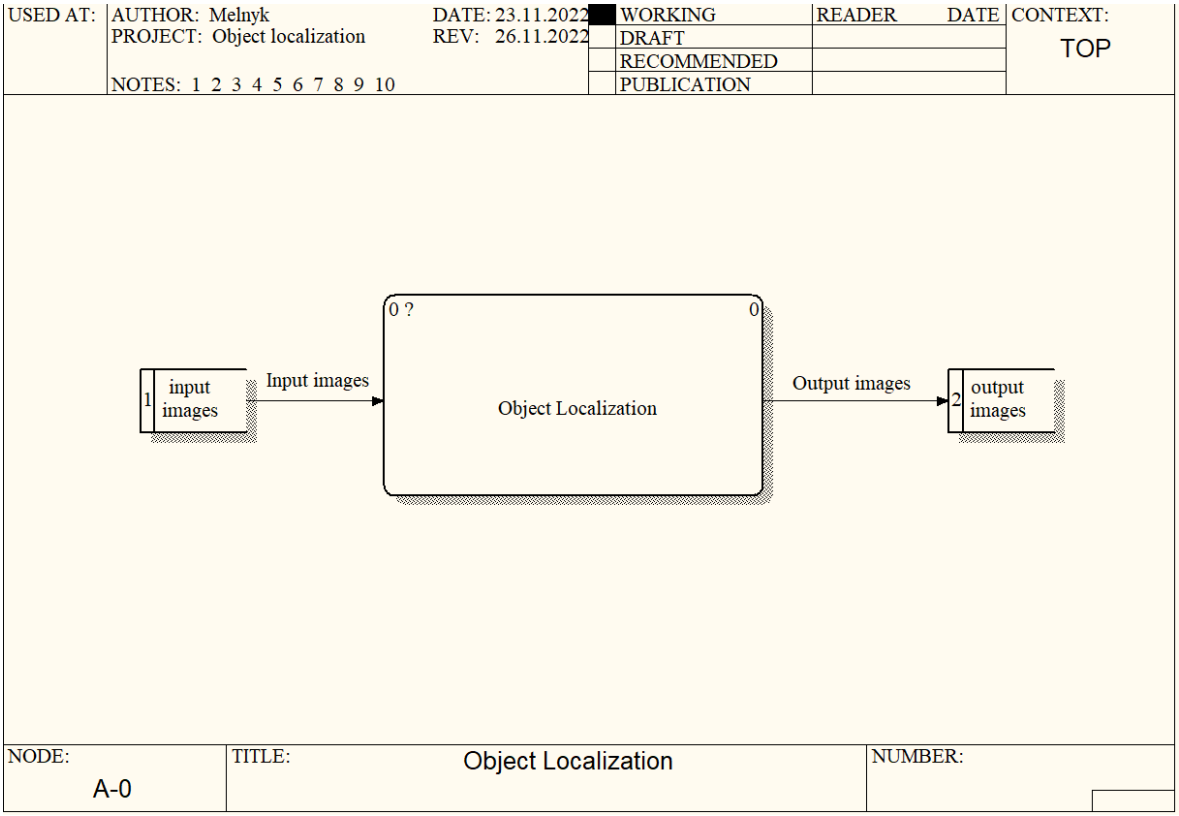

Рисунок 3.3 – DFD діаграма системи

Зміни другого рівня діаграми наведені на рисунку 3.4. Вони вказують сховища даних, в яких знаходиться зображення до, підчас та після задачі.

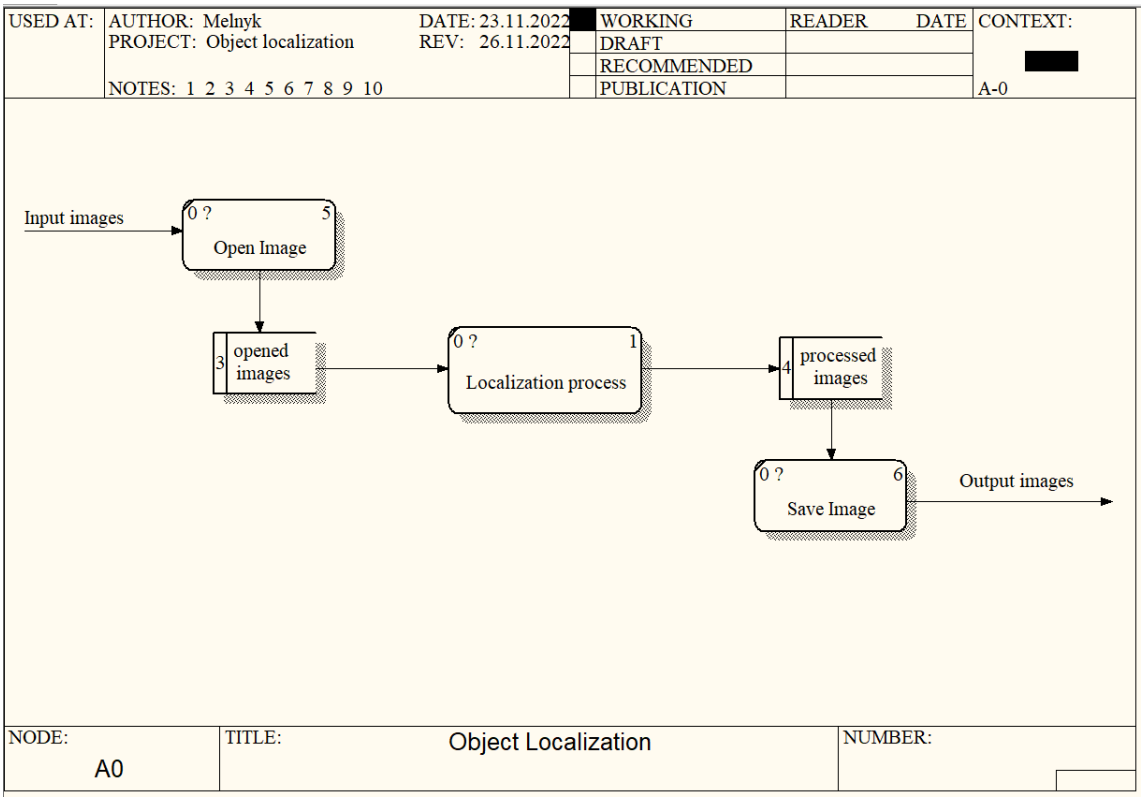

Рисунок 3.4 – DFD діаграма другого рівня

3.3. Інтерфейс

Після запуску застосунку відкривається головне вікно (рис 3.5).

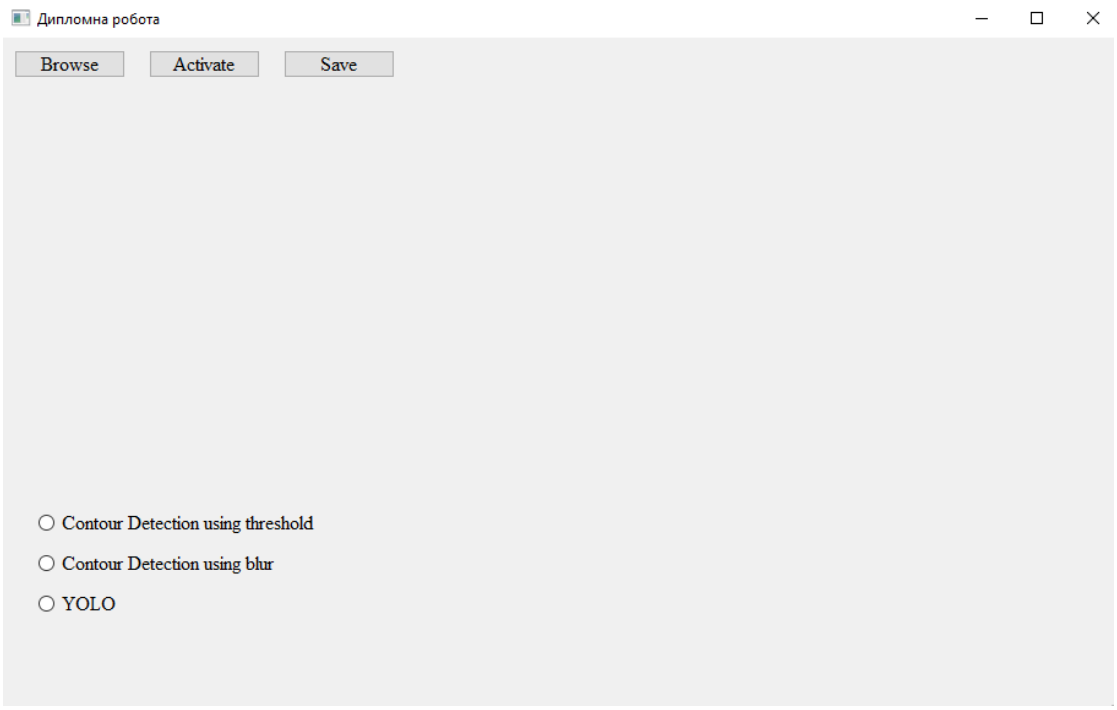

Рисунок 3.5 – Головне вікно застосунку

На головному вікні розташовані кнопки, які відіграють роль меню програми:

– «Browse» – дозволяє відкрити зображення розташоване будь де на комп’ютері, і додати його до системи;

– «Activate» – відповідає за локалізацію об’єктів на зображенні, одним з методів які розташовані, для вибору користувачем, в лівому нижньому куту;

– «Save» – зберігає файл на комп’ютері користувача.

Після натиснення кнопки «Browse» обране зображення буде додано до застосунку (рис. 3.6)

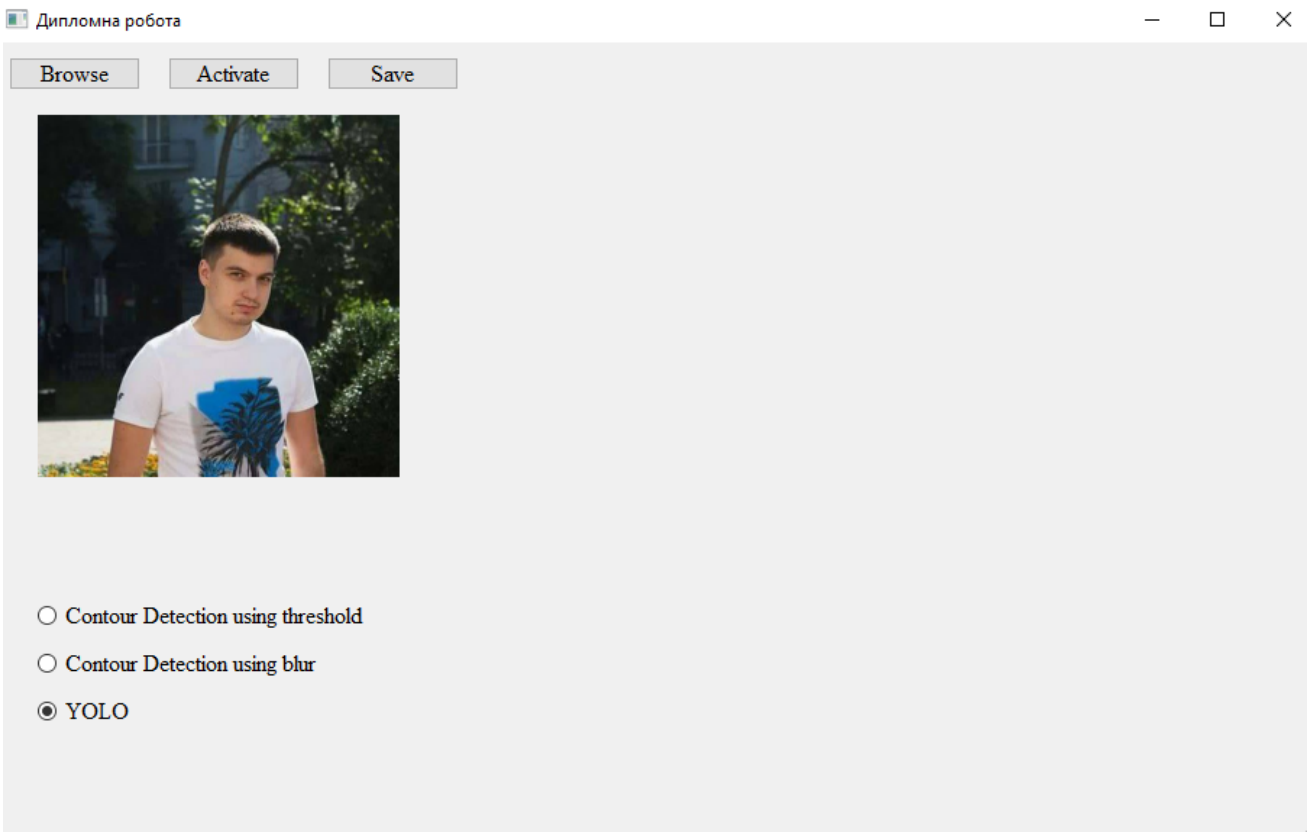

Рисунок 3.6 – Вікно застосунку з відкритим вхідним зображенням

При натисненні кнопки «Activate» користувач активує обраний метод локалізації об’єктів на зображенні. В правій частині екрану буде виведено оброблене методом локалізації зображення та кількість визначених об’єктів. Після чого натиснувши кнопку «Save» користувач може обрати куди зберегти зображення (рис 3.7).

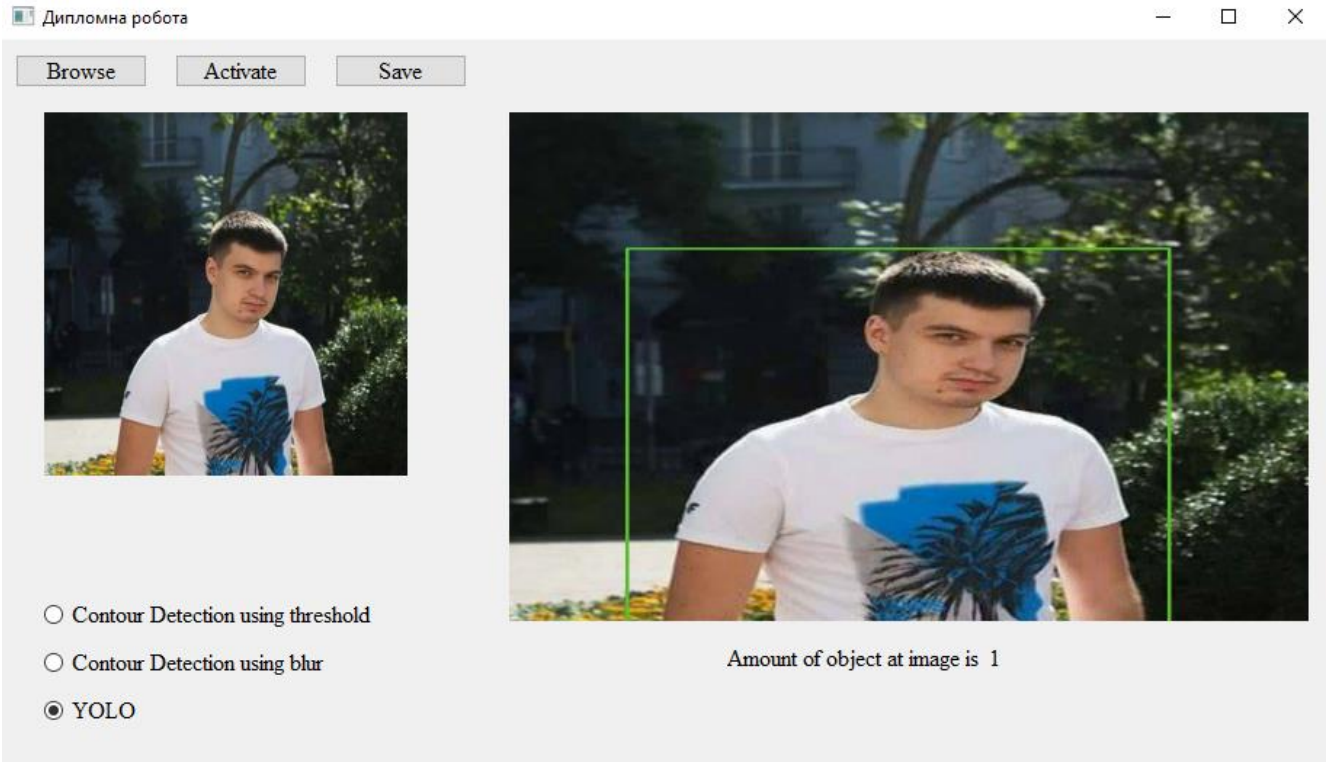

Рисунок 3.7 – Вікно застосунку з обробленим зображенням

3.4. Структурна схема системи локалізації об’єктів на зображенні

Відповідно до завдання мети розроблено структурну схему системи локалізації об’єктів на зображенні (рис.3.8).

Система складається з двох модулів:

1. Модуль відкриття/збереження зображень. Він відповідає за додавання зображення до застосунку та його збереження після обробки одним з алгоритмів локалізації які викликаються в наступному модулі.

2. Модуль локалізації об’єктів відповідає за локалізацію об’єктів на зображенні одним з алгоритмів:

1. Порогова бінаризація.

2. Розмивання Гауса;

3. Yolo.

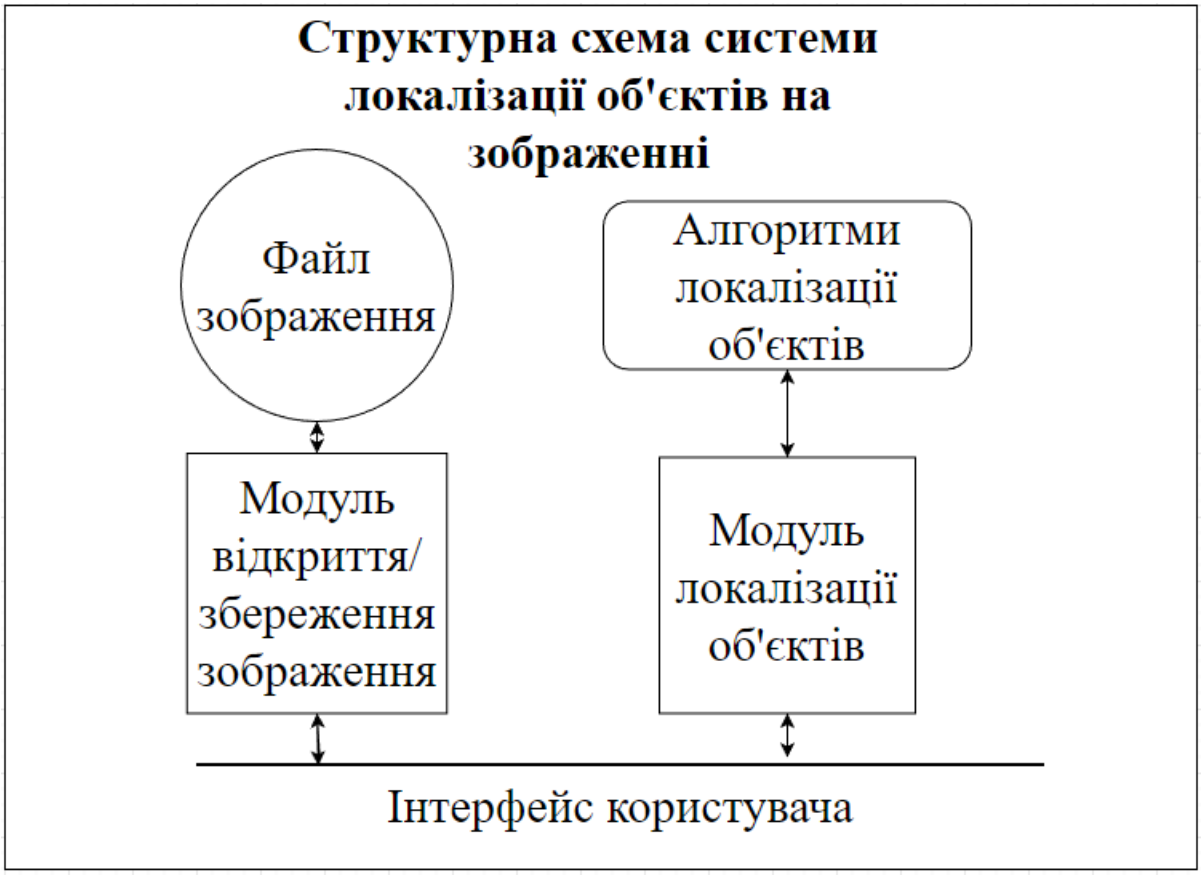

Рисунок 3.8 – Структура системи локалізації об’єктів на зображенні

Користуючись структурною схемою, розроблено діаграму класів системи. Вона надає структуру системи в термінології об’єктно-орієнтованого програмування. Діаграма класів відображає різні зв’язки між окремими об’єктами різних предметних областей, а також описує внутрішню структуру та типи відносин між ними [16].

Як показано на рисунку 3.9 система складається з класів: Main, Object_Localization, Start_Settings.

Клас Main відповідає за графічний інтерфейс, містить методи для зчитування, відображення та завантаження зображення. Містить об’єкти інших класів.

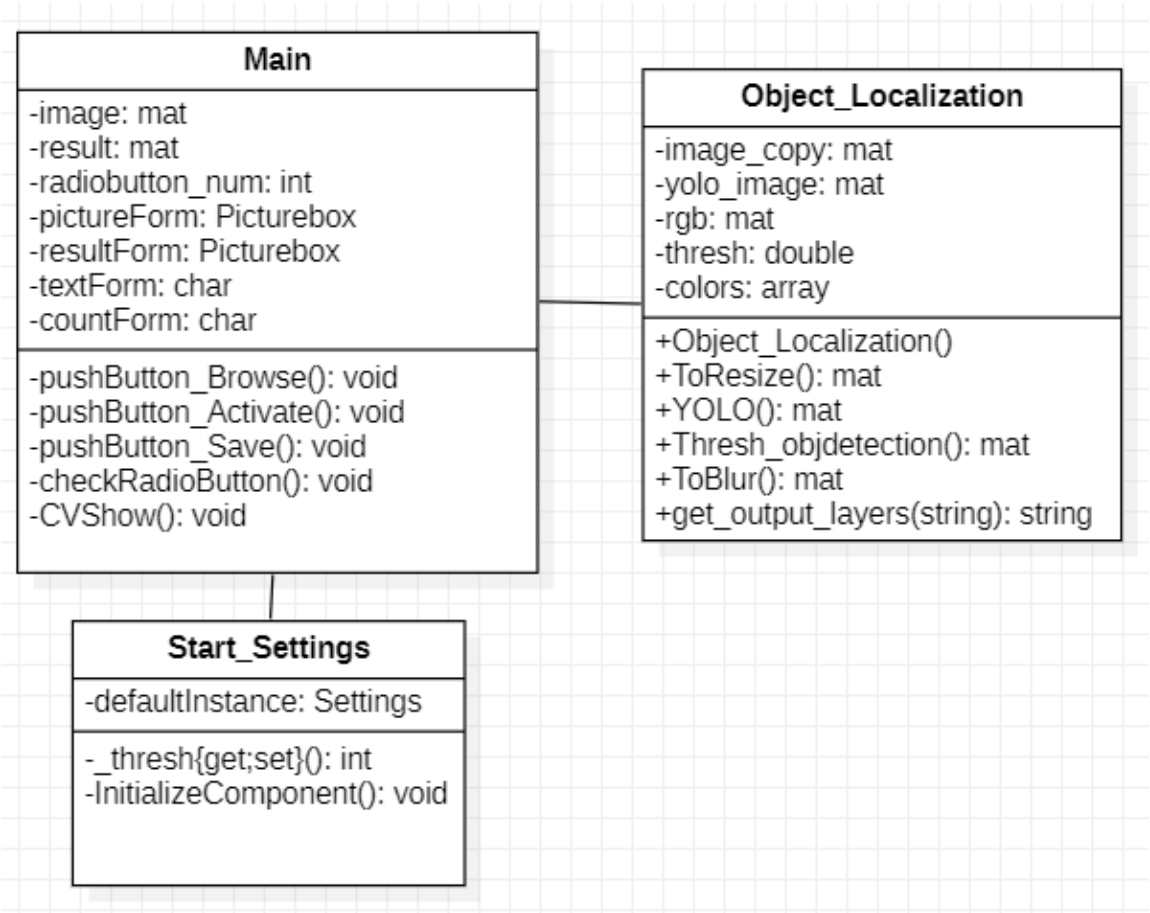

Рисунок 3.9 – UML діаграма системи локалізації об’єктів на зображенні.

Перелік членів класу Main.

Поля:

– Image – відкрите зображення;

– Result – зображення після виконання обробки;

– Radiobutton_num – визначає який з алгоритмів обраний;

– PictureForm, resultForm – поля в яких знаходиться вхідне і оброблене зображення;

– textForm, countForm – текстові поля в які записується кількість об’єктів на зображенні.

Методи:

– pushButton_Browse() – відкриття діалогово вікна для обирання зображення з комп’ютера;

– pushButton_Activate() – оброблення зображення одним з методів для локалізації об’єктів на зображенні;

– pushButton_Save() – збереження обробленого зображення;

– check_RadioButton() – визначає, який з алгоритм локалізації обраний;

– CVShow() – виведення обробленого зображення в програму.

Object_ Localization – клас в якому реалізовано методи локалізації об’єктів на зображенні.

Поля:

– Image_copy – зображення оброблене методом локалізації об’єктів використовуючи порогові значення бінаризації;

– Yolo_image – зображення оброблене алгоритмом YOLO;

– rgb – зображення оброблене за допомогою розмивання Гауса;

– thresh – порогове значення бінаризації;

– colors – масив в якому зберігаються, кольора для локалізації об’єктів алгоритмом YOLO.

Методи Object_ Localization:

– ToResize() – алгоритм який змінює розміри зображення для швидкодії алгоритмів локалізації, після чого повертає зображенню його початкові розміри;

– YOLO() – алггоритм локалізації YOLO;

– Thresh_objdetection() – алгоритм локалізації об’єктів використовуючи порогові значення бінаризації;

– ToBlur() – локалізація об’єктів за допомогою розмивання Гауса;

– Get_output_layers(string) – метод, в якому зберігаються налаштування алгоритму YOLO.

Клас Start_Settings виконує початкове розташування елементів у вікні застосунку та зберігає в собі параметр _thresh, завдяки якому можно змінити початкове порогове значення бінаризації, може приймати значення від 0 до 255.

Метод InitializeComponent() – ініціалізація елементів вікна програми.

3.5. Алгоритм роботи системи

Загальна схема алгоритму роботи системи локалізації об’єктів на зображенні наведено на рисунку 3.10. Даний алгоритм складається з 8 кроків:

1. Встановлення початкових параметрів для роботи системи.
2. Користувач завантажує зображення до застосунку.
3. Користувач обрає метод локалізації об’єктів.
4. Локалізація об’єктів на зображенні.
5. Виведення результату локалізації в правій частині вікна для перевірки користувачем якості обробки зображення, в порівнянні з початковим.
6. Якщо результат задовольняє користувача, то перехід до кроку 7, якщо ні до кроку 3.
7. Збереження обробленого зображення.
8. Якщо користувач виходить з системи, то кінець, якщо ні то перехід до кроку 2.

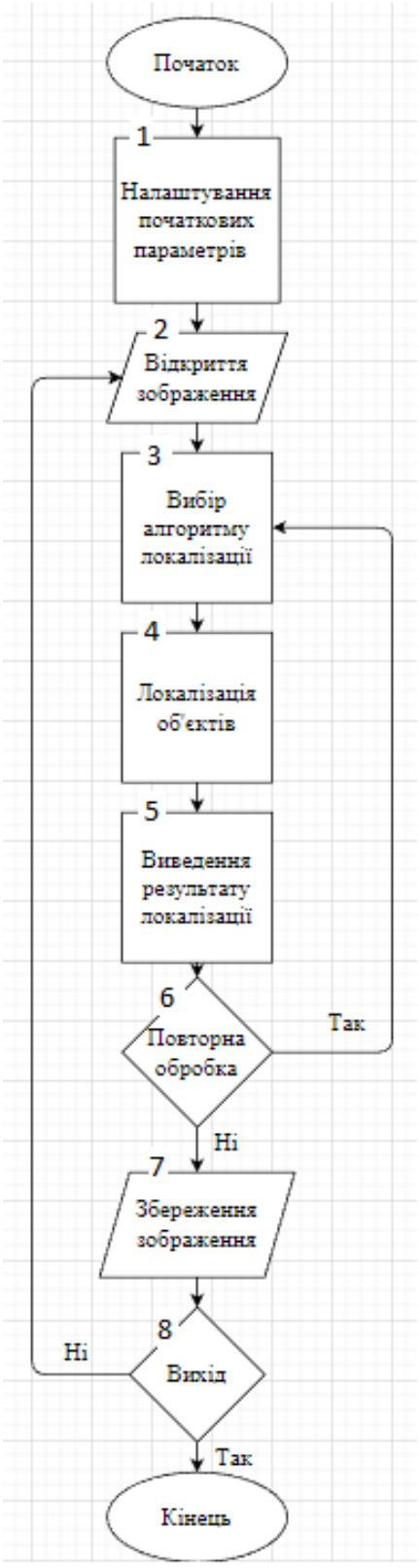

Рисунок 3.10 – Схема алгоритму роботи системи

3.6. Тестування реалізованої системи локалізації об’єктів

Тестування програмного продукту проводилось на зображеннях різних форматів, якостей та розмірів. На рисунку 3.11 наведено головне вікно застосунку із відкритим зображенням фігур.

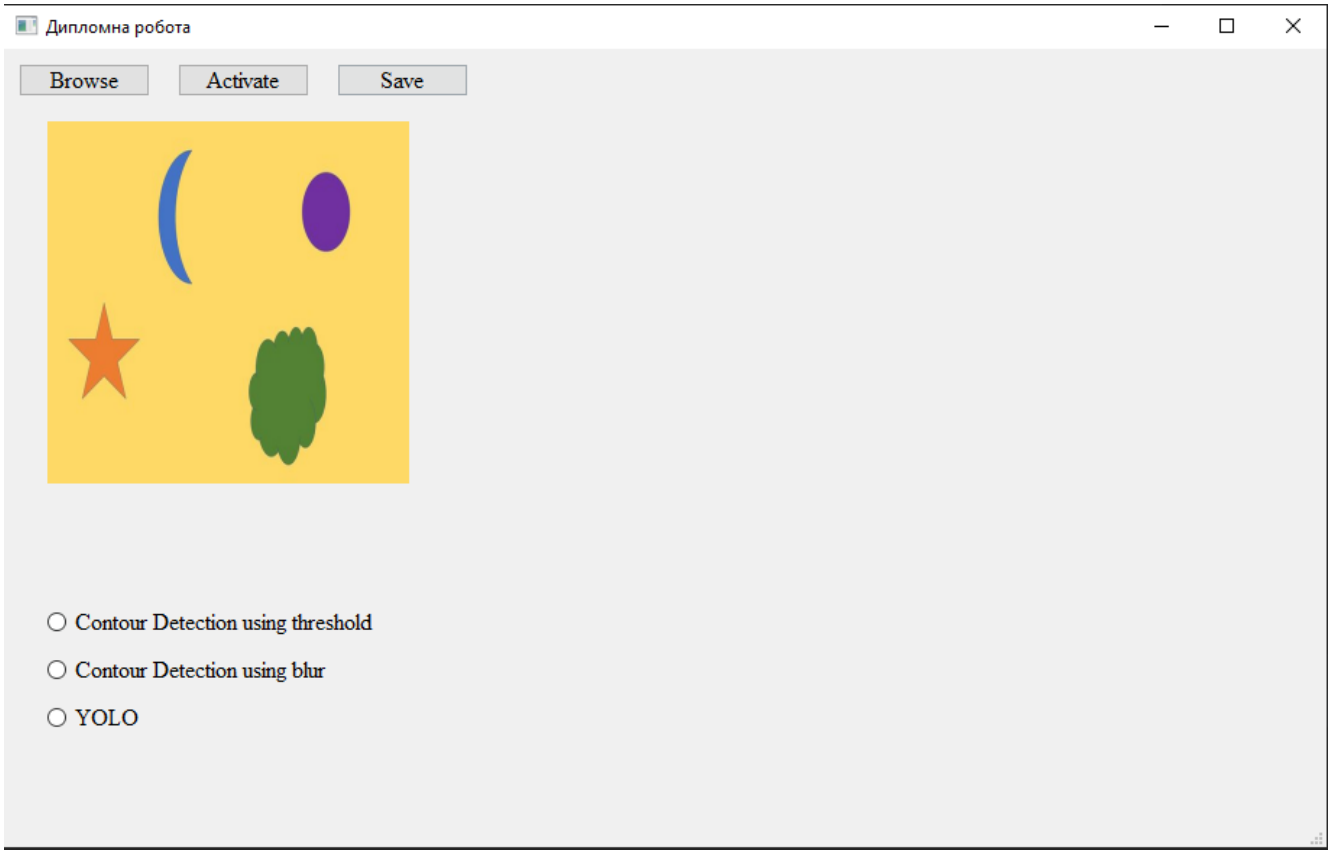

Рисунок 3.11 – Головне вікно програми з зображенням фігур

Спочатку протестовано роботу алгоритму локалізації контурів об’єктів використовуючи порогову бінаризацію. Алгоритм перетворює вхідне зображення до градації сірого після чого виконується порогова бінаризація (рис. 3.12).

```
img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(image=thresh, mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_NONE)
global image_copy
image_copy= image.copy()
cv2.drawContours(image_copy, contours, -1, (0, 255, 0), 2, lineType=cv2.LINE_AA)
```

Рисунок 3.12 – Програмна реалізація алгоритму локалізації з використанням порогової бінаризації

Як видно з рисунку 3.13 порогова бінаризація виконується коректно, однак одна з фігур втратила свої границі, у зв’язку з чим локалізація зазначеного зображення буде виконаною не коректно що і демонструє рисунок 3.14.

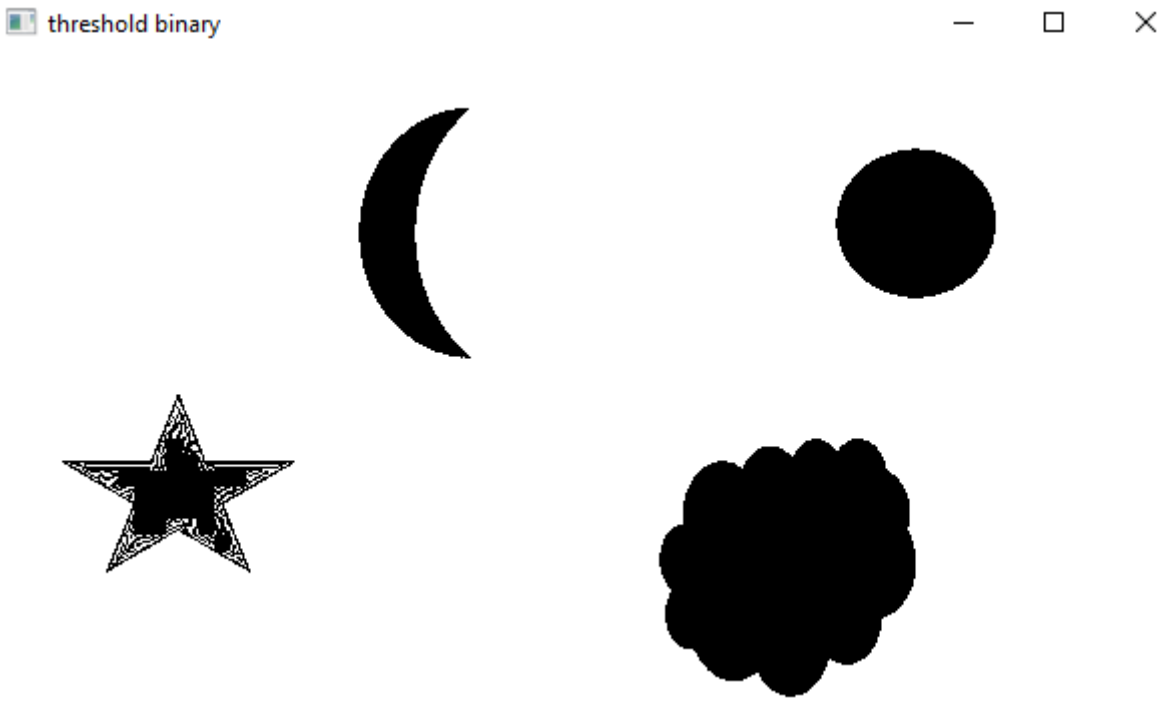

Рисунок 3.13 – Зображення оброблене алгоритмом порогової бінаризації

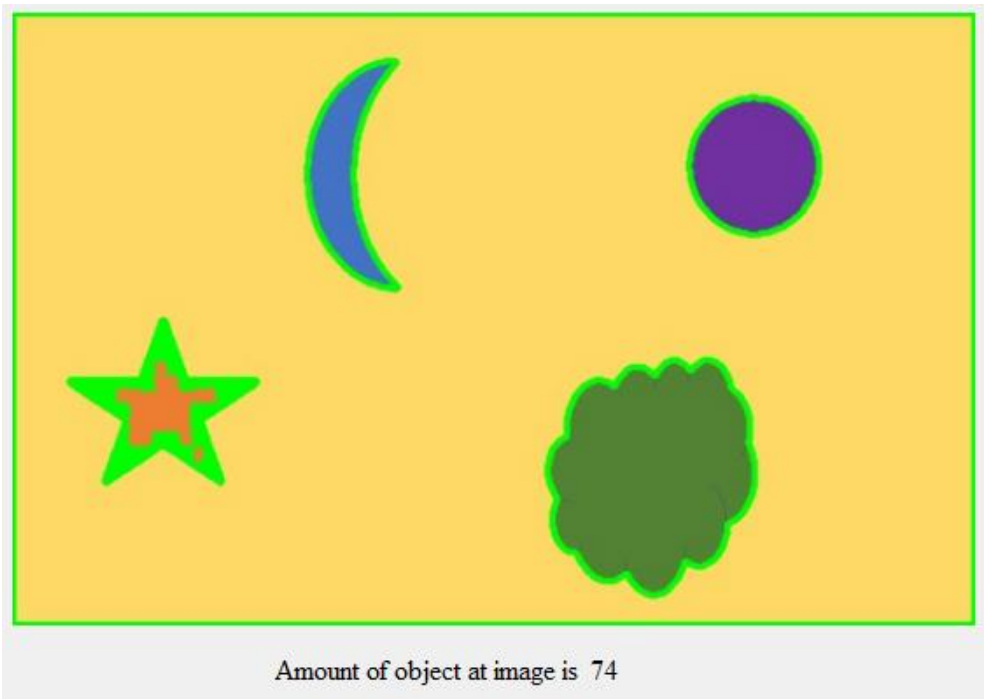

Рисунок 3.14 – Приклад некоректної локалізації об’єктів на зображенні

Для того щоб збільшити якість алгоритму було використано інвертовану порогову бінаризацію, однак такий варіант також не приніс бажаного результату, і лише збільшив кількість контурів навколо об’єктів, проте не визначив саме зображення ще одним об’єктом (рис.3.15).

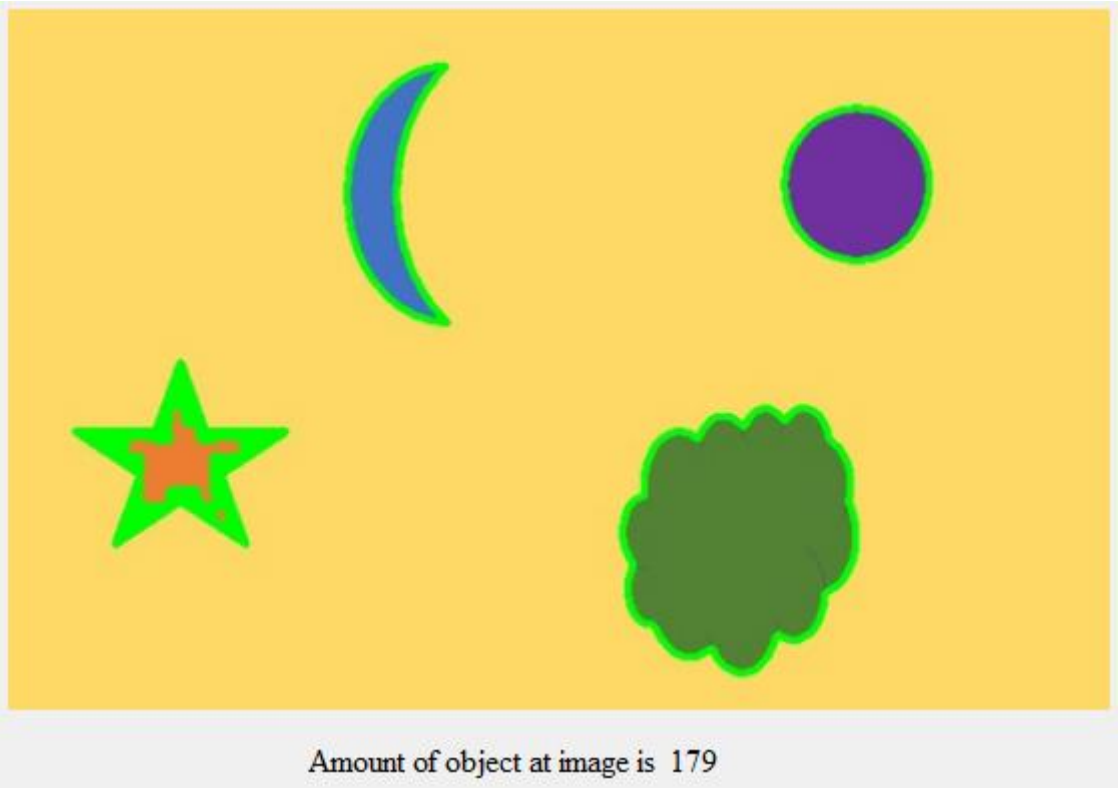

Рисунок 3.15 – Результат інвертованої порогової бінаризації.

Такий алгоритм більше підходить для локалізації об’єктів у випадку коли всі об’єкти на фото або наближені до темних або навпаки до світлих тонів. Рисунок 3.16 демонструє локалізацію об’єктів на світлому фоні.

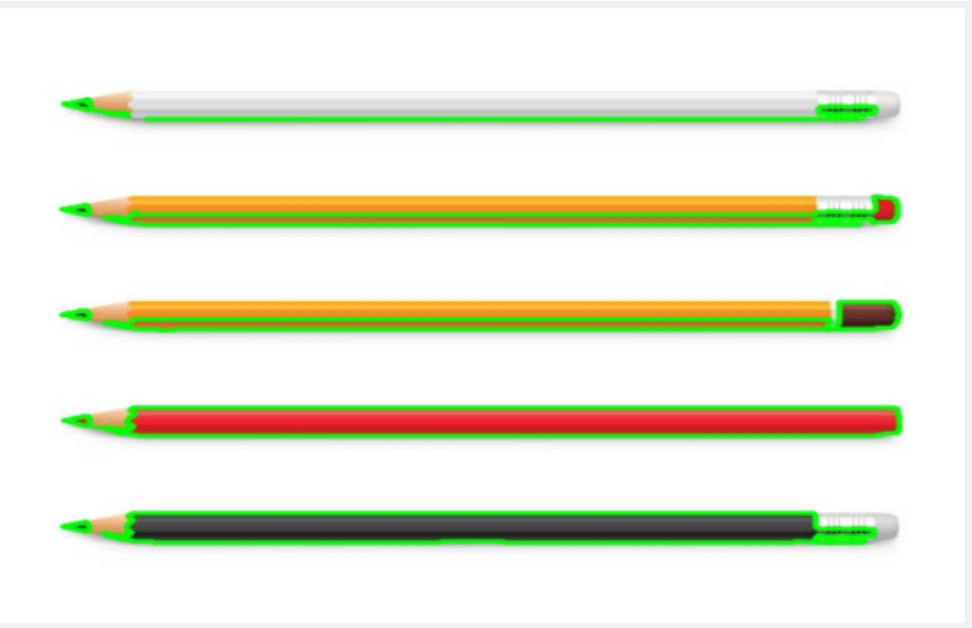

Рисунок 3.16 – Правильно локалізовані об’єкти на зображенні олівців.

Не завжди зображення можуть бути розташовані на світлому або темному. Тому спробуємо використати до зображень наступний алгоритм – розмивання Гауса (рис.3.17).

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (11, 11), 0)
cv2.imshow("GaussianBlur",gray)
canny = cv2.Canny(blur, 30, 150, 3)
dilated = cv2.dilate(canny, (1, 1), iterations=0)
cnt, hierarchy = cv2.findContours(
    dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
global rgb
rgb = image
cv2.drawContours(rgb, cnt, -1, (0, 255, 0), 2)
amount = "Amount of object at image is"
self.label_3.setText(amount)
sum_contour = len(cnt)
self.label_4.setText(str(sum_contour))
```

Рисунок 3.17 – Програмна реалізація локалізації з використанням розмивання Гауса

В його основу так само іде перетворення зображення до чорно-білого, після чого зображення оброблюється розмиванням Гауса(рис.3.18). До обробленого розмивання Гауса зображення використовуються алгоритм пошуку контурів Кенні і морфологічна операція розширення.

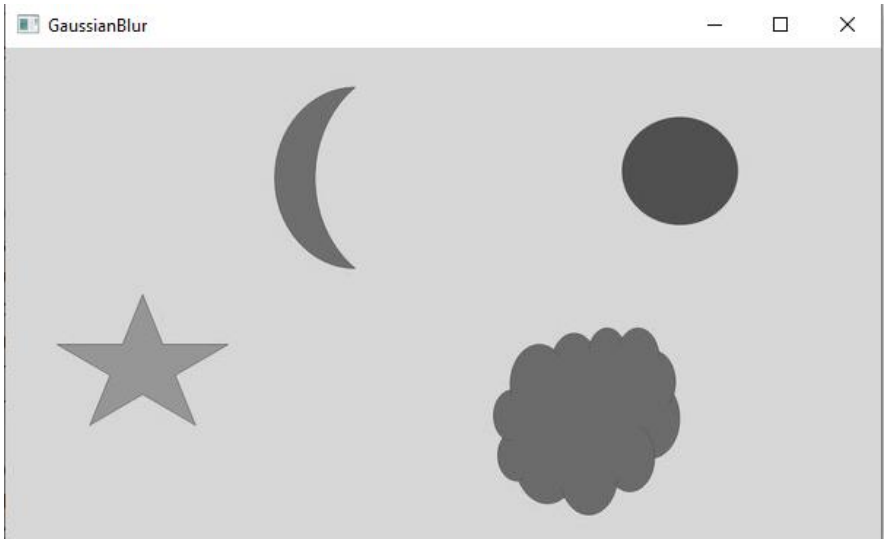

Рисунок 3.18 – Результат застосування розмивання Гауса до зображеннях в відтінках сірого

Після чого визначені контури додаються до початкового зображення завдяки чому і виконується локалізація об’єктів.

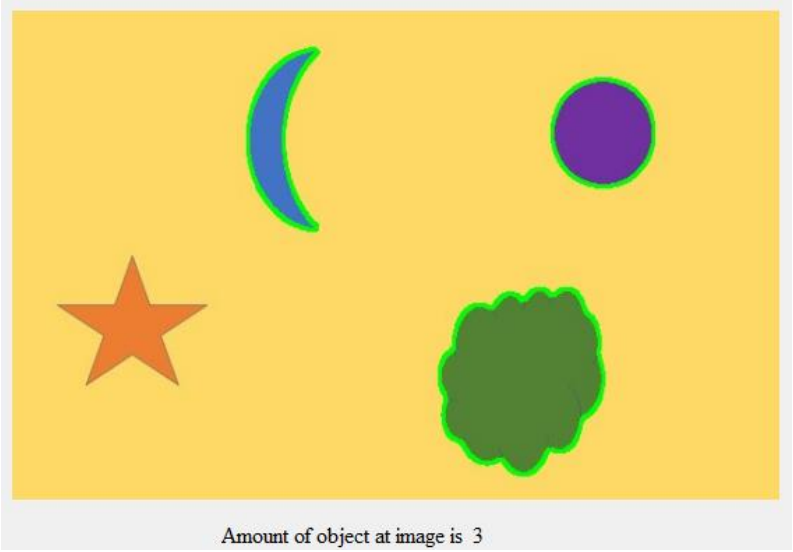

Рисунок 3.19 – Результат локалізації шляхом використання розмивання Гаусу

Як видно на рисунку 3.19 алгоритм зміг визначити 3 з 4 об’єктів, це пов’язано з тим що в основі пошуку контурів Кенні свою роль відіграє контрастні об’єктів з фоном, тому в цьому випадку після виконання обробки зірка була занадто подібною до кольору фону і не була визначеною як об’єкт. Рисунок 3.20 демонструє що при зміненні кольору зірка коректно відображається як один з об’єктів зображення.

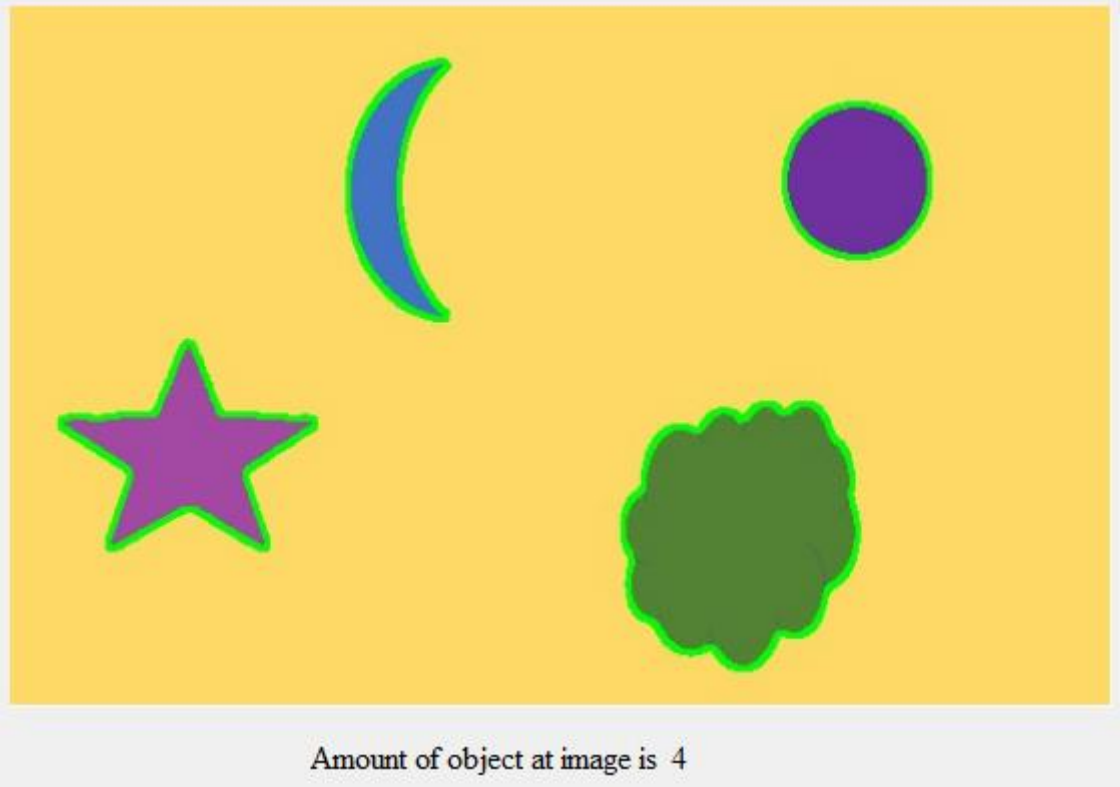

Рисунок 3.20 – Локалізація з зміненим кольором зірки

Не зважаючи на успіх з локалізацією різних фігур на зображенні, при поданні в даний алгоритм реальної фотографії тварин, або машин, якість локалізації не буде занадто високою. Рисунок 3.21 демонструє, що алгоритм коректно визначив розташування об’єктів які нас цікавлять однак, окрім них воно визначило як об’єкт перехід фону трави до дерев і декілька квітів біля собаки.

Рисунок 3.21 – Локалізація зображення з кішкою та собакою

Тому перейдемо до тестування третього алгоритму – YOLO. В основі даного алгоритму йде навчання нейронної мережі розпізнавати об’єкти різних типів, завдяки підготуванню попередньої бази класів і зображень пов’язаних з ними (рис.3.22).

```
yolo_image=image
Width = yolo_image.shape[1]
Height = yolo_image.shape[0]
scale = 0.00392

classes = None
with open("yolov3.txt", 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')
blob = cv2.dnn.blobFromImage(yolo_image, scale, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))
```

Рисунок 3.22 – Додавання потрібних для роботи алгоритму класів та конфігурації.

Всі попередні дії готують вхідне зображення для проходження через глибоку нейронну мережу. Після проходження зображення через глибоку нейронну мережу, буде визначено попереднє розташування об’єкту, однак через велику кількість подібних входжень один і той самий об’єкт може бути локалізований декілька разів, що уникнути такого ми використовуємо Non Maximum Suppresion(NMS), це дозволить нам прибрати локалізації які занадто далеко або занадто близько до об’єкту(рис.3.23).

```
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
for i in indices:
    try:

        box = boxes[i]
    except:
        i = i[0]
        box = boxes[i]

    x = box[0]
    y = box[1]
    w = box[2]
    w=w+x
    h = box[3]
    h=h+y
    color = COLORS[class_id]
    cv2.rectangle(yolo_image, (round(x), round(y)), (round(w), round(h)), color, 2)
```

Рисунок 3.23 – Програмна реалізація алгоритму YOLO

Данна програмна реалізація була натренована локалізувати об’єкти 80 різних типів. В таблиці 3.2 наведено деякі з них, повний список знаходиться в Додатку А.

Таблиця 3.2 – класи об’єктів які система локалізує.

Назва класу	Об’єкт локалізації
person	людина
car	машина
cat	кіт
dog	собака

Продовження таблиці 3.2

bus	автобус
airplane	літак
banana	банан
pizza	піца
donut	пончик
tv	телевізор
laptop	ноутбук
chair	стілець
book	книга
keyboard	клавіатура
microwave	мікрохвильовка

Зображення з кішкою та собакою використане раніше відмінно демонструє якість роботи алгоритму (рис.3.24).

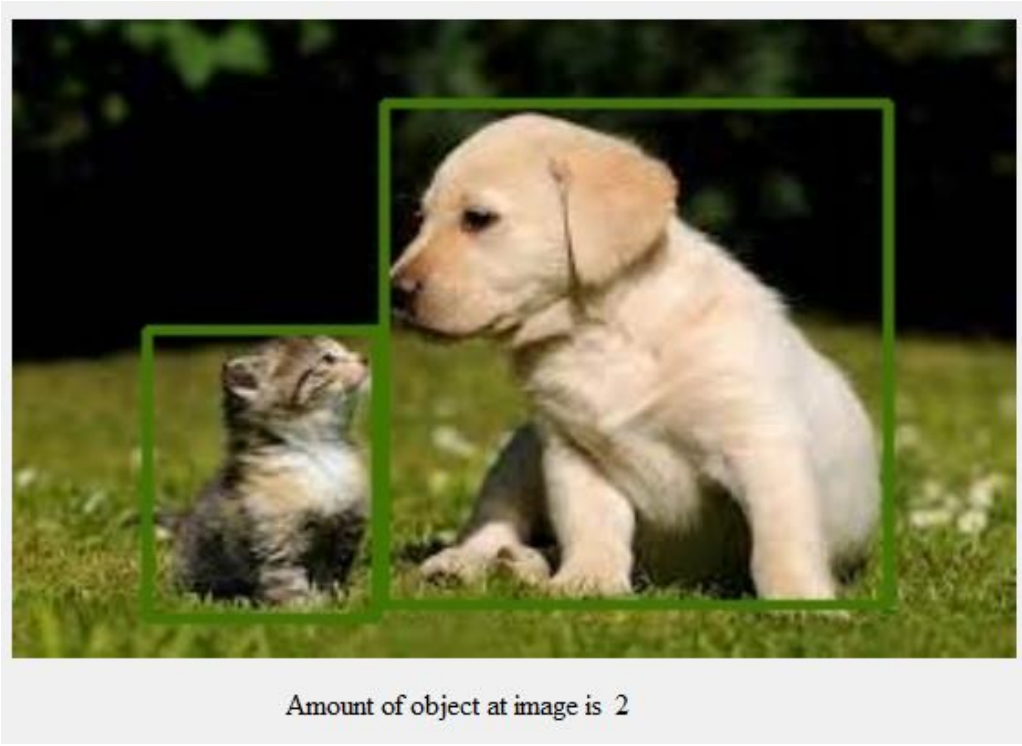

Рисунок 3.24 – Локалізація об’єктів з використанням алгоритму YOLO

Рисунок 3.25 демонструє, що не зважаючи на велику кількість об’єктів для локалізації алгоритм швидко визначає їх і має дуже малі похибки.

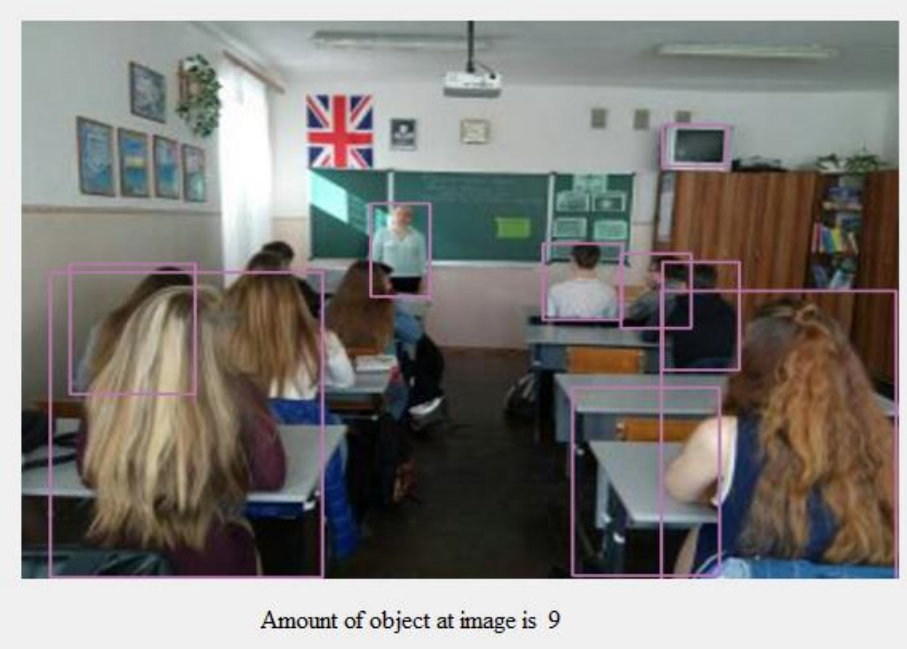

Рисунок 3.25 – Локалізація великої кількості об’єктів на одному зображенні

Однак за межами об’єктів на яких даний алгоритм натренований результати будуть відсутніми, однак їх легко можна буде змінити якщо натренувати мережу по ще більшій кількості різноманітних об’єктів, файл yolov3.txt містить в собі всі об’єкти на яких мережа була натренована.

Тестування було проведено для зображень різних форматів та розмірів. В залежності від якості, кольорового тону і використаного алгоритму, обробка набувала цілком різноманітних результатів і швидкостей локалізації об’єктів на зображенні.

3.7. Висновок

В ході програмної реалізації системи локалізації об’єктів на зображенні обґрунтовано вибір бібліотеки OpenCV. Визначено переваги до яких відносяться: мультиплатформенність, легкість роботи з алгоритмами в основі яких лежить комп’ютерний зір.

Для програмної реалізації обрано мову Python, з якою легко працювати, а завдяки OpenCV-Python не втрачає швидкості роботи алгоритмів. Для побудови графічного інтерфейсу було використано PyQt.

За методологією SADT було спроектовано програмні засоби системи локалізації об’єктів на зображенні шляхом розробки діаграм IDEF0 та DFD, які описують функції та потоки даних системи.

Наведено інтерфейс застосунку з описом послідовності кроків для зручності роботи в програмі користувача.

На основі структурної схеми локалізації об’єктів на зображенні побудовано і описано діаграму класів та алгоритм роботи системи.

Виконано тестування алгоритмів бінаризації порогових значень, розмивання Гауса, YOLO. Визначено плюси та мінуси кожного з алгоритмів. В залежності від якості, кольорового тону і використаного алгоритму, обробка набувала цілком різноманітних результатів і швидкостей локалізації об’єктів на зображенні.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи розроблено системи локалізації об’єктів на зображенні. При аналізі методів цифрової обробки зображень відзначено, що обробка зображень відіграє важливу роль в різноманітних галузях. Виконано дослідження і опис стадій цифрової обробки зображень. Визначено типові задачі для відтворення зображень, та розглянуто просторову класифікацію відносно розв’язку задач. Проаналізовано вже існуючи програмні засоби та деякі алгоритми, які в них використовуються.

У другому розділі магістерської кваліфікаційної роботи досліджено і проаналізовано типові методи локалізації об’єктів.

Визначено основні напрямки застосування нейронних мереж, та розглянуто класифікацію нейронних мереж.

Проаналізовано яким чином працюють згорткові нейронні мережі, та виконано порівняння точності локалізації та швидкодії вже існуючих нейронних мереж, таких як: 30Hz DPM, YOLO, Fastest DPM, R-CNN, Fast R-CNN, SSD512.

Після виконаних досліджень до програмної реалізації системи локалізації об’єктів на зображенні було вирішено реалізувати такі методи:

– Бінаризація порогових значень.
– розмивання Гауса.
– YOLO.

Третій розділ описує проєктування та розробку програмного забезпечення системи локалізації об’єктів на зображенні ображень на мові програмування Python. Обґрунтовано вибір бібліотеки OpenCV.

Спроєктовано програмні засоби системи локалізації об’єктів на зображенні за методологією SADT. Розроблено діаграми IDEF0 та DFD, які описують функціонал та потоки даних системи.

Наведено інтерфейс застосунку з описом послідовності кроків для зручності роботи в програмі користувача.

Виконано тестування алгоритмів бінаризації порогових значень, розмивання Гауса, YOLO. Визначено плюси та мінуси кожного з алгоритмів. В залежності від якості, кольорового тону і використаного алгоритму, обробка набувала цілком різноманітних результатів і швидкостей локалізації об’єктів на зображенні.

Розроблена система:

– дозволяє вибрати метод локалізації об’єктів на зображенні після додавання зображення з трьох можливих (бінаризації порогових значень, розмивання Гауса, YOLO);

– виділяє об’єкти на зображенні контурами або квадратом різноманітних кольорів;

– в консолі виводяться координати локалізованих об’єктів.

– рахує кількість об’єктів на зображенні;

– дозволяє зберегти результат локалізації.

Розроблену систему можна використовувати:

– в медичній галузі для діагностування та виявлення пухлин та інших патологій, вивчення анатомічної структури органів;

– в сфері забезпечення суспільної безпеки міста для розпізнавання обличь, відбитків пальців, небезпечних предметів(залишених без нагляду);

– в військовій сфері для виявлення снайперів, диверсійно-розвідувальних груп, типу кораблів та військової техніки;

– декодування об’єктів, перевірки змін на поверхні землі шляхом порівняння даних супутникових знімків за якісь проміжки часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вовк С.М. Методи обробки зображень так комп’ютерний зір: навч. посіб. Дніпро : Ліра, 2016. 148 с.

2. Гонсалес Р.С., Вудс Р.Е. Цифровая обработка изображений / пер. с англ Л.И. Рубанова, П.А. Чочиа. Москва : Техносфера, 2012. 1104 с.

3. О.А. Кобилін, І.С. Творошенко. Методи цифрової обробки зображень: навч. посібник. Харків: ХНУРЕ, 2021. 124 с.

4. Поляков. А.Ю., Брусенцев В.А. Методы и алгоритмы компьютерной графики в примерах на Visual C++, 2-е изд. СПб.: БХВ-Петербург, 2003. 560 с.

5. Потапов А. Системы компьютерного зрения: современные задачи и методы. *CONTROL ENGINEERING* 2014. №49 URL:https://controleng.ru/wp-content/uploads/CE_149_sistemy_kompyuternogo_zreniya.pdf. (дата звернення: 20.10.2022).

6. Прэтт У. Цифровая обработка изображений. Москва.: Мир, 2001. 780с.

7. Фисенко В.Т., Фисенко Т.Ю. Компьютерная обработка и распознавание изображений: учеб. пособие. СПб: СПбГУ ИТМО, 2008. 192 с.

8. Шапиро Л., Стокман. Дж. Компьютерное зрение /пер. с англ 3-е изд. Москва: БИНОМ. Лаборатория знаний, 2015. 763 с.

9. Prateek Joshi, David Millan Escriva, Vinicius Godoy. OpenCV By Example. Birmingham: Packt Publishing Ltd., 2016. 306 с.

10. Retinex Poisson Equation: a Model for Color Perception. IPOL Journal. URL: http://www.ipol.im/pub/art/2011/lmps_rpe (дата звернення 20.11.2022)

11. About – OpenCV library. Url: https://opencv.org/about (дата звернення 25.10.2022)

12. What is MATLAB. URL: https://www.mathworks.com/discovery/what-is-matlab.html (дата звернення 25.10.2022)

13. About Qt - Qt WIKI. URL: https://wiki.qt.io/About_Qt (дата звернення 25.10.2022)

14. Гвоздева Т.В., Б.А.Баллод. Проектирование информационных систем: учеб. пособие. Ростов н/Д: Феникс, 2009. 508 с.

15. Маклаков С.В. Моделирование бизнес-процессов с AllFusion PM. – 2-е изд., испр. и дополн. Москва: Диалог-МИФИ, 2008. 224 с.

16. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования - 3-е изд./ пер. с англ. А. Петрухов СПб: СимволПлюс, 2011. 192 с.

17. Mark S. Nixon, Alberto S. Aguado. Feature Extraction and Image Processing. Woburn: Butterworth-Heinemann, 2008. 360 c.

18. Erik Reinhard. High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting. Massachusetts: Morgan Kaufmann, 2006. 278 c.

19. Nils J. Nilsson. Introduction to machine learning. California: Stanford University, 2005, c.188.

20. PyTorch documentation. URL:https://pytorch.org/docs/stable/index.html (дата звернення 20.11.2022).

21. Кравченко С.М, Гришкун Є.О., Власенко О.В. Методи класифікації машинного навчання з використанням бібліотеки Scikit-Learn. *Інформатика, обчислювальна техніка та автоматизація*. 2020. Том 31 Ч. 1 № 3, С. 121-125.

22. You Only Look Once: Unifed, Real-Time Object Detection. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf (дата звернення 24.11.2022).

23. YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. URL: https://arxiv.org/pdf/2209.02976.pdf (дата звернення 24.11.2022).

24. YOLOv3: An Incremental Improvement. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf (дата звернення 24.11.2022).

25. Mouton C., Myburgh C. J., Davel H. M. Stride and Translation Invariance in CNNs. SACAIR 2021: Artificial Intelligence Research pp 267–281.

26. Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO. Object Detection Algorithms. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (дата звернення 21.11.2022).

27. Hao Gao. Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch. URL: https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad (дата звернення 20.11.2022).

28. The PASCAL Visual Object Classes Challenge 2012. URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/ (дата звернення 13.11.2022).

Додаток А

Лістинг програми

Main.py

```
from PyQt5 import  QtCore, QtGui,QtWidgets,uic
from PyQt5.QtWidgets import QMainWindow, QApplication,
QWidget,QLineEdit,QPushButton,QFileDialog
from PyQt5.QtGui import QImage
from tkinter import filedialog

import sys,cv2,imutils,os
import numpy as np

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(914, 551)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(110, 380, 300, 100))
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        font.setBold(False)
        font.setWeight(50)
        self.verticalLayoutWidget.setFont(font)
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.radioButton_2 = QtWidgets.QRadioButton(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        font.setBold(False)
        font.setWeight(50)
        self.radioButton_2.setFont(font)
        self.radioButton_2.setObjectName("radioButton_2")
        self.verticalLayout.addWidget(self.radioButton_2)
        self.radioButton = QtWidgets.QRadioButton(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        font.setBold(False)
        font.setWeight(50)
        self.radioButton.setFont(font)
        self.radioButton.setObjectName("radioButton")
        self.verticalLayout.addWidget(self.radioButton)
        self.radioButton_3 = QtWidgets.QRadioButton(self.verticalLayoutWidget)
        font = QtGui.QFont()
        font.setFamily("Times New Roman")
        font.setPointSize(12)
        font.setBold(False)
        font.setWeight(50)
        self.radioButton_3.setFont(font)
        self.radioButton_3.setObjectName("radioButton_3")
        self.verticalLayout.addWidget(self.radioButton_3)
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
```

```
self.pushButton.setGeometry(QtCore.QRect(10, 10, 91, 23))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
font.setBold(False)
font.setWeight(50)
self.pushButton.setFont(font)
self.pushButton.setObjectName("pushButton")
self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_2.setGeometry(QtCore.QRect(120, 10, 91, 23))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
font.setBold(False)
font.setWeight(50)
self.pushButton_2.setFont(font)
self.pushButton_2.setObjectName("pushButton_2")
self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
self.pushButton_3.setGeometry(QtCore.QRect(230, 10, 91, 23))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
font.setBold(False)
font.setWeight(50)
self.pushButton_3.setFont(font)
self.pushButton_3.setObjectName("pushButton_3")

self.label = QtWidgets.QLabel(self.centralwidget)
self.label.setGeometry(QtCore.QRect(30, 50, 250, 250))

self.label.setScaledContents(True)
self.label.setObjectName("label")

self.label_2 = QtWidgets.QLabel(self.centralwidget)
self.label_2.setGeometry(QtCore.QRect(350, 50, 550, 350))

self.label_2.setScaledContents(True)
self.label_2.setObjectName("label_2")

self.label_3 = QtWidgets.QLabel(self.centralwidget)
self.label_3.setGeometry(QtCore.QRect(500, 250, 550, 350))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
font.setBold(False)
font.setWeight(50)
self.label_3.setFont(font)
self.label_3.setObjectName("label_3")

self.label_4 = QtWidgets.QLabel(self.centralwidget)
self.label_4.setGeometry(QtCore.QRect(680, 250, 580, 350))
font = QtGui.QFont()
font.setFamily("Times New Roman")
font.setPointSize(12)
font.setBold(False)
font.setWeight(50)
self.label_4.setFont(font)
self.label_4.setObjectName("label_4")



MainWindow.setCentralWidget(self.centralwidget)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
```

```
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

        self.fname=None
        self.image=None
        self.filepath=None
        self.pixmap=None
        self.tmp=None




    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Дипломна робота"))
        self.radioButton_2.setText(_translate("MainWindow", "Contour Detection
using threshold"))
        self.radioButton.setText(_translate("MainWindow", "Contour Detection using
blur "))
        self.radioButton_3.setText(_translate("MainWindow", "YOLO"))
        self.pushButton.setText(_translate("MainWindow", "Browse"))
        self.pushButton_2.setText(_translate("MainWindow", "Activate"))
        self.pushButton_3.setText(_translate("MainWindow", "Save"))
        self.pushButton.clicked.connect(self.pushButton_Browse)
        self.pushButton_2.clicked.connect(self.pushButton_Activate)
        self.pushButton_3.clicked.connect(self.pushButton_Save)


    def pushButton_Browse(self):
        self.filepath = filedialog.askopenfilename()
        global image
        image = cv2.imread(self.filepath)
        self.pixmap = QtGui.QPixmap(self.filepath)
        self.label.setPixmap(self.pixmap)


    def pushButton_Activate(self):
        if self.radioButton_2.isChecked():
            img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            ret, thresh = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY)
            contours, hierarchy = cv2.findContours(image=thresh,
mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_NONE)
            global image_copy
            image_copy= image.copy()
            cv2.drawContours(image_copy, contours, -1, (0, 255, 0), 2,
lineType=cv2.LINE_AA)

            amount= "Amount of object at image is"
            self.label_3.setText(amount)
            sum_contour=len(contours)
            self.label_4.setText(str(sum_contour))
            cv2.imwrite("contours_threshold_image1.jpg", image_copy)
            self.filepath="D:/Dyplom/venv/contours_threshold_image1.jpg"
            self.pixmap = QtGui.QPixmap(self.filepath)
            os.remove("D:/Dyplom/venv/contours_threshold_image1.jpg")
            self.label_2.setPixmap(self.pixmap)
        if self.radioButton.isChecked():
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray, (11, 11), 0)
            canny = cv2.Canny(blur, 30, 150, 3)
```

```
            dilated = cv2.dilate(canny, (1, 1), iterations=0)

            cnt, hierarchy = cv2.findContours(
                dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
            global rgb
            rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            cv2.drawContours(rgb, cnt, -1, (0, 255, 0), 2)
            amount = "Amount of object at image is"
            self.label_3.setText(amount)
            sum_contour = len(cnt)
            self.label_4.setText(str(sum_contour))
            cv2.imwrite("contours_blur_image1.jpg", rgb)
            self.filepath = "D:/Dyplom/venv/contours_blur_image1.jpg"
            self.pixmap = QtGui.QPixmap(self.filepath)
            os.remove("D:/Dyplom/venv/contours_blur_image1.jpg")
            self.label_2.setPixmap(self.pixmap)
        if self.radioButton_3.isChecked():
            yolo_image=image
            Width = yolo_image.shape[1]
            Height = yolo_image.shape[0]
            scale = 0.00392

            classes = None
            with open("yolov3.txt", 'r') as f:
                classes = [line.strip() for line in f.readlines()]

            COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

            net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')

            blob = cv2.dnn.blobFromImage(yolo_image, scale, (416, 416), (0, 0, 0),
True, crop=False)

            net.setInput(blob)

            outs = net.forward(get_output_layers(net))

            class_ids = []
            confidences = []
            boxes = []
            conf_threshold = 0.5
            nms_threshold = 0.4

            for out in outs:
                for detection in out:
                    scores = detection[5:]
                    class_id = np.argmax(scores)
                    confidence = scores[class_id]
                    if confidence > 0.5:
                        center_x = int(detection[0] * Width)
                        center_y = int(detection[1] * Height)
                        w = int(detection[2] * Width)
                        h = int(detection[3] * Height)
                        x = center_x - w / 2
                        y = center_y - h / 2
                        class_ids.append(class_id)
                        confidences.append(float(confidence))
                        boxes.append([x, y, w, h])

            indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
nms_threshold)
            for i in indices:
                try:
```

```
                    box = boxes[i]
                except:
                    i = i[0]
                    box = boxes[i]

                x = box[0]
                y = box[1]
                w = box[2]
                w=w+x
                h = box[3]
                h=h+y
                color = COLORS[class_id]
                cv2.rectangle(yolo_image, (round(x), round(y)), (round(w),
round(h)), color, 2)


            amount = "Amount of object at image is"
            self.label_3.setText(amount)
            sum_contour = len(indices)
            self.label_4.setText(str(sum_contour))
            cv2.imwrite("contours_blur_image1.jpg", yolo_image)
            self.filepath = "D:/Dyplom/venv/contours_blur_image1.jpg"
            self.pixmap = QtGui.QPixmap(self.filepath)
            os.remove("D:/Dyplom/venv/contours_blur_image1.jpg")
            self.label_2.setPixmap(self.pixmap)

    def pushButton_Save(self):
        savepath=filedialog.askdirectory()
        if self.radioButton_2.isChecked():

cv2.imwrite(os.path.join(savepath,"Threshold_localization_image.jpg"),image_copy)
        if self.radioButton.isChecked():
            cv2.imwrite(os.path.join(savepath,"Blur_localization_image.jpg"),rgb)
        if self.radioButton.isChecked():
            cv2.imwrite(os.path.join(savepath,"Blur_localization_image.jpg"),rgb)
        if self.radioButton_3.isChecked():
            cv2.imwrite(os.path.join(savepath,"YOLO_localization_image.jpg"),
yolo_image)

        print("Button pressed")


def get_output_layers(net):
    layer_names = net.getLayerNames()
    try:
        output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]
    except:
        output_layers = [layer_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]

    return output_layers


if __name__ == "__main__":

    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

yolo.txt

```
person
bicycle
car
motorcycle
airplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse
sheep
backpack
umbrella
handbag
tie
suitcase
frisbee
skis
snowboard
sports ball
kite
baseball bat
baseball glove
skateboard
surfboard
tennis racket
bottle
wine glass
cup
fork
knife
spoon
bowl
banana
apple
sandwich
orange
broccoli
carrot
hot dog
pizza
donut
cake
chair
couch
potted plant
bed
dining table
toilet
tv
laptop
mouse
remote
```

keyboard
cell phone
microwave
oven
toaster
sink
refrigerator
book
clock