Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

«ІНФОРМАЦІЙНІ СИСТЕМА РОЗПІЗНАВАННЯ ЕЛЕМЕНТІВ ЗОБРАЖЕНЬ»

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія
програмного забезпечення»
Дяденчук Павло Андрійович

Науковий керівник:
к.ф.-м.н., доцент Мороз І. П.

Рівне – 2025

ЗМІСТ

ВСТУП

В епоху стрімкого технологічного прогресу та цифровізації суспільства, комп'ютерний зір утвердився як один з ключових напрямків розвитку інформаційних технологій, що найбільш динамічно розвиваються. Здатність машин «бачити» та інтерпретувати візуальну інформацію відкриває широкі перспективи для автоматизації складних завдань у різноманітних сферах. Важливою складовою комп'ютерного зору є функція розпізнавання об'єктів, яка передбачає виконання послідовних процесів: отримання візуальних даних, їх ретельну обробку, глибокий аналіз та подальшу класифікацію ідентифікованих об'єктів з цифрового зображення. Ефективне вирішення цих завдань ґрунтується на синергії методів, запозичених з фундаментальних наук, таких як фізика (оптика), математична статистика, проективна геометрія, та сучасних досягнень у теорії машинного навчання.

Особливе значення в рамках прикладних застосувань комп'ютерного зору набуває задача автоматичного розпізнавання елементів дорожньої інфраструктури. Цей напрямок є критично важливим для забезпечення безпеки та ефективності функціонування сучасних транспортних потоків, зокрема для розвитку систем допомоги водієві (Advanced Driver-Assistance Systems, ADAS), створення надійних систем автономного керування транспортними засобами, а також для автоматизованого моніторингу стану дорожнього покриття та оперативного оновлення цифрових картографічних сервісів. Серед усього спектру дорожніх інформаційних елементів, знаки «початок населеного пункту» відіграють вирішальну роль, оскільки вони інформують учасників дорожнього руху про зміну швидкісного режиму, пріоритетів та інших специфічних правил, що діють у межах забудованих територій, тим самим безпосередньо впливаючи на безпеку та організацію дорожнього руху.

Актуальність теми даної кваліфікаційної роботи визначається нагальною потребою у розробці та впровадженні високопродуктивних та надійних інформаційних систем, здатних в автоматичному режимі з високою точністю ідентифікувати саме знаки початку населеного пункту. Вирішення цієї

задачі ускладнюється значною варіативністю умов отримання зображень у реальному дорожньому середовищі: динамічна зміна освітлення (від яскравого сонячного світла до сутінків та нічного часу), несприятливі атмосферні явища (дощ, сніг, туман), наявність візуальних перешкод та шумів, можливість часткового перекриття знаків (оклюзія) елементами оточення (дерева, інші транспортні засоби, споруди), а також різний фізичний стан самих знаків (забруднення, вицвітання, деформації). Ці фактори висувають підвищені вимоги до робастності, адаптивності та обчислювальної ефективності алгоритмів розпізнавання.

Метою даної кваліфікаційної роботи є дослідження, розробка та програмна реалізація інформаційної системи, що спеціалізується на автоматичному виявленні та локалізації елементів зображень (наприклад, знаків початку населеного пункту на цифрових зображеннях, отриманих в умовах, наближених до реальних дорожніх сценаріїв).

Об’єктом дослідження виступає процес автоматизованого розпізнавання специфічних візуальних об’єктів на статичних цифрових зображеннях.

Предметом дослідження є методи, моделі та алгоритми цифрової обробки зображень, включаючи техніки попередньої підготовки даних, сегментації цільових об’єктів, аналізу їх геометричних та колірних характеристик, а також вивчення потенціалу застосування архітектур згорткових нейронних мереж для підвищення точності та надійності виявлення знаків початку населеного пункту.

Для досягнення поставленої мети в рамках виконання кваліфікаційної роботи передбачається вирішення таких **завдань**:

1. Провести теоретичний аналіз предметної області, включаючи основи розпізнавання образів, формалізацію задачі виявлення знаків початку населеного пункту та огляд релевантних методів обробки зображень.
2. Обґрунтувати вибір програмних засобів та налаштувати середовище розробки для реалізації інформаційної системи.
3. Розробити алгоритм виявлення знаків початку населеного пункту, що включає етапи обробки, сегментації та, можливо, кластеризації, а також розглянути впровадження згорткових нейронних мереж.

4. Здійснити програмну реалізацію розробленого алгоритму та провести тестування створеної інформаційної системи.

5. Підготувати розрахунково-пояснювальну записку, візуалізувати та оцінити отримані результати ефективності алгоритму.

Практична значущість виконаної роботи полягає у створенні функціональної інформаційної системи для виявлення знаків початку населеного пункту, яка демонструє життєздатність запропонованого підходу та може слугувати основою для подальших наукових досліджень, а також для розробки більш комплексних та інтелектуальних рішень у сфері автоматизації аналізу дорожньої інфраструктури та підвищення загального рівня безпеки дорожнього руху.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Основи теорії розпізнавання образів та їх застосування

Розпізнавання образів є фундаментальною науковою дисципліною, що знаходиться на стику комп'ютерних наук, штучного інтелекту та прикладної статистики, і займається розробкою теоретичних основ, алгоритмів та програмних систем, здатних автоматично виявляти, аналізувати та класифікувати закономірності, або образи, у різноманітних даних. Ця галузь досліджує методи ідентифікації характерних особливостей та властивостей об'єктів, що дозволяє здійснювати їх подальшу класифікацію в межах певної предметної області. Під образом розуміється група об'єктів, об'єднаних спільними рисами та ознаками, що тим самим формують певну класифікацію. Розпізнавання образів, таким чином, може трактуватися як процес класифікації отриманих даних на основі попередньо вивчених знань або статистичної інформації, що міститься в цих даних. Для ефективного порівняння та класифікації, усі необхідні знання та характеристики задаються для кожного з визначених класів об'єктів [8, 13, 21].

У контексті аналізу візуальної інформації, зокрема цифрових зображень, процес розпізнавання образів є багатоетапним. Він розпочинається з отримання вихідних даних, таких як цифрові зображення або відеопослідовності, які фіксують об'єкти або сцени, що підлягають аналізу. Наступним важливим кроком є попередня обробка цих даних, метою якої є покращення їх якості, усунення шумів та артефактів, корекція освітленості, а також нормалізація, що суттєво полегшує подальші етапи аналізу. Для того щоб обчислювальна машина могла сприйняти візуальну інформацію, зображення має пройти процес цифрової обробки, відомий як оцифровка, тобто перетворення образотворчої форми у числові дані. З точки зору машини, зображення представляється як набір чисел, що відображають інтенсивність кольору або яскравості в кожній точці. Знімок розділяється на дрібні елементи, які називаються пікселями, і, таким чином, перетворюється на матрицю цілих чисел. Після попередньої обробки здійснюється сегментація – процес виділення на зображенні областей, що

становлять інтерес, тобто власне об’єктів, відокремлюючи їх від фону або інших нерелевантних елементів. Далі відбувається етап виділення ознак, на якому формується компактний та інформативний опис кожного виділеного об’єкта. Ці ознаки можуть включати геометричні характеристики (форма, розмір, орієнтація), колірні особливості, текстурні параметри тощо. Можливе представлення цих особливостей за допомогою дискретних, безперервних або бінарних змінних, а алгоритм знаходження важливих параметрів об’єкта називається функцією особливості. На основі отриманих даних формується так званий вектор властивостей (ознак), який слугує вхідними даними для наступного етапу – класифікації [17, 3, 25].

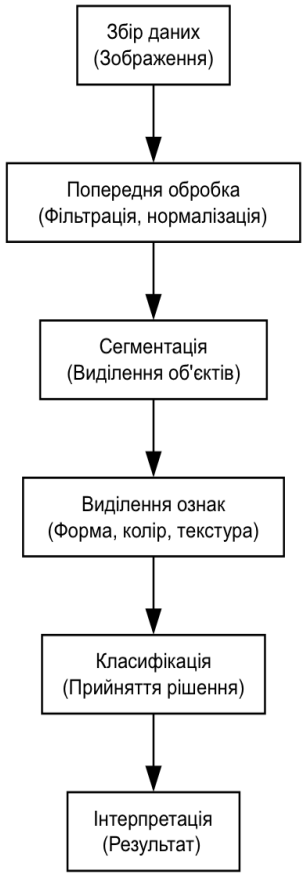

Рисунок 1.1 – Узагальнена схема процесу розпізнавання образів

На етапі класифікації, використовуючи певний алгоритм або модель (наприклад, на основі машинного навчання), об’єкт на базі його вектора ознак

відносять до одного з попередньо визначених класів. При успішному знаходженні відповідності між характеристиками об’єкта та знаннями системи про певний клас, об’єкт отримує відповідну мітку, що свідчить про успішну класифікацію. Завершальним етапом є інтерпретація отриманих результатів розпізнавання та прийняття на їх основі відповідних рішень. Загальну послідовність цих етапів наочно ілюструє блок-схема, наведена на рис. 1.1.

Невід’ємною частиною систем розпізнавання образів є процес навчання, який дозволяє системі адаптуватися та підвищувати свою точність шляхом аналізу прикладів. Якість та точність результатів системи розпізнавання значною мірою залежать від алгоритму навчання. Для цього використовуються спеціально підготовлені набори даних: навчальний та тестовий. Навчальний набір, що є збіркою прикладів (наприклад, зображень) з відомою класифікацією, використовується для побудови та налаштування моделі розпізнавання. Система виконує алгоритм, удосконалюючи свої вихідні дані на основі навчального набору. Тестовий набір, який не використовувався під час навчання, слугує для оцінки якості роботи системи в умовах, максимально наближених до реальних, та визначення її здатності до узагальнення [11, 31].

Технології розпізнавання образів знайшли надзвичайно широке застосування у різноманітних сферах сучасного життя. Вони використовуються для розпізнавання місцевості, класифікації змісту текстів, аналізу відеоматеріалів. Прикладами успішного впровадження є системи біометричної ідентифікації (розпізнавання облич, відбитків пальців, сканування сітківки ока), медична діагностика (аналіз рентгенівських знімків, МРТ, КТ), промисловий контроль якості продукції, системи безпеки та відеоспостереження. Також ці технології є основою для оптичного розпізнавання символів (OCR), систем розпізнавання мови, аналізу супутникових знімків у геоінформаційних системах, інтерпретації даних сейсмічного аналізу та класифікації радіолокаційних сигналів. Розпізнавання вже відомих об’єктів з бази даних та порівняння їх особливостей з новим об’єктом на фото або відео має дуже широке використання. У контексті даної кваліфікаційної роботи, фундаментальні принципи та методи теорії розпізнавання образів будуть застосовані для

розробки інформаційної системи, спеціально призначеної для виявлення такого специфічного типу об’єктів, як знаки початку населеного пункту [6, 19, 34].

1.2. Формалізація задачі розпізнавання елементів зображень

Формалізація задачі є фундаментальним етапом у проектуванні будь-якої інформаційної системи, адже саме вона закладає основу для подальшої розробки, чітко окреслюючи вхідні та вихідні параметри, а також бажані характеристики функціонування. **Дана** кваліфікаційна робота фокусується на створенні спеціалізованої інформаційної системи, головним завданням якої є автоматичне виявлення та точна локалізація дорожніх знаків, що сигналізують про початок населеного пункту, на цифрових зображеннях. Для українських доріг це, передусім, знаки двох типів: 5.49, який має білий фон, та 5.51, що характеризується синім фоном. Саме на їх ідентифікації і зосереджений розроблений алгоритм. Вхідними даними для цієї системи виступають цифрові зображення I, котрі можна розглядати як двовимірні функції I(x, y), де x та y є просторовими координатами, а значення функції визначає колір відповідного пікселя. Важливо розуміти, що ці зображення, отримані з різноманітних джерел, як-от автомобільні відеореєстратори чи камери спостереження, неминуче несуть у собі значну варіативність – вони відрізняються за розміром, якістю, умовами освітлення та ракурсом, що висуває додаткові вимоги до стійкості алгоритму. Для наочної ілюстрації об’єктів зацікавленості, на рисунку 1.2 представлено приклади дорожніх знаків 5.49 та 5.51, які є ціллю для розпізнавання інформаційною системою [8, 22, 1, 14, 28, 30].

Рисунок 1.2 – Приклади цільових дорожніх знаків

Результатом роботи системи є структурована інформація про наявність та місцезнаходження цільових дорожніх знаків. Формально, це список Ls = {S1, S2, ..., Sn}, де кожен елемент Si відповідає одному виявленому знаку. Для кожного такого знаку система повинна розрахувати його обмежувальний прямокутник Bi = (xs, ys, ws, hs), що визначає координати верхнього лівого кута, ширину та висоту знайденого об’єкта на зображенні. Візуально система позначає білі знаки зеленою рамкою, а сині – синьою, здійснюючи таким чином неявну класифікацію за кольором [5].

В основі алгоритму лежить аналіз характерних візуальних ознак цільових знаків. Ключовими є їхня прямокутна форма та кольорові характеристики. Для надійного розрізнення кольорів, незважаючи на зміни освітлення, використовується колірний простір HSV. Система налаштована на пошук специфічних діапазонів білого та синього кольорів, що відповідають знакам 5.49 та 5.51. Після етапу кольорової сегментації, де виділяються потенційні області-кандидати, настає черга комплексної геометричної фільтрації. По-перше, аналізується площаконтуру, що дозволяє відсіяти дрібні шуми і, водночас, забезпечує можливість виявлення знаків, що знаходяться на значній відстані і займають невелику частину кадру. По-друге, перевіряється співвідношення сторін, щоб відібрати об’єкти з пропорціями, характерними для дорожніх знаків (1.5-2.8). По-третє, критично важливою є перевірка форми: за допомогою апроксимації контуру cv2.approxPolyDP система шукає саме чотирикутники, що дозволяє їй ідентифікувати знаки, навіть якщо вони зняті під кутом і мають певні перспективні спотворення. Нарешті, для підвищення надійності використовується аналіз солідності контуру – показника його щільності. Висока солідність (понад 0.9) свідчить про цілісність об’єкта і допомагає відрізнити справжні знаки від об’єктів складної форми. Саме ця комбінація методів дозволяє алгоритму демонструвати гнучкість та потенціал до роботи в неідеальних умовах [10, 21].

Звісно, незважаючи на закладені механізми стійкості, реальні дорожні умови завжди представляють певні виклики. Сильні перепади освітлення, такі як глибокі тіні або пряме сонячне світло, можуть ускладнити кольорову

сегментацію. Значне перекриття знаків гілками дерев, іншими автомобілями чи елементами інфраструктури, а також суттєві перспективні спотворення при зйомці під дуже гострим кутом, можуть завадити коректному виділенню контуру та його аналізу. Також завжди існує ймовірність хибних спрацьовувань на об’єктах, що випадково мають схожий колір та форму. Саме тому етап ретельного тестування на різноманітних даних є невід’ємною частиною розробки та оцінки реальної ефективності системи [33, 16, 27].

Отже, підсумовуючи, формалізована задача полягає у розробці та реалізації обчислювального алгоритму D, який, спираючись на методи кольорової сегментації в HSV-просторі та багатоетапної геометричної фільтрації контурів, здатен аналізувати вхідне зображення I та формувати на виході список Ls, що містить координати Bi виявлених знаків початку населеного пункту, прагнучи до максимальної точності, повноти та мінімальної кількості хибних спрацьовувань.

1.3. Огляд методів обробки, сегментації зображень та кластеризації для розв’язання задачі виявлення знаків

Виявлення дорожніх знаків на цифрових зображеннях є складною задачею комп’ютерного зору, яка вимагає послідовного застосування низки методів для перетворення необроблених піксельних даних у значущу інформацію про наявність та розташування об’єктів. Цей процес не є монолітним; він розбивається на кілька фундаментальних етапів, кожен з яких вирішує специфічне підзавдання. До таких етапів належать попередня обробка зображень для покращення їх якості, сегментація для виділення областей інтересу, та подальший аналіз цих областей, що може включати виділення ознак, кластеризацію та, зрештою, класифікацію або локалізацію. Вибір та комбінація методів на кожному етапі суттєво впливають на кінцеву точність та надійність системи розпізнавання, особливо в умовах реальних дорожніх сцен з їхньою варіативністю. Узагальнену послідовність цих етапів та ключові групи методів, що застосовуються на кожному з них, наочно представлено на рисунку 1.3 [2].

Етап попередньої обробки зображень відіграє критичну роль, оскільки його метою є підготовка вхідного зображення до подальшого аналізу шляхом

усунення шумів та покращення візуальних характеристик цільових об’єктів. Шуми, що можуть виникати через особливості сенсора камери, умови освітлення або стиснення даних, ефективно усуваються за допомогою фільтрів, таких як медіанний фільтр, що добре справляється з імпульсним шумом («сіль і перець»), або фільтр Гаусса, який розмиває зображення, зменшуючи високочастотний шум. Важливим кроком є також перетворення колірного простору. Хоча зображення зазвичай отримуються у форматі RGB, для задач, де колір є ключовою ознакою (як у випадку синіх та білих дорожніх знаків), часто переходять до простору HSV (Hue, Saturation, Value). Перевага HSV полягає у тому, що він розділяє колірний тон (Hue) від насиченості (Saturation) та яскравості (Value), що дозволяє виділяти певні кольори більш надійно, навіть при змінах освітлення, які в основному впливають на компонент V. Також можуть застосовуватися методи покращення контрасту, наприклад, гістограмне вирівнювання, щоб зробити знаки більш помітними на фоні [13, 24].

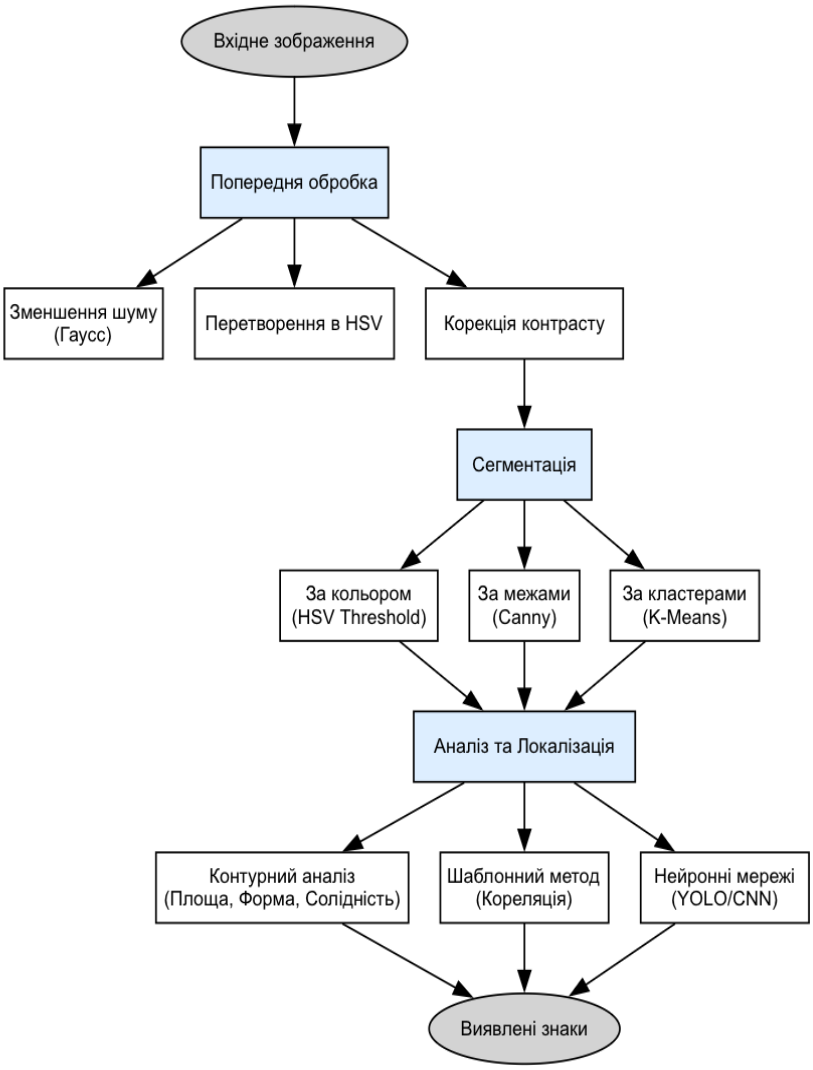

Рисунок 1.3 – Основні етапи та методи в задачах розпізнавання дорожніх знаків

Наступним логічним кроком є сегментація, тобто розділення зображення на значущі області. Для виявлення дорожніх знаків найбільш релевантними є методи, що базуються на кольорі та межах. Кольорова сегментація, особливо в HSV-просторі, є одним з найефективніших підходів для знаків зі стандартизованими кольорами. Вона реалізується шляхом визначення нижньої та верхньої меж для кожної компоненти HSV (H, S, V), що відповідають цільовому кольору (білому або синьому). Застосовуючи функцію cv2.inRange або аналогічну, створюється бінарна маска, де білі пікселі відповідають областям шуканого кольору. Цей підхід дозволяє швидко виділити потенційних

кандидатів. Іншим потужним методом є виділення меж, яке базується на пошуку різких перепадів яскравості. Алгоритм Canny, що є багатоетапним (розмиття, обчислення градієнтів, пригнічення немаксимумів, подвійна порогова фільтрація з гістерезисом), дозволяє отримати тонкі та чіткі контури об’єктів, які можуть точно окреслювати межі дорожніх знаків. Виявлені межі можна далі аналізувати для пошуку замкнених контурів, що відповідають знакам. Іноді ці два підходи (кольоровий та межовий) комбінують для досягнення кращої точності [7, 35].

Кластеризація є методом машинного навчання без вчителя, який дозволяє групувати дані на основі їхньої внутрішньої структури та схожості, без попередньої розмітки. У контексті розпізнавання знаків, кластеризація може знайти застосування на етапі сегментації, де, наприклад, алгоритм K-Means може бути використаний для групування пікселів зображення у K кластерів за їхніми кольоровими значеннями. Це може призвести до більш адаптивної та гнучкої сегментації порівняно з жорстким пороговим методом. Також кластеризація може бути корисною на пізніших етапах для групування виділених об’єктів за їхніми ознаками (розмір, форма, текстура), що може допомогти у подальшій класифікації або відсіюванні хибних спрацьовувань. Важливо розуміти, що кластеризація досліджує природні групи в даних і є цінним інструментом аналізу, хоча в простих системах виявлення знаків вона може не бути основним методом [13, 35].

Після того, як потенційні області знаків виділені (сегментовані), необхідно провести їх аналіз для остаточної локалізації та ідентифікації. Один з найпоширеніших підходів, що використовується і в даній роботі, – це аналіз контурів. Він включає пошук замкнених контурів на бінарній масці або межовому зображенні (cv2.findContours), а потім їх ретельну фільтрацію за геометричними критеріями. Перевірка площі дозволяє відсіяти занадто малі (шум) та занадто великі (фон) контури, а також дає можливість працювати зі знаками на різних відстанях. Аналіз співвідношення сторінобмежуючого прямокутника допомагає відібрати об’єкти з пропорціями, характерними для дорожніх знаків. Застосування апроксимації контуру (cv2.approxPolyDP) дозволяє перевірити, чи є контур чотирикутником, що є ключовою ознакою для

знаків і дозволяє ідентифікувати їх навіть під кутом. Додаткова перевірка на солідність (відношення площі контуру до площі його опуклої оболонки) допомагає відрізнити цілісні прямокутні знаки від об’єктів складної форми. Крім контурного аналізу, існують і структурні методи, що шукають лінії та їх перетини, та порівняння з еталоном, хоча останнє менш стійке до змін. Особливе місце займають згорткові нейронні мережі (CNN), такі як архітектури YOLO та SSD, які революціонізували детектування об’єктів. Вони здатні автоматично вивчати ієрархічні ознаки безпосередньо з зображень і одночасно визначати як місцезнаходження (обмежувальний прямокутник), так і клас об’єкта, демонструючи високу ефективність навіть у складних умовах [24, 35].

Таким чином, для вирішення задачі виявлення дорожніх знаків існує багатий арсенал методів, від класичних алгоритмів обробки зображень до сучасних глибинних нейронних мереж. Вибір конкретного підходу залежить від вимог до точності, швидкості та доступних ресурсів. У даній роботі реалізовано підхід, що поєднує переваги кольорової сегментації в HSV-просторі з ретельною геометричною фільтрацією контурів, як ефективний спосіб розв’язання поставленої задачі.

РОЗДІЛ 2. ВИБІР ТА НАЛАШТУВАННЯ ІНСТРУМЕНТІВ

2.1. Аналіз та обґрунтування вибору програмних засобів для розробки алгоритму розв'язання задачі виявлення знаків початку населеного пункту

Ефективна реалізація інформаційної системи для виявлення знаків початку населеного пункту неможлива без ретельного аналізу та обґрунтованого вибору відповідного стеку програмних технологій. Цей вибір є стратегічно важливим, оскільки він безпосередньо впливає на швидкість розробки, можливості масштабування, продуктивність кінцевого рішення та доступність ресурсів для вирішення потенційних проблем. Основними критеріями при виборі слугували: наявність потужних та спеціалізованих бібліотек для завдань комп'ютерного зору, простота інтеграції різних компонентів системи, висока швидкість виконання обчислювально-інтенсивних операцій з обробки зображень, якість та повнота документації, а також активність та розмір спільноти розробників, що забезпечує підтримку та обмін досвідом [1, 18].

Як основна мова програмування для розробки системи було обрано Python. Це рішення зумовлене численними перевагами Python у контексті наукомістких проєктів та прототипування. По-перше, Python вирізняється своїм чистим, читабельним та лаконічним синтаксисом, що суттєво полегшує процес написання, тестування та підтримки коду, а також дозволяє швидше реалізовувати складні алгоритми. По-друге, Python володіє надзвичайно розвиненою екосистемою сторонніх бібліотек та фреймворків, серед яких особливе місце займають інструменти для наукових обчислень, машинного навчання та, що найважливіше для даної роботи, комп'ютерного зору. Його кросплатформність також є суттєвим плюсом, забезпечуючи можливість запуску розробленої системи на різних операційних системах [18, 20].

Для реалізації основних функцій обробки зображень та алгоритмів комп'ютерного зору була обрана бібліотека OpenCV (Open Source Computer Vision Library). OpenCV є визнаним світовим стандартом у цій галузі, надаючи вичерпний набір функцій та алгоритмів, оптимізованих для роботи в реальному часі. Її відкритий вихідний код, велика спільнота та детальна документація

роблять її привабливим вибором. У рамках даного проєкту були задіяні такі ключові можливості OpenCV: функції для зчитування та запису зображень різних форматів; інструменти для перетворення між колірними просторами, зокрема перехід з RGB в HSV, який є фундаментальним для надійного виділення знаків за їх кольоровими характеристиками в умовах змінного освітлення; реалізація методів пошуку контурів об'єктів (cv2.findContours); функції для аналізу геометричних властивостей знайдених контурів, такі як обчислення площі (cv2.contourArea), побудова обмежувального прямокутника (cv2.boundingRect), апроксимація форми контуру до багатокутника (cv2.approxPolyDP) для перевірки на чотирикутність, та розрахунок опуклої оболонки (cv2.convexHull) для подальшого аналізу солідності форми; а також функції для візуалізації результатів, наприклад, малювання прямокутників на зображенні [4, 20].

Ефективна робота з зображеннями в OpenCV та виконання числових операцій значною мірою забезпечується бібліотекою NumPy (Numerical Python). Зображення в OpenCV представлені у вигляді NumPy-масивів, що дозволяє виконувати швидкі векторні операції над піксельними даними. NumPy є основою для багатьох наукових бібліотек в Python і надає оптимізовані структури даних (багатовимірні масиви) та функції для математичних обчислень, що критично важливо для продуктивності алгоритмів обробки зображень, де маніпуляції з великими обсягами даних є звичайною практикою [12, 20].

Для створення графічного інтерфейсу користувача (GUI), який дозволяє інтерактивно завантажувати зображення та запускати різні алгоритми обробки, була використана бібліотека PyQt5. PyQt5 є набором прив'язок мови Python до потужного кросплатформного фреймворку Qt, що широко використовується для розробки настільних додатків. Вибір PyQt5 обумовлений його гнучкістю, широким набором готових віджетів (таких як кнопки QPushButton, мітки для тексту та зображень QLabel, діалогові вікна для відкриття файлів QFileDialog), можливістю тонкого налаштування зовнішнього вигляду за допомогою таблиць стилів (QSS), а також підтримкою механізму сигналів та слотів для організації взаємодії між елементами інтерфейсу. Це дозволило створити інтуїтивно

зрозумілий інтерфейс для демонстрації та тестування функціональності розробленої системи [12, 26].

Окрім реалізації основного алгоритму виявлення знаків НП на базі класичних методів OpenCV, до складу програмного забезпечення було включено можливість застосування сучасного підходу до детектування об'єктів за допомогою згорткових нейронних мереж. Для цього була інтегрована бібліотека Ultralytics YOLO, яка надає зручний інтерфейс для роботи з популярними моделями YOLO (You Only Look Once). Зокрема, використовується попередньо навчена модель yolov5s.pt, що дозволяє розпізнавати широкий спектр об'єктів без необхідності тривалого процесу навчання власної моделі з нуля. Це включення слугує демонстрацією потужності глибинного навчання та показує потенційний напрямок для подальшого вдосконалення системи, наприклад, шляхом донавчання моделі YOLO на специфічному наборі даних дорожніх знаків для підвищення точності їх розпізнавання. Робота моделей YOLO забезпечується фреймворком для глибинного навчання PyTorch (Torch), який є однією з провідних платформ у цій галузі, відомою своєю гнучкістю та ефективністю [12, 20, 26].

Таким чином, обрана комбінація програмних засобів – Python як основна мова, OpenCV для обробки зображень, NumPy для числових операцій, PyQt5 для графічного інтерфейсу, та Ultralytics YOLO з PyTorch для демонстрації нейромережевих можливостей – формує потужну та гнучку платформу. Цей стек технологій дозволяє ефективно вирішувати поставлені завдання в рамках кваліфікаційної роботи, забезпечуючи як реалізацію основного алгоритму, так і можливості для дослідження більш просунутих підходів у сфері комп'ютерного зору.

2.2. Налаштування середовища розробки та тестування інструментів для реалізації алгоритму

Ефективна розробка інформаційної системи виявлення знаків початку населеного пункту вимагала створення стабільного, функціонального та добре конфігурованого середовища розробки. Цей процес не обмежувався простим

встановленням програмного забезпечення, а включав ретельне налаштування кожного компонента для забезпечення їхньої сумісності, оптимальної продуктивності та створення зручних умов для кодування, відлагодження й всебічного тестування розробленого алгоритму.

Першочерговим кроком стало визначення та розгортання основного інструменту – інтерпретатора мови програмування Python. Для даної роботи була обрана специфічна версія Python 3.11.2, завантажена з офіційного репозиторію python.org. Вибір конкретної версії важливий для забезпечення відтворюваності результатів та уникнення проблем сумісності з бібліотеками, деякі з яких можуть мати строгі вимоги до версії інтерпретатора. Керування пакетами та залежностями здійснювалося за допомогою стандартного для Python інструменту – менеджера пакетів pip. Однак, для запобігання конфліктам між глобально встановленими пакетами та тими, що специфічні для цього проєкту, а також для забезпечення ізоляції та чистоти робочого простору, було створено віртуальне середовище. Це було реалізовано за допомогою вбудованого в Python модуля venv шляхом виконання команди python3 -m venv .venv у кореневій директорії проєкту (де .venv – стандартна назва для папки віртуального середовища). Активація віртуального середовища, що для операційних систем macOS та Linux виконується командою source .venv/bin/activate, стала обов’язковою процедурою перед встановленням будь-яких бібліотек, гарантуючи, що всі пакети будуть інстальовані локально, не впливаючи на системні налаштування Python.

Наступним етапом стало встановлення ключових бібліотек у активоване віртуальне середовище. Кожна бібліотека інсталювалася за допомогою команди pip install <назва_пакету>. Для фундаментальних завдань комп’ютерного зору було встановлено OpenCV (pip install opencv-python). Цей процес, як правило, проходить без ускладнень, але на деяких системах може потребувати наявності компіляторів C++ та відповідних бібліотек розробки, якщо pip вирішує компілювати частину пакету з вихідних кодів. Для ефективних числових операцій та роботи з багатовимірними масивами (які є основою представлення зображень в OpenCV) було інстальовано NumPy (pip install numpy). Для розробки

графічного інтерфейсу користувача було обрано PyQt5 (pip install PyQt5). Також було встановлено бібліотеку Pillow (pip install Pillow), яка надає розширені можливості для роботи із зображеннями і була передбачена для загальних завдань та потенційного використання її функціоналу у майбутньому. Особливу увагу було приділено встановленню фреймворку для глибинного навчання PyTorch та бібліотеки Ultralytics для роботи з YOLO. Встановлення PyTorch (pip install torch torchvision torchaudio) може мати нюанси, пов'язані з вибором версії (CPU-only або з підтримкою CUDA для GPU-прискорення) та сумісністю з драйверами відеокарти, проте для цілей даної кваліфікаційної роботи використовувалася CPU-версія, що спростило процес. Після встановлення PyTorch, інсталяція Ultralytics (pip install ultralytics) зазвичай проходить без проблем. Коректність інсталяції кожної бібліотеки ретельно перевірялася. Це включало не тільки спробу імпорту відповідного модуля в Python (import cv2, import numpy тощо), але й виконання невеликих тестових скриптів для перевірки базового функціоналу: наприклад, завантаження та відображення зображення за допомогою OpenCV, створення та маніпуляція NumPy-масивом, запуск простого вікна PyQt5. Це дозволило переконатися у відсутності проблем з динамічними бібліотеками чи конфігурацією системних шляхів.

Для безпосереднього написання, редагування, аналізу та відлагодження програмного коду було обрано та налаштовано інтегроване середовище розробки (IDE) Visual Studio Code (VS Code). Це середовище було обране за його легкість, гнучкість, потужні інструменти для розробки на Python та широкі можливості кастомізації через розширення. Ключовим було встановлення та налаштування офіційного розширення Python від Microsoft, яке забезпечує інтелектуальне автодоповнення коду (IntelliSense), підсвітку синтаксису, навігацію по коду, а також інтеграцію з відлагоджувачем. Додатково були налаштовані інструменти для підтримки якості коду: лінтер (наприклад, Pylint або Flake8) для автоматичного виявлення потенційних помилок та невідповідностей стандартам кодування, та форматер коду (наприклад, Black або autopep8) для автоматичного приведення коду до єдиного стилю. Важливим аспектом конфігурації VS Code було вказання шляху до інтерпретатора Python,

що знаходиться у створеному раніше віртуальному середовищі .venv, що забезпечило використання коректних залежностей проєкту безпосередньо з IDE. Невід'ємною частиною сучасного процесу розробки є система контролю версій, тому для проєкту було ініційовано локальний репозиторій Git. Це дозволило фіксувати всі зміни у вихідному коді (коміти), створювати гілки для експериментів, відстежувати історію модифікацій та, за необхідності, легко повертатися до попередніх робочих версій коду, що значно підвищує надійність розробки. Приклад робочого простору наведено на рисунку 2.1 [9, 23].

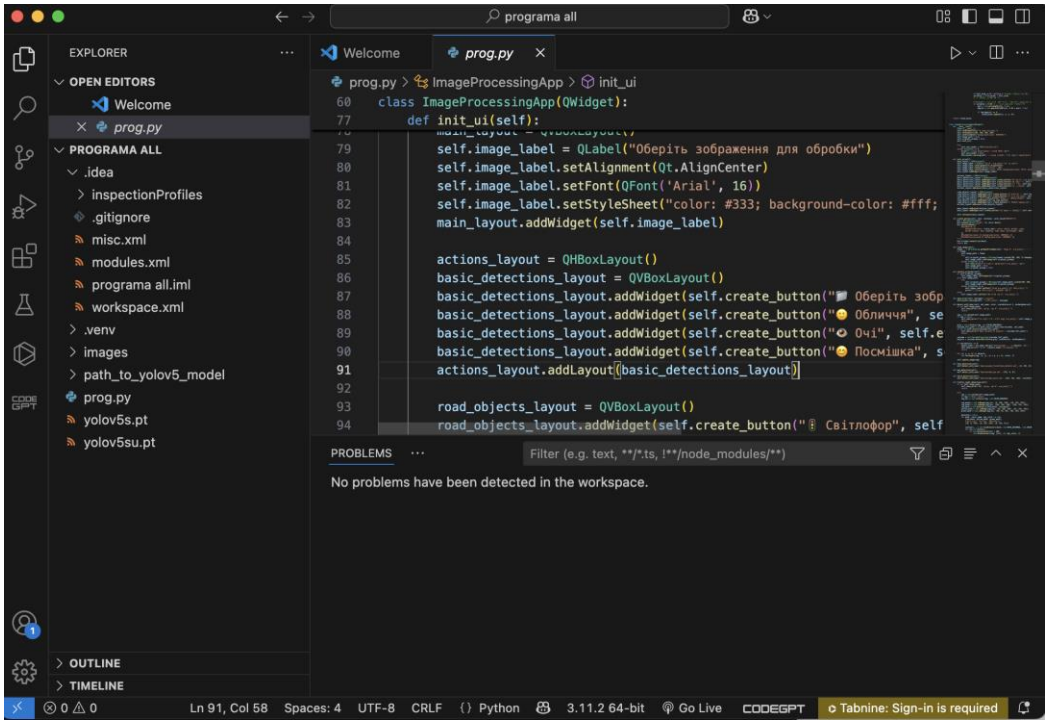

Рисунок 2.1 – Приклад середовища розробки Visual Studio Code з активним віртуальним середовищем проєкту

Паралельно з налаштуванням інструментів розробки здійснювалася підготовка до тестування як самих інструментів, так і розробленого алгоритму. «Тестування інструментів» на цьому етапі полягало у перевірці працездатності кожної встановленої бібліотеки не лише через успішний імпорт, але й шляхом виконання базових операцій, специфічних для кожної з них. Наприклад, для OpenCV це могло бути завантаження тестового зображення, застосування декількох фільтрів та відображення результату; для PyQt5 – створення та запуск мінімального віконного додатку з кількома віджетами; для NumPy – виконання

типових операцій над масивами. Для моделі YOLOv5 перевірялося успішне завантаження файлу ваг (наприклад, yolov5s.pt, який потрібно було попередньо завантажити та розмістити в директорії проєкту або іншому доступному місці) та виконання пробної детекції на одному-двох зображеннях для підтвердження коректної роботи нейромережі. Функціональне тестування окремих розроблених модулів, зокрема функції find_specific_signs, проводилося ітераційно на невеликих, але репрезентативних наборах зображень. Це дозволяло оперативно перевіряти правильність реалізації логіки виявлення знаків, зокрема точність роботи кольорових фільтрів у HSV-просторі та адекватність геометричних критеріїв. Інтеграційне тестування фокусувалося на перевірці коректної взаємодії між графічним інтерфейсом користувача (створеним за допомогою PyQt5) та модулями обробки зображень: завантаження зображень через файловий діалог, виклик відповідних функцій обробки при натисканні кнопок, та адекватне відображення результатів, включаючи оброблене зображення та інформаційні повідомлення. Ключовим для оцінки алгоритму стало використання спеціально сформованого набору тестових зображень, який охоплював широкий спектр можливих сценаріїв: знаки в ідеальних умовах, знаки при різному освітленні (яскраве сонце, тінь, сутінки), частково перекриті знаки, знаки, зняті під різними кутами та з різної відстані, а також зображення, що не містять цільових знаків, для виявлення хибних спрацьовувань. Саме візуальний аналіз результатів на такому наборі дозволив зробити первинні висновки про працездатність та якість роботи розробленого алгоритму [29].

Таким чином, ретельне та багатоетапне налаштування середовища розробки, що включало конфігурацію інтерпретатора Python версії 3.11.2, встановлення та базову перевірку всіх необхідних бібліотек, налаштування інтегрованого середовища розробки Visual Studio Code з відповідними розширеннями, використання системи контролю версій Git, та підготовку до різних рівнів тестування, створило надійний та ефективний робочий простір. Це забезпечило міцний фундамент для подальшої успішної реалізації, відлагодження та аналізу інформаційної системи виявлення знаків початку населеного пункту.

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ЕЛЕМЕНТІВ ЗОБРАЖЕНЬ ДЛЯ ВИЯВЛЕННЯ ЗНАКІВ ПОЧАТКУ НАСЕЛЕНОГО ПУНКТУ

3.1. Розробка алгоритму розв'язання задачі виявлення знаків початку населеного пункту

Розробка ефективного алгоритму для виявлення знаків початку населеного пункту є центральним завданням даної кваліфікаційної роботи. Спираючись на результати теоретичного аналізу (Розділ 1), зокрема формалізовану постановку задачі та огляд існуючих методів, було розроблено послідовний алгоритм, що використовує комбінацію методів обробки зображень для ідентифікації цільових об'єктів – знаків 5.49 (білий фон) та 5.51 (синій фон). Реалізація алгоритму здійснювалася з використанням бібліотеки OpenCV в середовищі Python. Ключові етапи розробленого алгоритму включають підготовку зображення, сегментацію за кольором та ретельну геометричну фільтрацію виділених кандидатів.

Першим кроком є завантаження вхідного зображення та його підготовка до аналізу. Для підвищення надійності виділення кольорових ознак, незалежно від умов освітлення, зображення перетворюється зі стандартного колірного простору BGR (який використовує OpenCV за замовчуванням) у колірний простір HSV (Hue, Saturation, Value). Простір HSV дозволяє ефективніше працювати з колірним тоном (Hue) та насиченістю (Saturation) як окремими характеристиками, що робить процес виділення кольорів більш стійким до варіацій яскравості (Value). У поточній версії алгоритму для спрощення та збереження швидкодії не застосовуються додаткові кроки попередньої обробки, такі як розмиття для зменшення шумів, хоча їх потенційна корисність була відзначена при огляді методів.

Наступний етап – сегментація за кольором. Метою цього етапу є виділення на HSV-зображенні областей, що за своїми кольоровими характеристиками відповідають білим або синім знакам. Для цього визначаються специфічні діапазони значень для компонент H, S та V. Для білих знаків типу 5.49

встановлюється діапазон, що охоплює відтінки з низькою насиченістю та високою яскравістю (наприклад, H: 0-180, S: 0-60, V: 180-255). Для синіх знаків типу 5.51 підбирається діапазон, що відповідає синім відтінкам з достатньою насиченістю та яскравістю (наприклад, H: 100-130, S: 100-255, V: 80-255). Застосовуючи функцію cv2.inRange до HSV-зображення з цими діапазонами, генеруються дві окремі бінарні маски: одна для потенційних білих об’єктів, інша – для синіх. На цих масках пікселі, що відповідають заданому кольору, позначаються білим (максимальне значення), а решта – чорним (нульове значення) [32, 15].

Після отримання бінарних масок здійснюється пошук та фільтрація контурів. На кожній масці за допомогою функції cv2.findContours виявляються всі замкнені контури, які є потенційними кандидатами на дорожні знаки. Оскільки таких контурів може бути багато, включаючи шуми та нерелевантні об’єкти, застосовується послідовна геометрична фільтрація. Спочатку відсіюються контури, площа яких (cv2.contourArea()) не потрапляє у заданий діапазон (наприклад, менше 0.5% від площі зображення, але не менше 300 пікселів, та не більше 95% площі зображення), що дозволяє ігнорувати дуже дрібні та дуже великі об’єкти. Далі, для кожного контуру обчислюється його обмежувальний прямокутник (cv2.boundingRect()) та перевіряється співвідношення сторін (ширини до висоти), яке для цільових знаків зазвичай лежить в межах від 1.5 до 2.8. Наступним важливим кроком є аналіз форми контуру: за допомогою функції cv2.approxPolyDP форма контуру апроксимується до багатокутника, і відбираються лише ті контури, які апроксимуються до чотирикутника (мають 4 вершини). Це дозволяє враховувати невеликі перспективні спотворення знаків, видимих під кутом. Фінальним критерієм фільтрації є солідність контуру, що розраховується як відношення площі контуру до площі його опуклої оболонки (cv2.convexHull()). Для прямокутних знаків цей показник має бути близьким до одиниці (наприклад, > 0.90), що допомагає відсіяти об’єкти складної, непрямокутної форми. Наочно послідовність операцій розробленого алгоритму представлена на блок-схемі (рисунок 3.1).

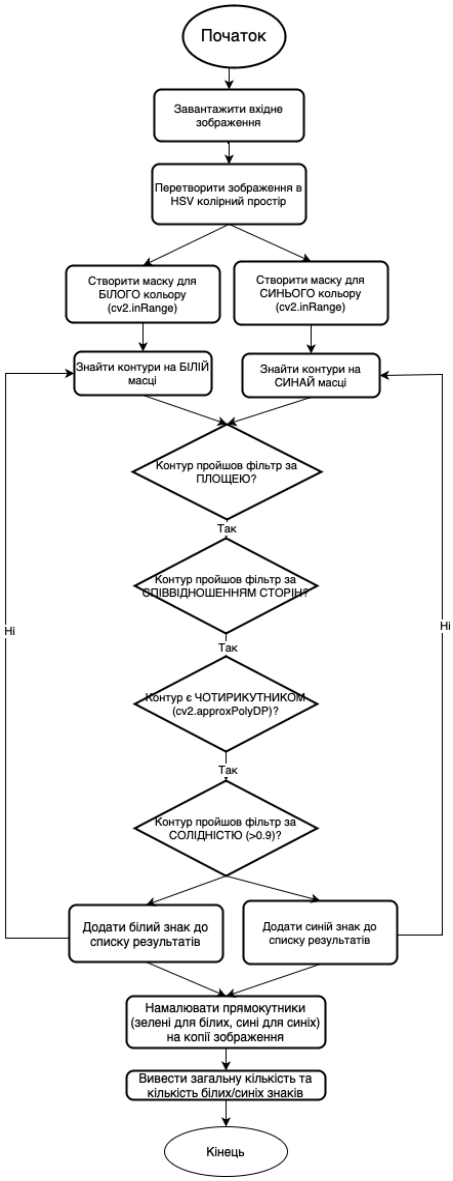

Рисунок 3.1 – Блок-схема алгоритму виявлення знаків початку населеного пункту

Контури, які успішно пройшли всі етапи фільтрації, вважаються виявленими дорожніми знаками. Для кожного такого знаку визначаються координати його обмежувального прямокутника, які й слугують результатом роботи алгоритму. Для візуалізації, знайдені білі знаки позначаються зеленою рамкою, а сині – синьою. Цей алгоритм, завдяки комбінації аналізу кольору в HSV-просторі та багатоетапній геометричній фільтрації, демонструє здатність виявляти цільові знаки в різних умовах, включаючи зміни ракурсу та відстані.

3.2. Впровадження згорткових нейронних мереж у алгоритм розв'язання задачі виявлення знаків

Згорткові нейронні мережі (CNN) репрезентують передовий край технологій у сфері комп'ютерного зору, демонструючи виняткову ефективність у задачах, що пов'язані з аналізом візуальної інформації, зокрема в детектуванні та класифікації об'єктів. На відміну від класичних алгоритмів, які покладаються на вручну визначені ознаки, такі як колір, форма чи текстура, згорткові нейронні мережі здатні автоматично вивчати складні ієрархічні представлення даних безпосередньо з пікселів вхідного зображення. Це досягається завдяки їхній багатошаровій архітектурі, що зазвичай включає згорткові шари для виділення локальних просторових ознак (наприклад, меж, кутів, текстур), шари підвибірки (pooling) для зменшення розмірності карт ознак та забезпечення певної інваріантності до невеликих зсувів чи деформацій об'єкта, та повнозв'язні шари наприкінці мережі для виконання фінальної класифікації або регресії (наприклад, визначення координат обмежувального прямокутника). Саме ця здатність до автоматичного навчання релевантних та високоінформативних ознак робить CNN надзвичайно потужними для роботи з великою варіативністю зображень, що є типовим для реальних дорожніх умов, де освітлення, ракурс, погодні умови та стан об'єктів можуть суттєво змінюватися.

У контексті розроблюваної інформаційної системи для виявлення знаків початку населеного пункту, інтеграція згорткових нейронних мереж може суттєво розширити її можливості та підвищити якість розпізнавання. Хоча основний алгоритм, детально описаний у підрозділі 3.1, використовує детермінований підхід, що базується на аналізі кольорових характеристик в HSV-просторі та геометричних властивостях контурів за допомогою функцій бібліотеки OpenCV, включення функціоналу на базі CNN дозволяє продемонструвати потенціал більш просунутих, керованих даними методів.

В рамках даної кваліфікаційної роботи було вирішено інтегрувати одну з найпопулярніших та найефективніших архітектур для детектування об'єктів в реальному часі – YOLO (You Only Look Once), а саме її п'яту версію (YOLOv5). Ця модель була обрана через її вдале поєднання високої швидкості роботи та

задовільної точності, а також завдяки широкій доступності попередньо навчених ваг, що значно спрощує її практичне використання без необхідності проведення тривалого та обчислювально-ресурсоємного процесу навчання власної моделі з нуля. Для роботи з YOLOv5 в проєкті використовується бібліотека Ultralytics, яка надає зручний та високорівневий Python-інтерфейс для завантаження моделей, їх запуску (здійснення інференсу) та подальшої обробки отриманих результатів. Основою для функціонування бібліотеки Ultralytics YOLO є фреймворк для глибинного навчання PyTorch, який забезпечує всі необхідні інструменти для побудови, тренування та виконання нейронних мереж, включаючи операції з тензорами та можливість використання графічних процесорів (GPU) для прискорення обчислень.

У програмній реалізації інформаційної системи передбачено окрему функцію, яка активується користувачем через відповідний елемент графічного інтерфейсу. При виклику цієї функції для обробки обраного зображення за допомогою YOLOv5 відбуваються наступні кроки. Спочатку система завантажує файл з вагами попередньо навченої моделі, у даному випадку yolov5s.pt. Літера «s» у назві моделі означає «small» – це одна з найменших та, відповідно, найшвидших конфігурацій YOLOv5, що робить її придатною для швидкого тестування та демонстрації на звичайному комп’ютерному обладнанні, яке може не мати потужних спеціалізованих графічних процесорів. Важливо зазначити, що ця модель була навчена на великому та різноманітному наборі даних COCO (Common Objects in Context), який містить близько 80 різних класів повсякденних об’єктів, таких як «людина», «автомобіль», «велосипед», «світлофор», «собака», «кіт» та інші. Після завантаження моделі, вхідне зображення, обране користувачем, передається їй на обробку. Бібліотека Ultralytics автоматично виконує всі необхідні перетворення зображення, включаючи зміну розміру до очікуваного моделлю (наприклад, 640x640 пікселів), нормалізацію значень пікселів та перетворення у відповідний формат тензора. Далі відбувається етап детектування об’єктів (інференс), під час якого модель обробляє підготовлене зображення та повертає список усіх виявлених нею об’єктів. Для кожного такого об’єкта надається детальна інформація:

присвоєний клас об’єкта (наприклад, «car», «person», відповідно до класів з набору COCO) та показник впевненості моделі у правильності цього розпізнавання (числове значення від 0 до 1).

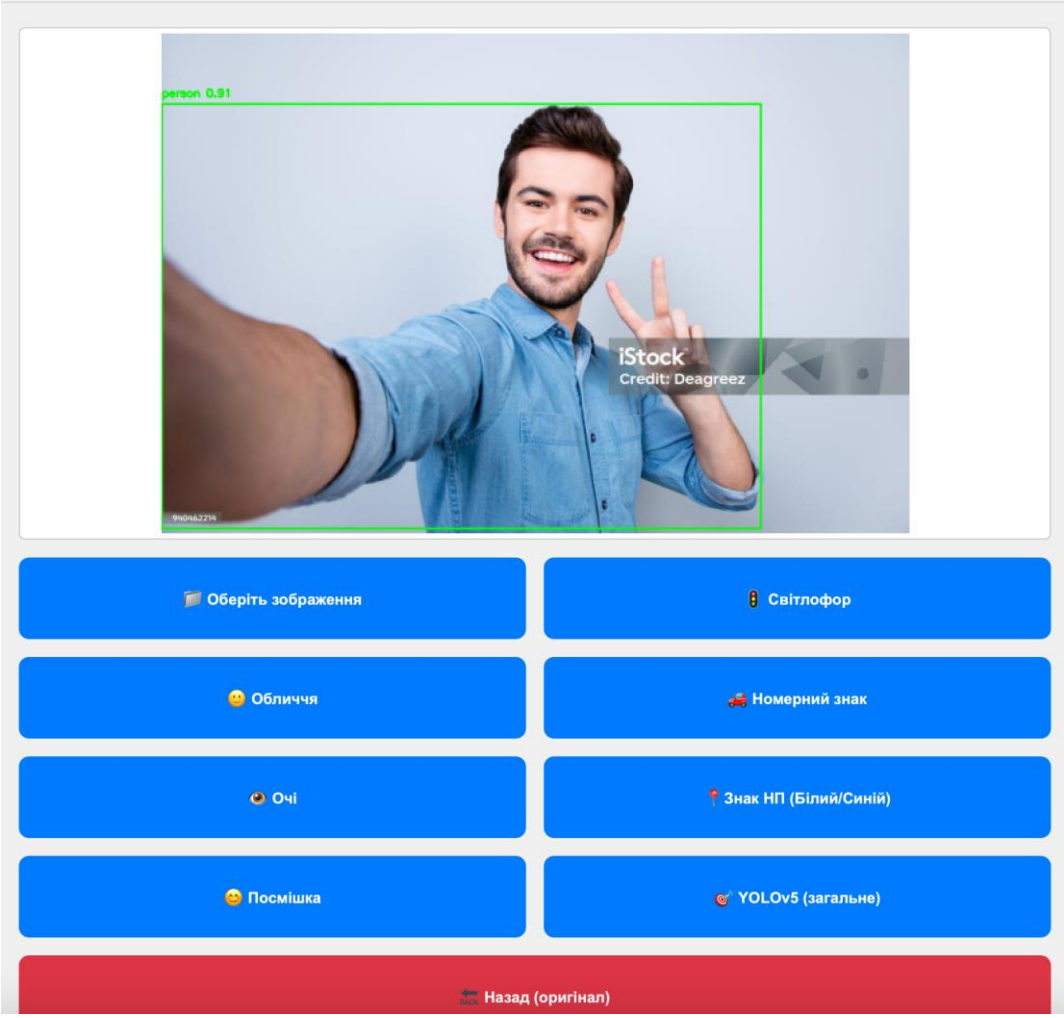

Рисунок 3.2 – Приклад детектування об’єктів на зображенні за допомогою попередньо навченої моделі YOLOv5s у розробленій системі

На завершальному етапі отримані результати візуалізуються: на копії вихідного зображення малюються обмежувальні прямокутники навколо всіх виявлених об’єктів, а поруч з кожним прямокутником відображається назва класу та показник впевненості. Оброблене таким чином зображення з результатами детектування потім відображається у графічному інтерфейсі користувача. Приклад детектування об’єктів показано на рис. 3.2.

Слід особливо підкреслити, що використання попередньо навченої моделі yolov5s.pt на наборі даних COCO у даній інформаційній системі слугує переважно демонстраційною та освітньою метою. Ця конкретна модель не була спеціально навчена розпізнавати дорожні знаки України як окремі класи, і тим більше специфічні знаки «Початок населеного пункту» (5.49 та 5.51). Однак, вона може виявити дорожній знак як деякий загальний «прямокутний об'єкт» або, якщо на знаку присутні чіткі зображення елементів, знайомих моделі (наприклад, силуети людей чи автомобілів на деяких інформаційних щитах), вона може спробувати класифікувати саме ці елементи. Основне значення такого впровадження полягає у можливості наочно продемонструвати принципову здатність сучасних згорткових нейронних мереж здійснювати комплексний аналіз візуальної сцени та виявляти на ній різноманітні об'єкти. Це також дозволяє користувачеві візуально порівняти результати роботи класичного алгоритму, що базується на жорстко визначених правилах аналізу кольору та геометрії за допомогою OpenCV, з результатами, отриманими від універсального детектора об'єктів YOLOv5.

Крім того, наявність інтегрованого та функціонуючого модуля YOLOv5 створює міцний фундамент для можливого майбутнього розвитку та вдосконалення системи. Наприклад, зібравши достатній за обсягом та репрезентативний набір даних, що містить анотовані зображення українських дорожніх знаків (включаючи знаки 5.49 та 5.51), можна було б донавчити (fine-tune) існуючу модель YOLOv5 або навчити меншу, більш спеціалізовану модель для точного та надійного розпізнавання саме цих цільових знаків. Такий підхід, заснований на навчанні з даними, ймовірно, забезпечив би значно вищу точність розпізнавання, більшу стійкість до змін умов освітлення, часткових перекриттів та різноманітних пошкоджень знаків порівняно з поточним алгоритмом, що реалізований на базі OpenCV.

Таким чином, хоча згорткові нейронні мережі у вигляді попередньо навченої моделі YOLOv5 не є основним інструментом для вирішення вузькоспеціалізованої задачі виявлення знаків початку населеного пункту в поточній реалізації інформаційної системи, їх інтеграція є важливим кроком.

Вона дозволяє розширити функціональність системи, продемонструвати користувачеві потужність та гнучкість сучасних технологій глибинного навчання, а також окреслити чіткі та перспективні напрямки для її подальшого розвитку в бік створення більш універсального, точного та надійного рішення для автоматичного аналізу дорожньої інфраструктури.

3.3. Програмна реалізація та тестування інформаційної системи для виявлення знаків початку населеного пункту

Програмна реалізація інформаційної системи, призначеної для виявлення знаків початку населеного пункту, була здійснена з використанням мови програмування Python, версії 3.11.2, та ключових бібліотек, вибір яких детально обґрунтовано у попередньому розділі (підрозділ 2.1). Фундаментом системи є багатоетапний алгоритм обробки зображень, описаний у підрозділі 3.1, що спрямований на ідентифікацію знаків за їхніми кольоровими та геометричними характеристиками, а також інтегрований модуль на основі згорткової нейронної мережі YOLOv5 для демонстрації сучасних підходів до детектування об’єктів (підрозділ 3.2). Вся логіка програми інкапсульована у клас ImageProcessingApp, який успадковується від QWidget бібліотеки PyQt5 і відповідає за створення, управління та функціонування графічного інтерфейсу користувача. Цей клас слугує центральним вузлом, що пов’язує візуальне представлення з обчислювальними модулями.

Графічний інтерфейс користувача (GUI), розроблений за допомогою PyQt5, є основним засобом взаємодії користувача із системою. Його архітектура спроєктована таким чином, щоб забезпечити інтуїтивно зрозумілий доступ до всіх функціональних можливостей. Ключовим елементом є мітка image_label (екземпляр QLabel), призначена для відображення як оригінальних, так і оброблених зображень. Завантаження зображень відбувається через стандартний файловий діалог (QFileDialog), що викликається функцією load_image. Панель інструментів представлена набором кнопок (QPushButton), кожна з яких активує відповідний режим обробки. Стилізація та створення кнопок уніфіковані за допомогою методу create_button. Результати обробки, включаючи візуальні

зміни на зображенні та текстові повідомлення про кількість знайдених об’єктів або помилки, виводяться через методи update_image та show_error (який використовує QMessageBox). Серед реалізованих функцій детектування присутні як класичні підходи на основі каскадів Хаара (для облич, очей, посмішок, номерних знаків), так і методи кольорової сегментації (для світлофорів). Центральне місце займає кнопка «Знак НП (Білий/Синій)», що запускає основний алгоритм виявлення знаків початку населеного пункту. Цей алгоритм реалізований у допоміжній функції find_specific_signs(img, lower_hsv, upper_hsv), яка отримує на вхід зображення у форматі OpenCV та діапазони HSV-кольорів. Всередині цієї функції послідовно виконуються: перетворення зображення в HSV-простір, створення бінарної маски за допомогою cv2.inRange, пошук контурів cv2.findContours, та їх подальша ретельна фільтрація за критеріями площі, співвідношення сторін обмежуючого прямокутника, апроксимація форми до чотирикутника cv2.approxPolyDP та перевірка солідності контуру (відношення площі контуру до площі його опуклої оболонки). Метод detect_settlement_signs_action() у класі ImageProcessingApp організовує виклик find_specific_signs() двічі – окремо для діапазонів білого та синього кольорів, після чого агрегує знайдені обмежувальні прямокутники та візуалізує їх на зображенні, використовуючи зелений колір для білих знаків та синій – для синіх. Інтеграція згорткової нейронної мережі реалізована у методі yolov5_detection(). Цей метод відповідає за завантаження попередньо навченої моделі yolov5s.pt з використанням бібліотеки Ultralytics, передачу їй вхідного зображення для аналізу, отримання результатів (координат обмежувальних прямокутників, передбачених класів об’єктів та відповідних показників впевненості) та їх подальшу візуалізацію на зображенні.

Тестування розробленої інформаційної системи було комплексним і мало на меті не лише виявлення потенційних помилок у програмному коді, але й перевірку відповідності реалізованого функціоналу поставленим вимогам та оцінку якості роботи алгоритмів, особливо основного алгоритму виявлення знаків. Перший етап тестування стосувався графічного інтерфейсу користувача. Ретельно перевірялася працездатність усіх інтерактивних елементів: коректність

завантаження зображень різних поширених форматів (JPEG, PNG), адекватність їх відображення у відведеному вікні, правильність реакції всіх кнопок на натискання, а також чіткість та інформативність діалогових вікон та повідомлень про помилки. Особлива увага приділялася функції відновлення оригінального зображення після серії обробок. Наступним і найбільш критичним був етап функціонального тестування основного алгоритму виявлення знаків початку населеного пункту. Для цього було заздалегідь підготовлено спеціалізований набір тестових зображень. Цей набір включав зображення знаків 5.49 (білих) та 5.51 (синіх) у різноманітних умовах: зняті при хорошому денному освітленні та в умовах сутінків чи тіні; розташовані на різній відстані від камери та під різними кутами огляду (що призводить до перспективних спотворень); зображення знаків з незначними забрудненнями або дефектами. Результати виявлення знаків початку населеного пункту представлені на рис. 3.3.

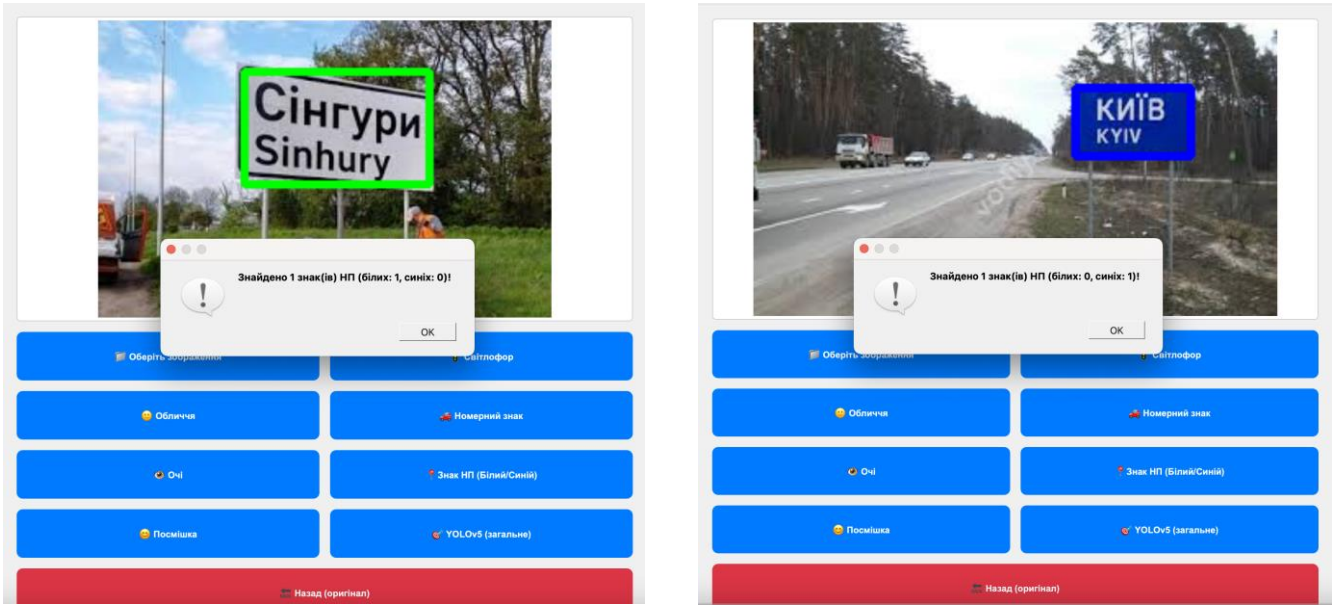

Рисунок 3.3 – Приклад успішного виявлення знаку початку населеного пункту: а) білий знак типу 5.49; б) синій знак типу 5.51

Також до набору входили контрольні зображення, що не містили цільових знаків, але мали об’єкти, схожі за кольором (великі білі або сині поверхні) або формою (інші прямокутні об’єкти, наприклад, рекламні щити), для оцінки стійкості алгоритму до хибних спрацьовувань. Результати роботи алгоритму на зображенні без цільових знаків зображено на рис. 3.4.

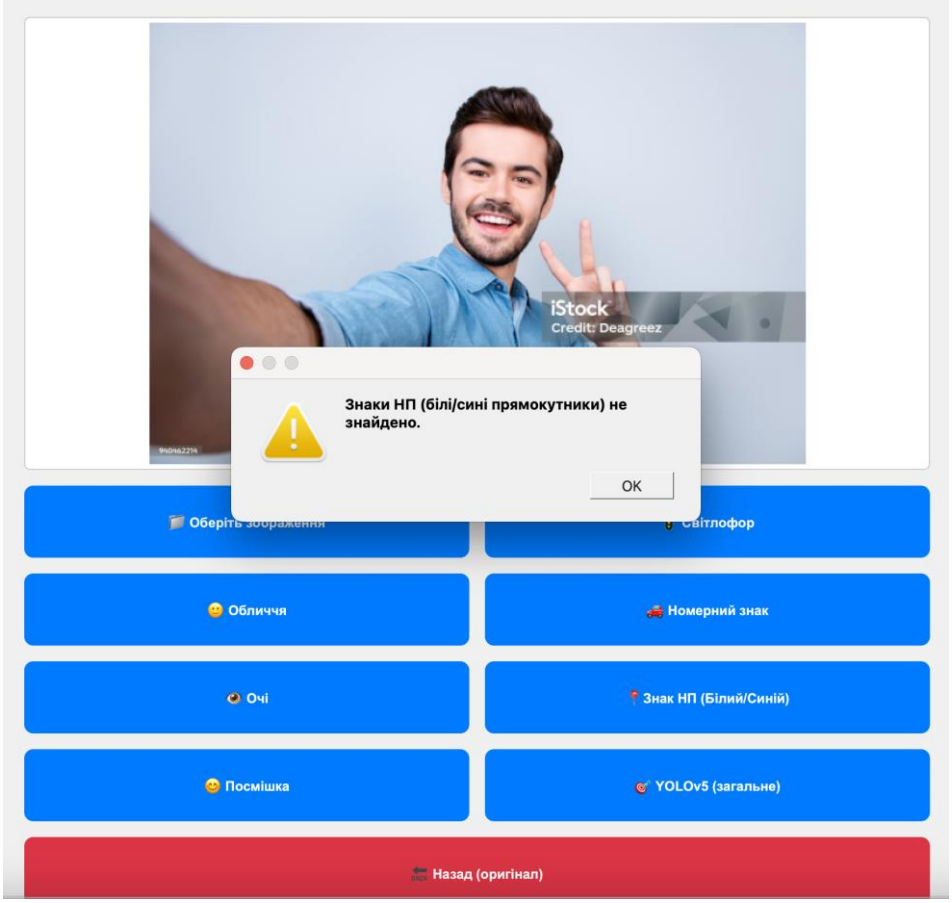

Рисунок 3.4 – Результат роботи алгоритму на зображенні без цільових знаків

На кожному зображенні з цього набору перевірялися такі аспекти: правильність ідентифікації системою кольору знаку (чи відповідає колір рамки типу знайденого знаку), точність обмежувального прямокутника (наскільки щільно він охоплює знак, не захоплюючи зайвого фону і не обрізаючи частину знаку), відсутність пропусків очевидних знаків та мінімізація хибних детектувань. Результати кожного тесту фіксувалися візуально, а також аналізувалися інформаційні повідомлення системи щодо кількості знайдених знаків кожного типу. Проведені тести показали, що алгоритм демонструє найкращу продуктивність на відносно якісних зображеннях з чітко видимими знаками правильної геометричної форми та стандартних кольорів. Було відзначено, що такі фактори, як сильні шуми на зображенні, значні відхилення кольору знаку від еталонного (через вицвітання, бруд, нестандартне освітлення), суттєві перспективні спотворення при зйомці під дуже гострим кутом, а також часткове перекриття знаку можуть призводити до пропусків або неточної локалізації. Разом з тим, впроваджена перевірка на солідність контуру виявилася

ефективною для зменшення кількості хибних спрацьовувань на об’єктах складної форми, які випадково потрапляли в заданий колірний діапазон.

Окремо проводилося тестування додаткових функцій детектування, таких як розпізнавання облич, очей, посмішок та номерних знаків за допомогою каскадів Хаара. На рис. 3.5 зображено приклад роботи функції детектування облич за допомогою каскаду Хаара

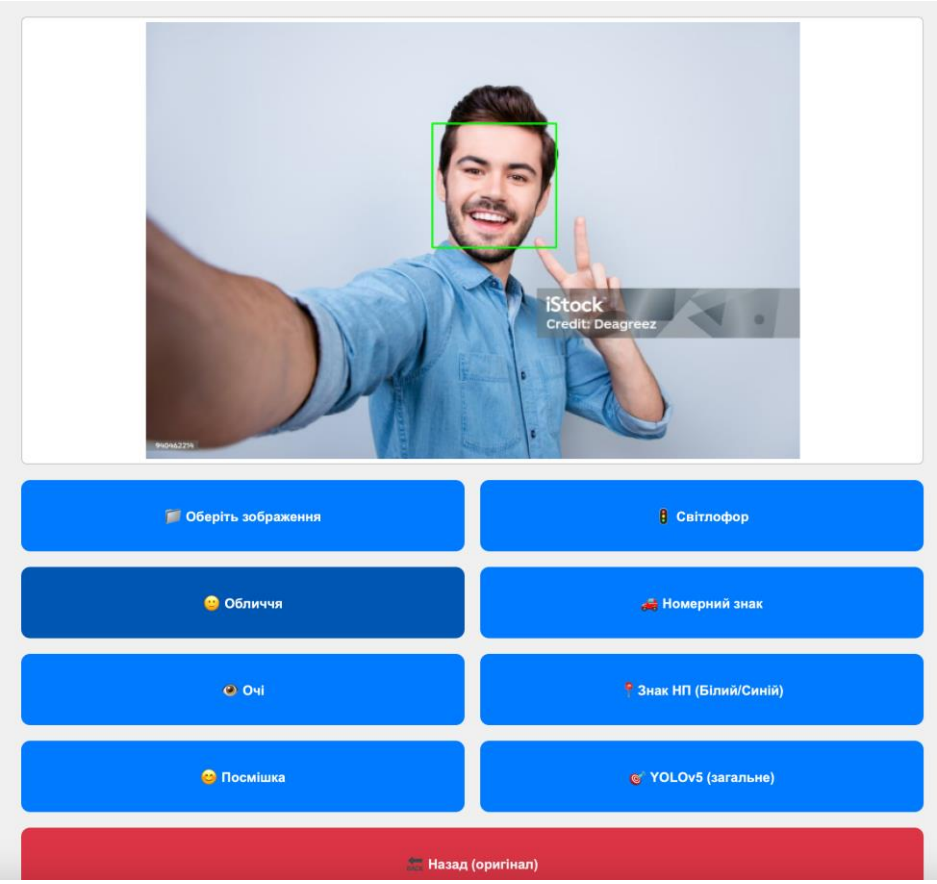

Рисунок 3.5 – Результат успішного детектування обличчя на тестовому зображенні

Для кожної з цих функцій використовувалися відповідні тематичні зображення, і перевірялася їхня загальна працездатність та коректність відображення результатів у рамках створеного графічного інтерфейсу. Щодо модуля YOLOv5, його тестування включало перевірку успішного завантаження моделі yolov5s.pt та її здатності розпізнавати об’єкти із загального набору класів COCO на різноманітних зображеннях, що підтвердило коректну інтеграцію даного нейромережевого компонента в систему. Також під час всього процесу

тестування зверталася увага на обробку помилок та виведення інформативних повідомлень користувачеві, наприклад, у випадках, коли зображення не було обрано перед натисканням кнопки обробки, або якщо виникали проблеми із завантаженням моделі YOLO.

Загалом, проведене тестування підтвердило, що розроблена інформаційна система виконує свої основні функції з виявлення знаків початку населеного пункту, спираючись на реалізований алгоритм аналізу кольору та геометричних параметрів. Було також визначено умови, за яких алгоритм працює найбільш ефективно, та виявлено фактори, що можуть знижувати його точність. Інтеграція та тестування функції детектування за допомогою YOLOv5 успішно продемонстрували можливості сучасних нейромережевих підходів. Детальний аналіз результатів тестування, кількісні та якісні оцінки ефективності алгоритму будуть представлені у наступному розділі даної кваліфікаційної роботи.

РОЗДІЛ 4. ОФОРМЛЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1. Представлення результатів розробки та функціонування інформаційної системи

Дана розрахунково-пояснювальна записка систематизує процес та результати розробки інформаційної системи для виявлення знаків початку населеного пункту. У попередніх розділах було проведено теоретичний аналіз предметної області, сформульовано постановку задачі, здійснено огляд релевантних методів обробки зображень, обґрунтовано вибір програмних засобів та описано налаштування середовища розробки. Розділ 3 детально висвітлив етапи розробки алгоритму виявлення знаків, включаючи методи обробки та сегментації, а також програмну реалізацію системи та первинне тестування її функціоналу. Цей підрозділ присвячений узагальненню підходів до оформлення результатів роботи та їх візуалізації, що є невід'ємною частиною представлення підсумків дослідження.

Для забезпечення наочності та кращого розуміння реалізованих рішень, у тексті пояснювальної записки активно використовуються візуальні матеріали. Блок-схеми застосовуються для ілюстрації як загальної класифікації методів обробки зображень, релевантних для задачі виявлення дорожніх знаків (Рисунок 1.3), так і для детального покрокового опису логіки розробленого алгоритму виявлення знаків початку населеного пункту (Рисунок 3.1). Ці схеми допомагають візуалізувати послідовність операцій та взаємозв'язки між окремими етапами алгоритму. Візуалізація програмної реалізації та середовища розробки представлена скріншотами. Наприклад, Рисунок 2.1 демонструє налаштоване інтегроване середовище розробки Visual Studio Code, що використовувалося для написання та відлагодження програмного коду, з активним віртуальним середовищем проєкту та структурою файлів. Особлива увага приділяється візуалізації безпосередніх результатів роботи інформаційної системи. У підрозділі 3.3, що описує тестування системи, наведені приклади роботи алгоритму на конкретних зображеннях. Зокрема, на рисунку 3.2 представлено приклад детектування об'єкту «person» на тестовому зображенні

за допомогою попередньо навченої моделі YOLOv5s. Цей приклад ілюструє здатність системи використовувати нейромережеві детектори для загального аналізу сцени, що доповнює спеціалізований алгоритм виявлення дорожніх знаків. Рисунок 3.3 ілюструє успішне виявлення та локалізацію знаків початку населеного пункту типів 5.49 (білий фон) та 5.51 (синій фон) за допомогою розробленого алгоритму. Рисунок 3.4 демонструє коректну реакцію системи на зображення, що не містять цільових об’єктів. Також для демонстрації ширшого функціоналу системи та знайомства з сучасними підходами, наведено приклад роботи модуля детектування облич за допомогою каскадів Хаара (Рисунок 3.5) та результати роботи інтегрованої згорткової нейронної мережі YOLOv5 для загального детектування об’єктів на зображенні. Ці скріншоти слугують візуальним підтвердженням працездатності розроблених модулів та дозволяють якісно оцінити результати їх функціонування.

Усі рисунки в тексті записки мають наскрізну нумерацію та супроводжуються відповідними підписами, що розкривають їх зміст. У тексті роботи на кожен рисунок зроблено посилання, що забезпечує логічну зв’язність викладеного матеріалу. Такий підхід до оформлення та візуалізації результатів спрямований на забезпечення максимальної ясності, повноти представлення виконаної роботи та полегшення сприйняття її ключових аспектів, що є важливим для подальшої оцінки ефективності розробленої інформаційної системи.

4.2. Оцінка ефективності алгоритму розв’язання задачі виявлення знаків початку населеного пункту

Ключовим аспектом завершення розробки інформаційної системи є об’єктивна оцінка ефективності її основного компонента – алгоритму виявлення знаків початку населеного пункту. Ця оцінка дозволяє не лише визначити практичну придатність реалізованого підходу, але й виявити його сильні сторони та потенційні обмеження, що є важливим для розуміння можливостей системи та окреслення напрямків її подальшого вдосконалення. Процес оцінки базувався на результатах тестування алгоритму на спеціально підготовленому наборі

зображень, що імітують різноманітні реальні умови, з якими система може зіткнутися під час експлуатації. Для проведення оцінки було використано набір з 30 тестових зображень. Цей набір включав як зображення, що містять цільові дорожні знаки 5.49 (білий фон) та 5.51 (синій фон) за різних умов освітлення, ракурсів зйомки та відстаней до об’єкта, так і зображення, що не містили таких знаків, для перевірки алгоритму на стійкість до хибних спрацьовувань. Кожне зображення з тестового набору було проаналізовано вручну для визначення фактичної наявності та розташування цільових знаків, що слугувало еталоном для порівняння з результатами роботи алгоритму.

Для кількісного вимірювання ефективності алгоритму було застосовано стандартні метрики, що широко використовуються в задачах детектування об’єктів. До таких метрик належать точність (Precision), повнота (Recall) та F1-міра (F1-Score). Точність відображає частку правильно виявлених знаків серед усіх об’єктів, які система класифікувала як знаки, і розраховується за формулою: Precision = TP / (TP + FP), де TP (True Positives) – це кількість правильно виявлених знаків, а FP (False Positives) – кількість об’єктів, помилково ідентифікованих системою як цільові знаки. Високий показник точності свідчить про те, що система генерує мало хибних спрацьовувань. Повнота, у свою чергу, визначає, яку частку з усіх реально присутніх на зображеннях знаків система змогла успішно виявити, і розраховується як: Recall = TP / (TP + FN), де FN (False Negatives) – це кількість реальних знаків, які алгоритм пропустив. Висока повнота вказує на те, що система пропускає мало цільових об’єктів. Оскільки часто існує компроміс між точністю та повнотою (наприклад, підвищення повноти може призвести до зниження точності через збільшення кількості хибних спрацьовувань, і навпаки), для отримання збалансованої оцінки використовується F1-міра, яка є гармонійним середнім цих двох показників: F1 = 2 * (Precision * Recall) / (Precision + Recall).

За результатами тестування на згаданому наборі з 30 зображень, розроблений алгоритм, що базується на кольоровій сегментації в HSV-просторі та подальшій геометричній фільтрації контурів за площею, співвідношенням сторін, формою (апроксимація до чотирикутника) та солідністю,

продемонстрував наступні кількісні показники. Було правильно ідентифіковано 21 цільовий знак, що відповідає TP=21. Важливо відзначити, що під час тестування хибних спрацьовувань, коли система помилково ідентифікувала об’єкт, що не є знаком, як цільовий знак, зафіксовано не було, отже, FP=0. Однак, 9 реальних знаків початку населеного пункту, присутніх на тестових зображеннях, залишилися невиявленими системою, що дає FN=9. Виходячи з цих даних, точність (Precision) алгоритму склала 21 / (21 + 0) = 1.0, або 100%. Такий стовідсотковий показник точності є дуже позитивним результатом і свідчить про високу надійність виявлених кандидатів: усі об’єкти, які інформаційна система ідентифікувала як знаки початку населеного пункту, дійсно ними були. Це мінімізує ризик невірної інтерпретації дорожньої ситуації на основі вихідних даних системи. Показник повноти (Recall) алгоритму становив 21 / (21 + 9) = 0.7, або 70%. Це означає, що система успішно виявляє більшу частину, а саме 70%, всіх реально присутніх на тестових зображеннях цільових знаків. Водночас, 30% знаків залишаються нерозпізнаними, що вказує на наявність простору для подальшого покращення чутливості алгоритму.

Збалансована оцінка, F1-міра, для отриманих результатів становить 2 * (1.0 * 0.7) / (1.0 + 0.7) = 1.4 / 1.7, що є приблизно 0.8235, або 82.4%. Цей показник дає загальне уявлення про ефективність алгоритму, поєднуючи його здатність не робити помилок при детектуванні та знаходити більшість наявних об’єктів.

Якісний аналіз результатів тестування дозволив виявити основні фактори, що впливають на ефективність роботи алгоритму. Найкращі показники детектування та локалізації знаків спостерігалися на зображеннях, що характеризуються хорошим та рівномірним освітленням, чіткою видимістю знаків, відсутністю значних пошкоджень чи забруднень на їх поверхні, а також мінімальним перекриттям сторонніми об’єктами. Розроблений алгоритм продемонстрував свою здатність успішно виявляти знаки, що зняті під помірним кутом огляду та знаходяться на різній відстані від камери. Це свідчить про ефективність застосованих геометричних фільтрів, зокрема апроксимації форми контуру до чотирикутника та аналізу співвідношення сторін і солідності. Випадки пропусків знаків (False Negatives), які призвели до показника повноти у

70%, найчастіше були пов’язані з наступними причинами. По-перше, це складні умови освітлення: наявність глибоких тіней, що падають на знак, або, навпаки, сильні засвіти від прямого сонячного світла суттєво змінювали сприйняття кольорових характеристик пікселів. Це призводило до того, що кольори знаку виходили за межі попередньо визначених діапазонів в HSV-просторі, і, як наслідок, знак не виділявся на етапі кольорової сегментації. По-друге, низька якість вхідного зображення або значна віддаленість знаку також негативно впливали на результат. На дуже віддалених або сильно стиснутих (з втратою якості) зображеннях знаки ставали розмитими, їхні межі втрачали чіткість, що ускладнювало як надійну кольорову сегментацію, так і подальший коректний аналіз форми контуру. По-третє, фізичний стан самих знаків відігравав роль: значне забруднення поверхні знаків або сильне вицвітання фарби спотворювало їхні оригінальні кольори, що також ускладнювало їх виділення. Нарешті, часткове перекриття знаку елементами оточення, такими як гілки дерев, інші дорожні знаки, стовпи або транспортні засоби, навіть якщо перекриття було незначним, могло порушити цілісність контуру знаку. Це, у свою чергу, призводило до того, що такий контур не проходив один або декілька етапів геометричної фільтрації (наприклад, за формою чи солідністю).

Загалом, досягнута точність у 100% є сильним боком розробленого алгоритму, що вказує на його високу специфічність. Показник повноти у 70% свідчить про хорошу базову функціональність, але також вказує на необхідність подальшого вдосконалення алгоритму для підвищення його чутливості та стійкості до несприятливих умов. Можливі напрямки покращення включають адаптивне налаштування порогів для кольорової сегментації, використання більш складних методів попередньої обробки зображень для усунення шумів та покращення контрасту, а також інтеграцію додаткових ознак (наприклад, текстурних) для більш надійної ідентифікації знаків.

ВИСНОВКИ

У ході виконання даної роботи було успішно вирішено актуальну задачу розробки інформаційної системи, призначеної для автоматичного виявлення знаків початку населеного пункту на цифрових зображеннях. Актуальність теми зумовлена зростаючою потребою в автоматизації аналізу дорожньої інфраструктури, що є важливим для розвитку систем допомоги водієві, автономного керування транспортними засобами та оновлення картографічних сервісів.

Основною метою роботи була розробка та програмна реалізація алгоритму, здатного ідентифікувати дорожні знаки 5.49 (білий фон) та 5.51 (синій фон) в умовах, наближених до реальних дорожніх сценаріїв. Для досягнення цієї мети було виконано наступні завдання:

1. Проведено теоретичний аналіз предметної області, що включав вивчення основ теорії розпізнавання образів, формалізацію задачі виявлення знаків та огляд існуючих методів обробки, сегментації зображень.
2. Обґрунтовано вибір програмних засобів, зокрема мови програмування Python та ключових бібліотек OpenCV, PyQt5, NumPy, а також налаштовано середовище розробки Visual Studio Code для реалізації інформаційної системи.
3. Розроблено багатоетапний алгоритм виявлення знаків, що базується на перетворенні зображення в колірний простір HSV, сегментації за кольором для виділення білих та синіх областей, подальшому пошуку контурів та їх ретельній геометричній фільтрації за критеріями площі, співвідношення сторін, апроксимації форми до чотирикутника та солідності.

Здійснено програмну реалізацію розробленого алгоритму у вигляді десктопного додатку з графічним інтерфейсом користувача, який дозволяє завантажувати зображення та візуалізувати результати детектування. Також до системи було інтегровано модуль на основі згорткової нейронної мережі YOLOv5 для демонстрації сучасних підходів до загального детектування

об’єктів. Проведено тестування розробленої системи на наборі з 30 зображень, що дозволило оцінити ефективність основного алгоритму виявлення знаків.

За результатами тестування основного алгоритму було встановлено, що він здатен ідентифікувати цільові знаки з високою точністю (Precision) у 100%, що означає відсутність хибних спрацьовувань серед виявлених об’єктів. Повнота (Recall) алгоритму склала 70%, вказуючи на те, що система виявляє більшість наявних знаків, хоча певна частина залишається нерозпізнаною, переважно через складні умови освітлення, низьку якість зображення або часткове перекриття знаків. Загальна F1-міра ефективності становить 82.4%. Якісний аналіз показав, що алгоритм найбільш ефективний на чітких зображеннях та має потенціал для подальшого покращення шляхом адаптації параметрів та впровадження додаткових методів обробки.

Розроблена інформаційна система демонструє практичну реалізацію підходу, що поєднує класичні методи комп’ютерного зору для вирішення специфічної задачі. Графічний інтерфейс забезпечує зручну взаємодію з користувачем та наочну демонстрацію результатів. Практична значущість виконаної роботи полягає у створенні функціонального прототипу системи, яка може слугувати основою для подальших наукових досліджень у сфері розпізнавання дорожніх знаків, а також для розробки більш комплексних та інтелектуальних рішень для аналізу дорожньої інфраструктури. Отримані результати підтверджують життєздатність обраного підходу та визначають напрямки для його потенційного вдосконалення, такі як підвищення стійкості до складних умов або інтеграція з нейромережевими моделями, спеціально навченими на розпізнавання дорожніх знаків України. Таким чином, поставлені у кваліфікаційній роботі завдання були виконані, а мета – досягнута. Розроблено та протестовано інформаційну систему, що реалізує алгоритм виявлення знаків початку населеного пункту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Виноградeнко С. О., Сєдов А. О. Інформаційна система земельних ресурсів: метод. рекомендації. Харків: ХНУРЕ, 2024. URL: https://repo.btu.kharkov.ua/handle/123456789/58966.
2. Владика Д. Я. М. Розробка системи розпізнавання емоцій людини із використанням алгоритмів машинного навчання. Львів: ЛНАУ, 2024. URL: https://repository.lnau.edu.ua/xmlui/handle/123456789/977.
3. Говоруха М. А. Система розпізнавання дорожніх знаків з відеореєстратора та моніторингу поведінки водія. Київ: КПІ ім. Ігоря Сікорського, 2024. URL: https://ela.kpi.ua/items/38de5a69-c4f4-4513-a897-1555a1657bad.
4. Дикун А. В. Система розпізнавання дорожніх знаків на автомобілі. Київ: КПІ ім. Ігоря Сікорського, 2023. URL: https://ela.kpi.ua/items/9b0069be-6dee-4175-b305-ca2cc85bddc4.
5. Дичка А. І. Алгоритмічне та програмне забезпечення процесів автоматичної ідентифікації на основі багатоколірних завадостійких штрихових кодів у медичних інформаційних системах. Київ: КПІ ім. Ігоря Сікорського, 2023. URL: https://ela.kpi.ua/items/66eca83a-cf7e-465f-8d82-3711d7729640.
6. Кобилін О. А., Творошенко І. С. Методи цифрової обробки зображень. Науковий журнал, 2021. URL: https://scholar.google.com.ua/scholar?start=20&q=ІНФОРМАЦІЙНІ+СИСТЕМИ+РОЗПІЗНАВАННЯ+ЕЛЕМЕНТІВ+ЗОБРАЖЕНЬ+ДЛЯ+ВИЯВЛЕННЯ+ЗНАКІВ&hl=uk&as_sdt=0,5&as_ylo=2021#d=gs_cit&t=1748910081852&u=%2Fscholar%3Fq%3Dinfo%3Amz0Jq0QjVygJ%3Ascholar.google.com%2F%26output%3Dcite%26scirp%3D23%26hl%3Duk.
7. Кондратенко К. В. Інформаційна система розпізнавання тексту з використанням штучних нейронних мереж. Суми: СумДУ, 2024. URL: https://essuir.sumdu.edu.ua/handle/123456789/97002.
8. Коновалов О. Ю. Розробка підсистеми розпізнавання дорожніх знаків автомобільного автопілота. Харків: ХНУРЕ, 2021. URL:

https://openarchive.nure.ua/entities/publication/34e99853-e683-46eb-a635-81ac295ef621.

9. Коропченко С. В. Інформаційна система пошуку і розпізнавання об’єкта зацікавленості. Суми: СумДУ, 2020. URL: https://essuir.sumdu.edu.ua/handle/123456789/80061.
10. Литвин В., Ринков А., Висоцька В. Інтелектуальна інформаційна система автоматичної генерації умовних знаків у стандарті APP-6D. Науковий журнал, 2023. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2023/jul/30928/n30487maket-302-330_0.pdf.
11. Лозицький А. О. Сегментація зображень в реальному часі для автономних транспортних засобів. Харків: ХНУРЕ, 2025. URL: https://openarchive.nure.ua/entities/publication/c9a5b137-740f-4a33-95aa-8b4b828350dc.
12. Максаков А. В. Інформаційна система з розпізнавання дорожніх знаків, розмітки та учасників дорожнього руху. Київ: КПІ ім. Ігоря Сікорського, 2021. URL: https://ela.kpi.ua/bitstreams/014feaa8-a550-432d-afa8-4cbf572873c7/download.
13. Марченков І. Д. Інформаційна система розпізнавання тексту технології класифікації додатків до атестатів. Київ: КПІ ім. Ігоря Сікорського, 2020. URL: https://ela.kpi.ua/server/api/core/bitstreams/6447c960-d0ca-4270-ac98-642c365e0a72/content.
14. Мулік Р. Ю. Програмна система для розпізнавання дорожніх знаків з використанням машинного навчання. Київ: КПІ ім. Ігоря Сікорського, 2023. URL: https://ela.kpi.ua/items/36eead1e-421c-4082-909a-62b8bf288694.
15. Підгорний П. В. Інформаційна система розпізнавання графічних образів з використанням нейронної мережі: магістр. дис. Суми: Сумський державний університет, 2022. URL: https://essuir.sumdu.edu.ua/handle/123456789/88600.
16. Редчук О. І. Дослідження здатності штучного інтелекту до розпізнавання жестової мови: магістр. дис. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2023. URL: https://elartu.tntu.edu.ua/handle/lib/44480.

17. Савчишин Я. С. Інформаційна система розпізнавання дорожніх знаків із використанням нейромережевих інструментів TensorFlow: бакалавр. дис. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2023. URL: https://elartu.tntu.edu.ua/handle/lib/41820.
18. Слатін М. Ю. Розроблення та дослідження нейронної мережі для розпізнавання дорожніх знаків. Харків: ХНУРЕ, 2024. URL: https://openarchive.nure.ua/entities/publication/7256e11e-4f3c-4d92-8f82-db93c0c3980f.
19. Темчур К. О. Дослідження методів розпізнавання дорожніх знаків України. Харків: ХНУРЕ, 2022. URL: https://openarchive.nure.ua/entities/publication/c9778cf5-6834-4067-9880-3a95f82e4f0c.
20. Чайка Б. В. Інформаційна система розпізнавання рухів у карате: магістр. дис. Суми: Сумський державний університет, 2023. URL: https://essuir.sumdu.edu.ua/handle/123456789/94733.
21. Шматко О. В., Голоскокова А. О., Мілевський С. В., Воропай Н. І. Інформаційна система розпізнавання зображень. Системи озброєння і військова техніка. 2021. № 4 (68). С. 130–137. URL: https://scholar.archive.org/work/evf722m7yvdthop4w4f4qzrrxm/access/wayback/https://journal-hnups.com.ua/index.php/soivt/article/download/986/865.
22. Abdullayeva G. G., Alizadeh U. M. An information recognition system for complex images. Adcaij-advances in distributed computing and artificial intelligence journal. 2020. P. 79–93. URL: https://www.torrossa.com/en/resources/an/4658314#page=81.
23. Ardianto S., Chen C. J., Hang H. M. Real-time traffic sign recognition using color segmentation and SVM. 2017 International Conference on Systems, Signals and Image Processing (IWSSIP). IEEE, 2017. P. 1–5. URL: https://ieeexplore.ieee.org/abstract/document/7965570/.

24. Bahlmann C., Zhu Y., Ramesh V., Pellkofer M., Koehler T. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information. IEEE Proceedings. Intelligent Vehicles Symposium, 2005. IEEE, 2005. P. 255–260. URL: https://ieeexplore.ieee.org/abstract/document/1505111/.
25. Daradkeh Y. I., Tvoroshenko I., Gorokhovatskyi V., Latiff L. A., Ahmad N. Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic. IEEE Access. 2021. Vol. 9. P. 13417–13428. URL: https://ieeexplore.ieee.org/abstract/document/9324755/.
26. Garcia-Garrido M. A., Sotelo M. A., Martin-Gorostiza E. Fast traffic sign detection and recognition under changing lighting conditions. 2006 IEEE Intelligent Transportation Systems Conference. IEEE, 2006. P. 811–816. URL: https://ieeexplore.ieee.org/abstract/document/1706843/.
27. Kanagaraj N., Hicks D., Goyal A., Tiwari S., Singh G. Deep learning using computer vision in self driving cars for lane and traffic sign detection. International Journal of System Assurance Engineering and Management. 2021. Vol. 12, № 6. P. 1011–1025. URL: https://link.springer.com/article/10.1007/s13198-021-01127-6.
28. Kaur J., Singh W. Tools, techniques, datasets and application areas for object detection in an image: a review. Multimedia Tools and Applications. 2022. Vol. 81, № 27. P. 38297–38351. URL: https://link.springer.com/article/10.1007/s11042-022-13153-y.
29. Kim J. H., Kim N., Park Y. W., Won C. S. Object detection and classification based on YOLO-V5 with improved maritime dataset. Journal of Marine Science and Engineering. 2022. Vol. 10, № 3. P. 377. URL: https://www.mdpi.com/2077-1312/10/3/377.
30. Lahmyed R., Ansari M. E., Kerkaou Z. Automatic road sign detection and recognition based on neural network. Soft Computing. 2022. Vol. 26, № 4. P. 1743–1764. URL: https://link.springer.com/article/10.1007/s00500-021-06726-w.

31. Li H., Sun F., Liu L., Wang L. A novel traffic sign detection method via color segmentation and robust shape matching. Neurocomputing. 2015. Vol. 169. P. 77–88. URL: https://www.sciencedirect.com/science/article/abs/pii/S0925231215006712.
32. Shadeed W. G., Abu-Al-Nadi D. I., Mismar M. J. Road traffic sign detection in color images. 10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003. IEEE, 2003. Vol. 2. P. 890–893. URL: https://ieeexplore.ieee.org/abstract/document/1301930/.
33. Soendoro D., Supriana I. Traffic sign recognition with Color-based Method, shape-arc estimation and SVM. Proceedings of the 2011 International Conference on Electrical Engineering and Informatics. IEEE, 2011. P. 1–6. URL: https://ieeexplore.ieee.org/abstract/document/6021584/.
34. Willman J. Overview of pyqt5. Modern PyQt: Create GUI Applications for Project Management, Computer Vision, and Data Analysis. Berkeley, CA: Apress, 2020. P. 1–42. URL: https://link.springer.com/chapter/10.1007/978-1-4842-6603-8_1.
35. Zhang Y., Guo Z., Wu J., Tian Y., Tang H., Guo X. Real-time vehicle detection based on improved yolo v5. Sustainability. 2022. Vol. 14, № 19. P. 12274. URL: https://www.mdpi.com/2071-1050/14/19/12274.

ДОДАТКИ

Додаток А

Лістинг файлу `prog.py`

```
    import sys
    import os
    import cv2
    import numpy as np
    from PyQt5.QtWidgets import (QApplication, QWidget,
QVBoxLayout, QPushButton,
                                 QLabel, QFileDialog, QHBoxLayout,
QMessageBox)
    from PyQt5.QtGui import QPixmap, QImage, QFont
    from PyQt5.QtCore import Qt
    from PIL import Image, ImageDraw, ImageFont
    import torch
    from ultralytics import YOLO

    def find_specific_signs(img, lower_hsv, upper_hsv):
        if img is None:
            return []

        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, lower_hsv, upper_hsv)

        contours, hierarchy = cv2.findContours(mask,
cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)

        original_height, original_width = img.shape[:2]
        found_boxes = []

        if hierarchy is not None:
            for i, cnt in enumerate(contours):
                if hierarchy[0][i][3] == -1:
                    area = cv2.contourArea(cnt)
                    min_area = original_width * original_height *
0.005
                    max_area = original_width * original_height *
0.95

                    if min_area < area < max_area:
                        x, y, w, h = cv2.boundingRect(cnt)
                        if h == 0: continue
                        aspect_ratio = float(w) / h
```

```
                        if 1.5 < aspect_ratio < 2.8:
                            hull = cv2.convexHull(cnt)
                            hull_area = cv2.contourArea(hull)
                            if hull_area == 0: continue
                            solidity = float(area) / hull_area

                            if solidity > 0.90:
                                peri = cv2.arcLength(cnt, True)
                                approx = cv2.approxPolyDP(cnt,
0.04 * peri, True)

                                if len(approx) == 4:
                                    found_boxes.append((x, y, w,
h))
        return found_boxes

    class ImageProcessingApp(QWidget):
        def __init__(self):
            super().__init__()
            self.setWindowTitle("Обробка зображень")
            self.setGeometry(100, 100, 850, 650)
            self.setStyleSheet("background-color: #f0f0f0;")
            self.image_path = None
            self.original_pixmap = None
            self.init_ui()

            try:
                self.yolo_model = YOLO("yolov5s.pt")
            except Exception as e:
                print(f"Помилка завантаження моделі YOLO: {e}")
                self.yolo_model = None
                QMessageBox.warning(self, "Помилка моделі", f"Не
вдалося завантажити модель YOLOv5s.pt: {e}")

        def init_ui(self):
            main_layout = QVBoxLayout()
            self.image_label = QLabel("Оберіть зображення для
обробки")
            self.image_label.setAlignment(Qt.AlignCenter)
            self.image_label.setFont(QFont('Arial', 16))
            self.image_label.setStyleSheet("color: #333;
background-color: #fff; border: 1px solid #ccc; border-radius:
5px; min-height: 400px;")
            main_layout.addWidget(self.image_label)
```

```
            actions_layout = QHBoxLayout()
            basic_detections_layout = QVBoxLayout()

basic_detections_layout.addWidget(self.create_button("□ Оберіть
зображення", self.load_image))

basic_detections_layout.addWidget(self.create_button("□ Обличчя",
self.face_detection))

basic_detections_layout.addWidget(self.create_button("□ Очі",
self.eye_detection))

basic_detections_layout.addWidget(self.create_button("□ Посмішка",
self.smile_detection))
            actions_layout.addLayout(basic_detections_layout)

            road_objects_layout = QVBoxLayout()
            road_objects_layout.addWidget(self.create_button("□
Світлофор", self.traffic_light_detection))
            road_objects_layout.addWidget(self.create_button("□
Номерний знак", self.license_plate_detection))
            self.np_sign_button = self.create_button("□Знак НП
(Білий/Синій)", self.detect_settlement_signs_action)
            road_objects_layout.addWidget(self.np_sign_button)
            road_objects_layout.addWidget(self.create_button("□
YOLOv5 (загальне)", self.yolov5_detection))
            actions_layout.addLayout(road_objects_layout)

            main_layout.addLayout(actions_layout)
            main_layout.addWidget(self.create_button("□ Назад
(оригінал)", self.restore_original, "#dc3545"))

            self.setLayout(main_layout)

        def create_button(self, text, callback,
color_hex="#007bff"):
            btn = QPushButton(text)
            btn.setFont(QFont("Arial", 12, QFont.Bold))
            btn.setStyleSheet(f"""
                QPushButton {{
                    background-color: {color_hex}; color: white;
border: none;
                    border-radius: 8px; padding: 12px 15px; min-
height: 40px;
                }}
```

```
                QPushButton:hover {{ background-color: #0056b3; }}
                QPushButton:pressed {{ background-color: #004085;
}}
            """)
            btn.clicked.connect(callback)
            return btn

        def load_image(self):
            fname, _ = QFileDialog.getOpenFileName(self, 'Виберіть
зображення', '', 'Image files (*.jpg *.png *.jpeg)')
            if fname:
                self.image_path = fname
                try:
                    self.original_pixmap =
QPixmap(fname).scaled(700, 500, Qt.KeepAspectRatio,
Qt.SmoothTransformation)

self.image_label.setPixmap(self.original_pixmap)
                except Exception as e:
                    self.show_error(f"Не вдалося завантажити
зображення: {e}")
                    self.image_path = None
                    self.original_pixmap = None

        def restore_original(self):
            if self.original_pixmap:
                self.image_label.setPixmap(self.original_pixmap)
            elif self.image_path:
                try:
                    self.original_pixmap =
QPixmap(self.image_path).scaled(700, 500, Qt.KeepAspectRatio,
Qt.SmoothTransformation)

self.image_label.setPixmap(self.original_pixmap)
                except Exception as e:
                    self.image_label.setText("Немає зображення для
відновлення.")
                    print(f"Помилка відновлення оригіналу: {e}")
            else:
                self.image_label.setText("Спочатку оберіть
зображення.")

        def show_error(self, message="Помилка"):
            QMessageBox.warning(self, "Повідомлення", message)
```

```
        def detect_with_haar(self, xml_name, color,
scaleFactor=1.1, minNeighbors=5):
            if not self.image_path:
                self.show_error("Будь ласка, оберіть зображення.")
                return

            img = cv2.imread(self.image_path)
            if img is None:
                self.show_error(f"Не вдалося прочитати файл
зображення: {self.image_path}")
                return

            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            cascade_full_path =
os.path.join(cv2.data.haarcascades, xml_name)
            if not os.path.exists(cascade_full_path):
                self.show_error(f"Файл каскаду не знайдено:
{cascade_full_path}")
                return

            cascade = cv2.CascadeClassifier(cascade_full_path)
            objects = cascade.detectMultiScale(gray, scaleFactor,
minNeighbors)

            if len(objects) == 0:
                object_name = xml_name.replace('haarcascade_',
'').replace('.xml', '').replace('_default', '')
                self.show_error(f"Об'єкт ({object_name}) не
знайдено.")
                return

            for (x, y, w, h) in objects:
                cv2.rectangle(img, (x, y), (x + w, y + h), color,
2)

            self.update_image(img)

        def face_detection(self):

self.detect_with_haar('haarcascade_frontalface_default.xml', (0,
255, 0))

        def eye_detection(self):
            self.detect_with_haar('haarcascade_eye.xml', (255, 0,
0))
```

```
        def smile_detection(self):
            self.detect_with_haar('haarcascade_smile.xml', (255,
192, 203), scaleFactor=1.7, minNeighbors=22)

        def traffic_light_detection(self):
            if not self.image_path:
                self.show_error("Будь ласка, оберіть зображення.")
                return

            try:
                img = cv2.imread(self.image_path)
                if img is None: return
                img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

                red_mask1 = cv2.inRange(img_hsv, (0, 100, 100),
(10, 255, 255))
                red_mask2 = cv2.inRange(img_hsv, (160, 100, 100),
(180, 255, 255))
                red_mask = cv2.bitwise_or(red_mask1, red_mask2)
                yellow_mask = cv2.inRange(img_hsv, (20, 100, 100),
(35, 255, 255))
                green_mask = cv2.inRange(img_hsv, (40, 50, 50),
(85, 255, 255))

                detected = False
                for mask, color_name, bgr_color in zip(
                    [red_mask, yellow_mask, green_mask],
                    ["Червоний", "Жовтий", "Зелений"],
                    [(0, 0, 255), (0, 255, 255), (0, 255, 0)]):

                    contours, _ = cv2.findContours(mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                    for cnt in contours:
                        if cv2.contourArea(cnt) > 400:
                            cv2.drawContours(img, [cnt], -1,
bgr_color, 2)
                            x, y, w, h = cv2.boundingRect(cnt)
                            cv2.putText(img, color_name, (x, y -
5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, bgr_color, 2)
                            detected = True

                if not detected:
                    self.show_error("Світлофор не знайдено.")
                    return

                self.update_image(img)
```

```
            except Exception as e:
                self.show_error(f"Помилка при обробці світлофора:
{e}")

        def license_plate_detection(self):
            if not self.image_path:
                self.show_error("Будь ласка, оберіть зображення.")
                return

            try:
                img = cv2.imread(self.image_path)
                if img is None: return
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

                cascade_full_path =
os.path.join(cv2.data.haarcascades,
'haarcascade_russian_plate_number.xml')
                if not os.path.exists(cascade_full_path):
                    self.show_error("Каскад для номерних знаків не
знайдено.")
                    return

                plate_cascade =
cv2.CascadeClassifier(cascade_full_path)
                plates = plate_cascade.detectMultiScale(gray, 1.1,
10)

                if len(plates) == 0:
                    self.show_error("Номерний знак не знайдено.")
                    return

                for (x, y, w, h) in plates:
                    cv2.rectangle(img, (x, y), (x + w, y + h), (0,
0, 255), 2)

                self.update_image(img)
            except Exception as e:
                self.show_error(f"Помилка при обробці номерного
знаку: {e}")

        def draw_text_with_pil(self, img_cv, text, position,
font_size=28, color=(255, 0, 0)):
            img_pil = Image.fromarray(cv2.cvtColor(img_cv,
cv2.COLOR_BGR2RGB))
            draw = ImageDraw.Draw(img_pil)
            try:
```

```
                font = ImageFont.truetype("Arial.ttf", font_size)
            except IOError:
                font = ImageFont.load_default()
            draw.text(position, text, font=font, fill=color)
            return cv2.cvtColor(np.array(img_pil),
cv2.COLOR_RGB2BGR)

        def yolov5_detection(self):
            if not self.image_path:
                self.show_error("Будь ласка, оберіть зображення.")
                return

            if not self.yolo_model:
                self.show_error("Модель YOLOv5 не завантажена.")
                return

            try:
                results = self.yolo_model(self.image_path)
                img = cv2.imread(self.image_path)
                if img is None: return

                found_objects = False
                for box in results[0].boxes:
                    found_objects = True
                    x1, y1, x2, y2 = map(int, box.xyxy[0])
                    conf = box.conf[0]
                    cls = int(box.cls[0])
                    label = f'{self.yolo_model.names[cls]}
{conf:.2f}'
                    cv2.rectangle(img, (x1, y1), (x2, y2), (0,
255, 0), 2)
                    cv2.putText(img, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 2)

                if not found_objects:
                    self.show_error("YOLOv5 не виявила об'єктів.")
                    return

                self.update_image(img)
            except Exception as e:
                self.show_error(f"Помилка при обробці YOLOv5:
{e}")

        def detect_settlement_signs_action(self):
            if not self.image_path:
                self.show_error("Будь ласка, оберіть зображення.")
```

```
                return

            try:
                img = cv2.imread(self.image_path)
                if img is None:
                    self.show_error("Не вдалося завантажити
зображення для обробки.")
                    return

                img_for_drawing = img.copy()

                lower_white = np.array([0, 0, 180])
                upper_white = np.array([180, 60, 255])

                lower_blue = np.array([100, 100, 80])
                upper_blue = np.array([130, 255, 255])

                white_boxes = find_specific_signs(img,
lower_white, upper_white)
                blue_boxes = find_specific_signs(img, lower_blue,
upper_blue)

                total_count = len(white_boxes) + len(blue_boxes)

                if total_count == 0:
                    self.show_error("Знаки НП (білі/сині
прямокутники) не знайдено.")
                    self.update_image(img)
                else:
                    for (x, y, w, h) in white_boxes:
                        cv2.rectangle(img_for_drawing, (x, y), (x
+ w, y + h), (0, 255, 0), 3)

                    for (x, y, w, h) in blue_boxes:
                        cv2.rectangle(img_for_drawing, (x, y), (x
+ w, y + h), (255, 0, 0), 3)

                    QMessageBox.information(self, "Результат",
f"Знайдено {total_count} знак(ів) НП (білих: {len(white_boxes)},
синіх: {len(blue_boxes)})!")
                    self.update_image(img_for_drawing)

            except Exception as e:
                self.show_error(f"Помилка при пошуку знаків НП:
{e}")
                import traceback
```

```
                traceback.print_exc()

        def update_image(self, image):
            try:
                if len(image.shape) == 2 or image.shape[2] == 1:
                    image = cv2.cvtColor(image,
cv2.COLOR_GRAY2BGR)

                rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                h, w, ch = rgb_image.shape
                bytes_per_line = ch * w
                q_image = QImage(rgb_image.data, w, h,
bytes_per_line, QImage.Format_RGB888)
                pixmap = QPixmap.fromImage(q_image)

self.image_label.setPixmap(pixmap.scaled(self.image_label.width()
- 10, self.image_label.height() - 10 , Qt.KeepAspectRatio,
Qt.SmoothTransformation))
            except Exception as e:
                print(f"Помилка оновлення зображення: {e}")
                self.show_error(f"Помилка відображення результату:
{e}")

    if __name__ == '__main__':
        app = QApplication(sys.argv)
        window = ImageProcessingApp()
        window.show()
        sys.exit(app.exec_())
```