Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмного забезпечення автоматичного розпізнавання образів у підсистемі безпеки розумний будинок»

Студента 2м курсу, 5 групи,
спеціальності 121«Інженерія програмного забезпечення»,
спеціалізації «Інженерія програмного забезпечення» Куткового Нікіти Георгійовича

_____
підпис студента

Науковий керівник
кандидат технічних наук,
доцент Савченко Тетяна Віталіївна

_____
підпис керівника

Гарант освітньої програми
доктор технічних наук,
професор Криворучко Олена Володимирівна

_____
підпис керівника

КИЇВ – 2019

ЗМІСТ

#

ВСТУП

Актуальність роботи. Величезна увага в даний час приділяється системам, що використовують машинний зір в якості основного джерела інформації. Для цього ведуться пошуки нових алгоритмів обробки і розпізнавання образів та зображень.

Однією з областей інформатики, досягнення якої можна використовувати вже зараз є комп'ютерний зір. Алгоритми для знаходження границь одиничних об'єктів, детектування груп різних об'єктів, будь то аналіз людських осіб або автоматичне визначення номерів автомобілів, є лише малою частиною досягнень даної науки.

Комп'ютерний зір як наукова дисципліна з'явився в кінці 1970-х років. Основна мета комп'ютерного зору – автоматизація процесу розпізнавання і обробки цифрових зображень і відеофайлів. Дана дисципліна знайшла застосування в безлічі областей, наприклад в розробці розумного дому.

Основним застосуванням комп'ютерного зору є розпізнавання образів, яке в науці прийнято називати сегментацією. Сегментація – це групування пікселів зображень за певними ознаками. Після сегментації зображення проводиться детектування об'єктів, присутніх в даному зображенні.

Метою роботи є розробка програмного забезпечення автоматичного розпізнавання образів у підсистемі безпеки розумного дому.

Об’єктом дослідження є автоматичне розпізнавання образів.

Предметом дослідження є способи, засоби та методи розробки програмного забезпечення розумного дому.

Задачами дослідження виступають:

- розгляд сутності концепції розумного дому;
- огляд поняття комп'ютерного зору як елементу системи розпізнавання образів у підсистемі безпеки розумного дому;
- розгляд задач, функцій та проблем комп'ютерного зору;
- опис мови програмування Python;
- опис відкритої бібліотеки OpenCV;
- аналіз поняття машинного навчання;
- аналіз сегментації зображень та зіставлення шаблонів;
- опис процесу виділення країв та детектора границь Канні;
- аналіз примітивів Хаара та їхнього математичного визначення;
- опис пошуку об'єкту за кольором;
- визначення архітектури програмного забезпечення;
- опис модуля препроцесінгу;
- опис модуля визначення ключових особливостей;
- опис корпусу даних;
- опис роботи програмного забезпечення.

Наукова новизна дослідження заключається в розробці автоматичного розпізнавання образів за допомогою мови Python, яка виступає одним з найпрогресивніших засобів програмного забезпечення

Методи дослідження. В роботі використовуються методи об'єктно орієнтованого програмування, розпізнавання, машинного навчання, декомпозиції задач, синтезу та аналізу.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ ОБРАЗІВ

1.1. Сутність концепції розумного дому

Поняття «розумний будинок» було сформульовано Інститутом інтелектуальної будівлі у Вашингтоні (округ Колумбія) в 1970-х роках: це будівля, що забезпечує продуктивне й ефективне використання робочого простору [3].

Але саме розвиток концепція розумних будинків починає з 1990-х років. Найбільш поширене визначення – будинок, який досить розумний для допомоги його мешканцям жити незалежно і комфортно за допомогою технології, яка визначається як «Розумний дім». В розумному домі всі механічні та електронні пристрої формують єдину мережу і можуть комунікувати між собою, з користувачами і з зовнішнім світом.

Все більша частина активності людини починає проводитися вдома. Це один з головних тез концепції розумних будинків. За даними асоціації всесвітнього огляду цінностей (WVSA) сучасне суспільство переходить на нові форми зайнятості, коли йде в минуле поняття робочого місця і поїздки на роботу [16]. При цьому зайнятість стає постійною і перекладається на місце проживання людини, таким чином значну частину життя людина починає проводити в своєму будинку. У числі інших наслідків таких змін – підвищення вимог до житлових будинків по частині життєзабезпечення і автоматизації.

Функції розумного будинку дуже великі. Основний функціонал традиційної системи може бути представлений в наступному вигляді:

- Система електроживлення, освітлення та опалення;
- Система аудіо- і відеотехніки;
- Комп'ютерні системи управління, зв'язку і призначеного для користувача інтерфейсу;
- Модулі управління систем вентиляції, кондиціонування, водопостачання та каналізації;
- Система обслуговування території;
- Система метеоконтролю;
- Система охоронно-пожежної сигналізації.

Всі ці функції реалізуються за допомогою спеціальних датчиків, підключених до головного контролеру.

Одним з основоположних компонентів концепції розумного будинку є Інтернет речей. Він включає в себе систему ідентифікації всіх задіяних об'єктів, систему вимірювання їх станів і перетворення їх в дані і систему передачі між ними інформації та енергії. В інтернет речей може входити вся «Розумна техніка». Починаючи з кухні, де кожен її компонент буде розумний, тобто забезпечений датчиками, що аналізують дії людей, що становлять статистику дій, і мінімізують їх виконання. Наприклад, смарт-холодильник. Дисплей покаже вам всю необхідну інформацію, яку, як правило, дивляться вранці при зборах: новини, погода, список продуктів, невеликий список справ і інше, сам замовить відсутні продукти. Духова шафа на Android з програмою приготування бажаної страви, можливість стеження в реальному часі за процесом запікання і автоматичним вимиканням. Розумна посудомийна машина, що розпізнає ступінь забрудненості посуду і необхідну для її промивки кількість води. Розумна кавоварка, мультиварка і чайник з віддаленим керуванням і багато іншого. У підсумку ми отримуємо економію електричних і водних ресурсів, і найголовніше економія часу, що значно спрощує рутинні побутові дії. Голосові помічники, роботи-пилососи, розумні

ковдри, браслет, що передає серцебиття близьких, розумні фоторамки, електронні годівниці для тварин, електронні замки, обладнання доповненої реальності, розумні лампи і багато іншого. Автоматизації піддається будь-яка річ у домі.

У науковій літературі існує досить великий обсяг публікацій, присвячений системам розумного будинку. На підставі існуючих аналітичних оглядів [2, 3, 11] можна систематизувати принципові положення про системи розумного будинку.

В першу чергу виділимо основні властивості систем, які пропонуються в даних публікаціях:

1) Автоматизація, як здатність до пристосування автоматичних пристроїв або до виконання автоматичних функцій;

2) Мульти-функціональність, як здатність виконувати різні обов'язки і функції;

3) Пристосовність, як здатність до навчання, до прогнозування та обслуговування потреб користувачів;

4) Інтерактивність як здатність до взаємодії між користувачами і до обміну інформацією між компонентами системи;

5) Ефективність як здатність виконувати функції так, щоб економити час і гроші.

Всі технічні продукти на ринку автори поділяють на 3 категорії:

1) Пристрої управління;

2) Графічні пристрої призначеного для користувача інтерфейсу;

3) Виконавчі пристрої.

При цьому пристрої управління поділяються на вторинні категорії:

а) Центральне управління, яке забезпечує комунікацію з безліччю додатків і пристроїв в будинку. Користувачі можуть керувати ним з різних вхідних точок;

6) Прикладне управління, яке забезпечує контроль на рівні окремих пристроїв і можливість для користувача безпосередньо взаємодіяти з ними;

в) Мережеве управління – автономна система прямого і зворотного зв'язку між пристроями, що забезпечує їх коректне взаємодія на основі базових примітивних функцій.

Графічний інтерфейс може включати в себе найпростіші дисплеї для моніторингу витрати ресурсів, онлайн / офлайн платформу для комплексного управління будинком, спеціальні додатки для смартфонів та інше.

Виконавчі пристрої підрозділяються на:

а) Сенсори і датчики, за допомогою яких збираються вихідні дані про те, що відбувається в будинку і на прилеглій просторі;

б) Комунікації – дротові або бездротові пристрої, що забезпечують зв'язок між компонентами системи і підтримують мережу будинку. Вони забезпечують також зв'язок із зовнішнім світом;

в) Робоче обладнання – електричні приводи, двигуни, джерела звуку і світла – все, чим керує система.

1.2. Поняття комп’ютерного зору як елементу системи розпізнавання образів у підсистемі безпеки розумного дому

Людина отримує великий обсяг інформації за допомогою зору. Для людини можливість бачити і розпізнавати об'єкти є природною і звичною можливістю, що для комп'ютера є надзвичайно складним завданням.

Комп'ютерний зір – це теорія і технологія створення машин, які здатні виявляти, відстежувати і класифікувати об'єкти. Мета комп'ютерного зору – це автоматизація процесу розпізнавання і обробки цифрових зображень і відеофайлів, тобто метою є процес навчання комп'ютера розпізнавати об'єкти на статичному зображенні або в відео.

Комп'ютерний зір як наукова дисципліна з'явився в кінці 1970-х років (в оригіналі Computer vision, далі CV) і відноситься до теорії і технології створення штучних систем, які отримують інформацію з зображень. Відеодані можуть бути представлені безліччю форм, такими як відеопослідовність,

зображення з різних камер або тривимірні дані, наприклад, з пристрою Kinect або медичного сканера [35].

На вхід комп'ютера подається деяке зображення, а на виході можливе отримання границь всіх важливих об'єктів і їх класифікацію. Вхідні дані можуть містити деяку контекстну інформацію, наприклад, «камера встановлена в машині». Рішенням може бути «чи є людина в цій сцені». Новим варіантом може бути процес перетворення кольорового зображення в чорно-біле або усунення ефекту руху камери з послідовності зображень.

Прикладами застосування систем комп’ютерного зору можуть бути:

1. Системи управління процесами (промислові роботи, автономні транспортні засоби).

2. Системи відеоспостереження.

3. Системи організації інформації (наприклад, для індексації баз даних зображень).

4. Системи моделювання об'єктів або навколишнього середовища (аналіз медичних зображень, топографічне моделювання).

5. Системи взаємодії (наприклад, пристрої введення для системи людино-машинної взаємодії).

6. Системи доповненої реальності.

7. Обчислювальна фотографія, наприклад, для мобільних пристроїв з камерами.

Людський мозок розділяє сигнал, що надійшов від зору на безліч каналів, які згодом передають різного виду інформацію в мозок людини. Мозок влаштований таким чином, що увага концентрується тільки на важливих ділянках зображення, виключаючи при цьому інші. Сигнали, що надходять на асоціативні входи, які прийшли від датчиків контролю м'язів і всіх інших почуттів, дозволяють мозку спиратися на перехресні асоціації, накопичені за роки прожитого життя. За рахунок зворотного зв'язку в мозку, процес повторюється знову і знову і включає в себе датчик (очі), який

механічно управляє освітленням за допомогою райдужної оболонки і налаштовує прийом на поверхні сітківки.

В системі комп'ютерного зору все по-іншому. Комп'ютер отримує тільки сітку з числами від камери або з диску. Зупинимося детальніше на галузях застосування комп’ютерного зору. Застосовується комп'ютерний зір у багатьох областях включаючи промислові, військові та побутові системи. Одним з найбільш важливих застосувань є обробка зображень в медицині. Ця область характеризується отриманням інформації з відеоданих для постановки медичного діагнозу пацієнтам. У більшості випадків, відеодані отримують за допомогою мікроскопії, рентгенографії, ангіографії, ультразвукових досліджень і томографії. Прикладом інформації, яка може бути отримана з таких відеоданих є виявлення пухлин, атеросклерозу або інших злоякісних змін. Також прикладом може бути вимір розмірів органів, кровотоку і т. д. Ця прикладна область також сприяє медичним дослідженням, надаючи нову інформацію, наприклад, про будову мозку або якість медичного лікування.

Іншою прикладної областю комп'ютерного зору є промисловість. Тут інформацію отримують для цілей підтримки виробничого процесу. Автоматизація дозволяє не тільки здешевити процес виробництва, але і зменшити кількість браку на виході. Прикладом може служити контроль якості, коли деталі або кінцевий продукт автоматично перевіряються на наявність дефектів. Іншим прикладом є вимір положення і орієнтації деталей, що піднімаються рукою робота.

Найбільшою областю застосування комп'ютерного зору є воєнно-промисловий комплекс. Прикладами можуть бути виявлення ворожих солдатів і транспортних засобів і управління ракетами. Найбільш досконалі системи управління ракетами, ґрунтуючись на отриманих відеоданих, можуть посилати ракету в задану область і замість конкретної цілі, виробляти селекцію цілей, коли ракета досягає заданої області. Використовується автоматична обробка даних, яка зменшує складність і збільшує надійність одержуваної інформації і сприяє прийняттю стратегічних рішень [35].

Новою областю застосування комп'ютерного зору є автономні транспортні засоби, включаючи підводні, наземні (роботи, машини), повітряні. Рівень автономності змінюється від повністю автономних (безпілотних) до транспортних засобів, де системи, засновані на комп'ютерному зорі, підтримують водія або пілота в різних ситуаціях. Повністю автономні (безпілотні) транспортні засоби використовують комп'ютерний зір для навігації, тобто для отримання інформації про місце свого перебування, для створення карти навколишнього оточення, для виявлення перешкод. Також вони використовуються для певних конкретних завдань, наприклад, для виявлення лісових пожеж. Прикладами таких систем можуть бути система попереджувальної сигналізації про перешкоди в автомобілях і системи автономної посадки літаків.

Також CV використовується при розробці концепцій розумних будинків.

В даний час CV набуло широкого застосування – це розпізнавання номерів автомобілів, осіб людей, тексту, дороги, автомобілів і багато іншого.

1.3. Задачі, функції та проблеми комп’ютерного зору

Розглянемо задачі комп’ютерного зору:

Розпізнавання – це класична задача в комп'ютерному зорі, обробці зображень і машинному зорі, яка визначає чи показують відеодані деякий характерний об'єкт, особливість чи активність. Існуючі методи вирішення цього завдання ефективні тільки для окремих об'єктів, таких як прості геометричні об'єкти (наприклад, багатогранники), людські обличчя, друковані або рукописні символи, автомобілі і тільки в певних умовах, зазвичай це певний рівень освітлення, фон і положення об'єкта відносно камери.

Рух – це завдання, пов'язані з оцінкою руху, в яких послідовність зображень або відео обробляється для знаходження оцінки швидкості кожної точки зображення або 3D сцени. Прикладами таких завдань є: визначення

тривимірного руху камери або стеження, тобто слідування за переміщеннями об'єкта (наприклад, машин або людей).

Відновлення зображень – це видалення шуму. Наприклад, шум датчика, розмитість рухомого об'єкту і т. д. Вирішуються ці завдання за допомогою різних типів фільтрів, таких як фільтри нижніх або середніх частот. Більш складні методи видалення шумів використовують початковий аналіз відеоданих на наявність різних структур, таких як лінії або межі, і тільки потім управління процесом фільтрації на основі цих даних.

Функціями комп’ютерного зору виступають:

Отримання зображень – цифрові зображення надходять від одного або декількох датчиків зображення, які включають різні типи світлочутливих камер, датчики відстані, радари, ультразвукові камери і т.д. Залежно від типу датчика отримані дані, можуть бути послідовністю зображень, 2D-зображенням або 3D-зображенням. Інтенсивності світла зазвичай відповідають значення пікселів в одній або декількох спектральних смугах (кольорові або зображення у відтінках сірого), які також залежать від різних фізичних вимірювань, як глибина, поглинання або відбиття звукових або електромагнітних хвиль, ядерний магнітний резонанс.

Попередня обробка: для отримання необхідної інформації потрібно обробити відеодані так, щоб вони задовольняли умовам використовуваного методу. Наприклад, повторна вибірка для переконання в тому, що координатна система зображення вірна, видалення шуму з тим, щоб видалити спотворення, що вносяться датчиком, поліпшення контрастності, для того щоб потрібна інформація могла бути виявлена і масштабування для кращого розрізнення структур на зображенні [23].

Виділення деталей: деталі зображення різного рівня складності виділяються з відеоданих. Наприклад, лінії, межі і кромки або локалізовані точки інтересу, такі як кути, краплі або точки і більш складні деталі, які відносяться до структури, форми або руху.

Детектування/Сегментація: на певному етапі обробки приймається рішення про те, які точки або ділянки зображення є важливими для подальшої обробки. Наприклад, виділення певного набору точок чи сегментація одного, або декількох ділянок зображення, які містять характерний об'єкт.

При обробці даних за допомогою комп’ютерного зору присутня низка проблем. Наведемо найголовніші з них:

Високорівнева обробка: вхідні дані тут представляють невеликий набір даних, що складаються з множин точок або ділянки зображення, в якому ймовірно перебуває певний об'єкт. Наприклад, перевірка даних, чи задовольняють вони умовам, що залежать від методу і застосування, оцінка характерних параметрів, таких як положення або розмір об'єкта, або класифікація виявленого об'єкту за різними категоріями.

Ракурси. При порівнянні двох фотографій особи однієї людини в анфас і в профіль ми побачимо, що вони мають більше відмінностей, ніж подібностей. Те ж саме відбувається і з безліччю інших об'єктів: з одного ракурсу ми можемо визначити об'єкт вухо, а з іншого – цього об’єкту вже немає. Але якщо знати, що об'єкти завжди будуть зображені в певній позиції, то задача визначення даних об'єктів значно полегшується. З іншого боку, при зміні ракурсу, складність визначення об'єкта зростає в багато разів.

Освітлення. Людина може отримати безліч інформації за допомогою освітлення і тіней від об'єктів і також з використанням контексту. Тіні і освітлення додають додаткові деталі. Наприклад, колір одягу об'єкта залежить від освітлення. Навчання машини розумінню тіней і зміни через них кольору об'єкта є одним із завдань CV.

Масштаб. Об'єкти можуть бути різних розмірів в залежності від їх розташування. Людське око може здогадатися, що на одному зображенні знаходиться автомобіль, а на другому – іграшкова машинка. Комп'ютер же визначить ці об'єкти однаково, якщо його не навчити.

Деформація. Живі об'єкти при русі можуть зазнавати деформації. Наприклад, після аварії автомобіль може сильно змінитися і розпізнати його стане складніше.

Перекриття. Наприклад, якщо дві машини їдуть по різних рядах і одна розпочне процес обгону , то перша машина перекриє другу щодо спостерігача, який припустимо знаходиться на узбіччі. В результаті частина, а іноді вся інформації про об'єкт пропадає.

Рух. Багато об'єктів в русі можуть виявитися розмитими або просто змінити свій ракурс.

Маскування. Для безпеки об'єкта доводиться використовувати маскування. Наприклад, знайти тканину однакового кольору зі стіною монотонного кольору не складе великих труднощів для зловмисника, але запросто обдурить примітивну автоматизовану систему відеоспостереження.

Внутрішньокласова мінливість. Всі люди виглядають по-різному: у кожного свій колір обличчя, форма голови, вух і т.д. У деяких людей може не бути певних частин тіла (рук, ніг). Велика кількість неживих об'єктів, таких як каміння, наприклад, можуть відрізнятися від об'єктів того ж типу. Це стосується і речей, вироблених самими людьми.

Контекст. З контексту людина отримує великий обсяг інформації. Наприклад, ведмідь знаходиться в музеї без решітки є опудалом і не є серйозним. Але якщо він в дикому лісі, то він живий і небезпечний для людей.

Висновки до розділу 1

Отже, розумний дім – житло сучасного типу, організоване для проживання людей за допомогою автоматизації і високотехнологічних пристроїв. Під «розумним» будинком слід розуміти систему, яка забезпечує безпеку і ресурсозбереження (в тому числі і комфорт) для всіх користувачів.

У найпростішому випадку вона повинна вміти розпізнавати конкретні ситуації, що відбуваються в будинку, і відповідним чином на них реагувати: одна з систем може управляти поведінкою інших по заздалегідь виробленим алгоритмам. Крім того, завдяки автоматизації декількох підсистем забезпечується синергетичний ефект для всього комплексу.

Це простіше зрозуміти, якщо уявити, наприклад, що система опалення ніколи не зможе працювати проти системи кондиціонування. А опалення здійснюється не тільки по погоді, але і з урахуванням цілого ряду інших факторів. Від сили вітру, за передбаченням, від часу доби (вночі комфортна температура менше).

Можна вважати, що це найбільш прогресивна концепція взаємодії людини (користувачів) з житловим простором, коли в автоматизованому режимі відповідно до зовнішніми і внутрішніми умовами задаються і відслідковуються режими роботи всіх інженерних систем і електроприладів.

При розробці програмного забезпечення автоматичного розпізнавання образів у підсистемі безпеки розумного дому використовується комп’ютерний зір – це теорія і технологія створення машин, які здатні виявляти, відстежувати і класифікувати об'єкти. Головною його метою є автоматизація процесу розпізнавання і обробки цифрових зображень і відеофайлів, тобто процес навчання комп'ютера розпізнавати об'єкти на статичному зображенні або в відео. Основними задачами комп’ютерного зору є розпізнавання, рух, відновлення зображень. Основними функціями виступають: отримання зображень, попередня обробка, виділення деталей, детектування/сегментація. Основними проблемами комп’ютерного зору є високорівнева обробка, ракурси, освітлення, масштаб, деформація, перекриття, рух, маскування, внутрішньокласова мінливість, контекст.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ВИБОРУ СПОСОБІВ РІШЕННЯ ЗАДАЧІ ТА ЗАСОБІВ РОЗРОБКИ

2.1. Мова програмування Python

Python – одна з тих рідкісних мов програмування, які одночасно претендують на звання простих і потужних. Офіційно Python представляють так:

Python – це проста в освоєнні і потужна мова програмування. Вона надає ефективні високорівневі структури даних, а також простий, але ефективний підхід до об'єктно-орієнтованого програмування. Її елегантний синтаксис і динамічна типізація разом з тим, що вона є інтерпретуємою, роблять її ідеальним мовою для написання сценаріїв та швидкої розробки додатків в різних областях і на більшості платформ.

Python – проста і мінімалістична мова. Читання хорошої програми на Python дуже нагадує читання англійського тексту, хоча і досить суворога. Така псевдокодоваа природа Python є однією з її найсильніших сторін. Вона дозволяє зосередитися на вирішенні завдання, а не на самій мові.

Вільний та відкритий

Python – це приклад вільного і відкритого програмного забезпечення – FLOSS (Free / Libré and Open Source Software). Простіше кажучи, ви маєте право вільно поширювати копії цього програмного забезпечення, читати його вихідні тексти, вносити зміни, а також використовувати його частини в своїх програмах. В основі вільного ПЗ лежить ідея спільноти, яка ділиться своїми

знаннями. Це одна з причин, за якими Python краще ніж інші мови: вона була створена і постійно поліпшується співтовариством, яке просто хоче зробити його краще.

Високорівнева мова

Також Python є мовою високого рівня. Це означає, що при написанні програми не потрібно буде відволікатися на такі низькорівневі деталі як управління пам'яттю, котра використовується програмою та ін.

Багатоплатформний

Завдяки своїй відкритій природі, Python був портований на багато платформ (тобто змінений таким чином, щоб працювати на них). Всі програми зможуть запускатися на будь-який з цих платформ без будь-яких змін, якщо тільки було уникнено використання системно-залежних функцій.

Python можна використовувати в GNU / Linux, Windows, FreeBSD, Macintosh, Solaris, OS / 2, Amiga, AROS, AS / 400, BeOS, OS / 390, z / OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE і навіть на PocketPC.

Також можна використовувати таку платформу, як Kivy для створення ігор для iOS (IPhone, iPad) і Android.

Інтерпретований

Python можливо інтерпретувати. Це означає, що програма, котра написана на компільованій мові програмування, як наприклад, C або C ++, перетворюється з вихідної мови (тобто C або C ++) в мову, зрозумілу комп'ютеру (бінарний код, тобто нулі і одиниці) за допомогою компілятора із застосуванням різноманітних прапорів і параметрів. Коли ви запускаєте таку програму, компонувальник / завантажувач копіює програму з диска в оперативну пам'ять і запускає її.

Python ж, навпаки, не вимагає компіляції в бінарний код. Програма просто виконується з вихідного тексту. Python сам перетворює цей вихідний текст в деяку проміжну форму, котру називають байткодом, а потім переводить його на машинну мову і запускає. Все це помітно полегшує

використання Python, оскільки немає необхідності піклуватися про компіляцію програми, підключенні і завантаженні потрібних бібліотек і т.д. Разом з тим, це робить програми на Python набагато більш мобільними, так як досить їх просто скопіювати на інший комп'ютер, і вони працюють.

Об'єктно-орієнтований

Python підтримує як процедурно-орієнтоване, так і об'єктно-орієнтоване програмування. У процедурно-орієнтованих мовах програми будуються на основі процедур або функцій, які представляють собою просто-напросто багаторазово використовувані фрагменти програми. В об'єктно-орієнтованих мовах програмування програми будуються на основі об'єктів, котрі об'єднують в собі дані і функціонал. Python надає прості, але потужні засоби для ООП, особливо в порівнянні з такими великими мовами програмування, як C ++ або Java.

Розширюваний

Якщо необхідно, щоб деяка критична частина програми працювала дуже швидко або розробник змушений приховати частину алгоритму, можна написати цю частину програми на C або C ++, а потім викликати її з програми на Python.

Вбудований

Python можна вбудовувати в програми на C / C ++, щоб надавати можливості написання сценаріїв їх користувачам.

Великі бібліотеки

Стандартна бібліотека Python просто величезна. Вона може допомогти у вирішенні найбільш різноманітних завдань, пов'язаних з використанням регулярних виразів, генеруванням документації, перевіркою блоків коду, розпаралелюванням процесів, базами даних, веб-браузерами, CGI, FTP, електронною поштою, XML, XML-RPC, HTML, WAV файлами, криптографією, GUI (графічний інтерфейс користувача) та іншими системно-залежними речами. Пам'ятайте, що все це є на кожному кроці, де встановлений Python. У цьому полягає філософія Python «Все включено».

Крім стандартної бібліотеки, існує безліч інших високоякісних бібліотек, які можна знайти в каталозі пакетів Python [29].

Отже, Python – дуже захоплююча і потужна мова. Вона має гарне співвідношення продуктивності і можливостей, що робить написання програм одночасно цікавим і легким.

Також в процесі розробки використовується ImageAI – бібліотека python, яка дозволяє програмістам і розробникам програмного забезпечення легко інтегрувати новітні технології комп'ютерного зору в свої вже існуючі або нові додатки, використовуючи лише кілька рядків коду.

Щоб виконати виявлення об'єкта за допомогою ImageAI, все, що потрібно зробити користувачу, це:

- Встановити Python на комп'ютер;
- Встановити ImageAI і всі його залежності;
- Завантажити модельний файл Object Detection;
- Запустити зразок коду (який складається з десяти рядків).

ImageAI надає багато корисних можливостей, які дозволяють кастомізувати функціонал і виконувати завдання з розпізнавання об'єктів для різного виробничого застосування програмного рішення. Нижче перераховані деякі з підтримуваних можливостей:

-Зміна мінімального відсотка точності: за замовчуванням, об'єкти, точність визначення яких нижче 50%, не показуються і не включаються до звіту. Користувач може встановити більш високий поріг для випадків з більшою ймовірністю виявлення або нижчий для тих випадків, коли необхідно визначити всі потенційні об'єкти.

- Персоналізоване розпізнавання об'єктів: за допомогою наданого модулем класу CustomObject, користувач може задати параметри, відповідно до яких клас виявлення буде включати до звіту результати по одному або декільком унікальних об'єктів.

- Швидкість виявлення: вибираючи швидкість виявлення між «швидко», «швидше», «ще швидше», користувач може зменшувати час, необхідний для виявлення об'єктів на зображенні.

- Тип вихідного файлу: в якості вихідного файлу користувач може вказати шлях до зображення, масив Numpy або файловий потік зображення.

- Тип підсумкового файлу: для функції detectObjectsFromImage користувач може задати параметри, згідно з якими функція буде зберігати зображення у вигляді файлу або масиву Numpy.

Також в ході розробки використовується tensorflow/keras. Tensorflow (далі – TF) – досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain. Довгий час фреймворк розроблявся в закритому режимі під назвою DistBelief, але після глобального рефакторінга 9 листопада 2015 року його було випущено в open source. За рік з невеликим TF доріс до версії 1.0, знайшов інтеграцію з keras, став значно швидше і отримав підтримку мобільних платформ. Останнім часом фреймворк розвивається ще й в сторону класичних методів, і в деяких частинах інтерфейсу вже чимось нагадує scikit-learn. До поточної версії інтерфейс змінювався активно і часто, але розробники пообіцяли заморозити зміни в API. Ми будемо розглядати тільки Python API, хоча це не єдиний варіант – також існують інтерфейси для C ++ і мобільних платформ.

Робота с TF будується навколо побудови і виконання графа обчислень. Граф обчислень – це конструкція, яка описує те, яким чином будуть проводитися обчислення. У класичному імперативному програмуванні ми пишемо код, який виконується за рядком. В TF звичний імперативний підхід до програмування необхідний тільки для якихось допоміжних цілей. Основа TF – це створення структури, яка задає порядок обчислень. Програми природним чином структуруються на дві частини – складання графа обчислень і виконання обчислень в створених структурах [20].

В TF граф складається з плейсхолдеров, змінних і операцій. З цих елементів можна зібрати граф, в якому будуть обчислюватися тензори.

Тензори – багатовимірні масиви, вони служать «паливом» для графа. Тензором може бути як окреме число, вектор ознак з розв'язуваної задачі або зображення, так і цілий батч описів об'єктів або масив із зображень. Замість одного об'єкта ми можемо передати в граф масив об'єктів і для нього буде вирахувано масив відповідей. Робота TF з тензорами схожа на те, як обробляє масиви numpy, у функціях якого можна вказати вісь масиву, щодо якої буде виконуватися обчислення.

Обчислювальні графи виконуються в сесіях. Об'єкт сесії (tf.Session) приховує в собі контекст виконання графа – необхідні ресурси, допоміжні класи, адресні простори.

Існує два типи сесій – звичайні, які реалізовані в tf.Session і інтерактивні (tf.InteractiveSession). Різниця між ними в тому, що інтерактивна сесія більше підходить для виконання в консолі і відразу визначає себе як сесія за замовчуванням. Основний ефект – об'єкт сесії не потрібно передавати у функції обчислення як параметр.

2.2. Відкрита бібліотека OpenCV

Бібліотека Open CV (Open Source Computer Vision Library) – це бібліотека комп'ютерного зору з відкритим вихідним кодом, тобто з набором алгоритмів, реалізованих у вигляді функцій CV, написана на C і C ++.

Головна перевага даної бібліотеки в тому, що вона повністю відкрита та безкоштовна як в навчальних, так і в комерційних цілях. Альфа версія вийшла в січні 1999 року, а перший реліз – в 2006 році. В даний момент остання стабільна версія 2.4.9, також випущена альфа- версія 3.0.

Існують дистрибутиви під Linux, Windows, Mac, iOS і Android. Бібліотека має онлайн-документацію, яка підтримує такі мови програмування: C / C ++, Python, Java / Scala MATLAB. Може вільно використовуватися в академічних і комерційних цілях – поширюється в умовах ліцензії BSD.

Open CV написана на мові високого рівня (C / C ++) і містить алгоритми для: інтерпретації зображень, калібрування камери за зразком, усунення оптичних спотворень, визначення подібності, аналіз переміщення об'єкта, визначення форми об'єкту та стеження за об'єктом, EE-реконструкція, сегментація об'єкта, розпізнавання жестів і т.д.

Спочатку бібліотека складалася з 5 основних модулів: Excore, Highqui, Cvaux, CvCam, але починаючи з версії 2.2, структура бібліотеки змінилася і на даний момент виглядає наступним чином:

- opencv_core – ядро: базові структури, обчислення: математичні функції, генерація псевдовипадкових чисел, DFT, DCT, введення і виведення в XML і т.п.
- opencv_imgproc – обробка зображень: фільтри, перетворення т.д.
- opencv_highgui – простий UI, завантаження і збереження відео і зображень.
- opencv_ml- методи і моделі машинного навчання: SVM, дерева прийняття рішень і т.д.
- opencv_features2d – різні дескриптори: SURF.
- opencv_video – аналіз руху і відстеження об'єктів: оптичний потік, шаблони руху, усунення фону.
- opencv_objdetect – детектування об'єктів на зображенні: вейвлети Хаара, HOG і т. д.
- opencv_calib3d – калібрування камери, пошук стерео-відповідності та елементи обробки тривимірних даних.
- opencv_flann – бібліотека швидкого пошуку найближчих сусідів: FLANN.
- opencv_contrib – супутній код, ще неготовий для застосування.
- opencv_legacy – застарілий код, збережений заради зворотної сумісності.
- opencv_gpu – прискорення деяких функцій Open CV за рахунок CUDA (Nvidia).

У OpenCV є кілька функцій для роботи з відео, найбільш використовуваними з яких є функція зчитування і запису відео [17].

CvCapture – структура, яка містить необхідну інформацію для зчитування кадрів з камери або з відеопотоку. Функції, які використовуються в залежності від джерела представлені на рис. 2.1

```
CvCapture*
cvCreateFileCapture(
const char* filename
);


CvCapture*
cvCreateCameraCapture( int
index
);
```

Рис. 2.1 Прототипи зчитування відеопотоку

cvCreateFileCapture () приймає всього один аргумент – ім'я AVI або MPG-файлу. Після виклику цієї функції Open CV відкриває і готує файл для зчитування.

Якщо відкриття файлу сталося успішно, то функція поверне покажчик на структуру CvCapture. Якщо з якоїсь причини, файл не існує або не знайдений відповідний кодек, функція поверне NULL. Необхідно знати який кодек застосовувати і чи встановлений він на комп'ютері.

Наприклад, необхідно прочитати файл, закодований за допомогою DIVX або стиснутий за допомогою MPG4 на комп'ютері з ОС Windows. Для коректної обробки повинна бути встановлена відповідна бібліотека, яка надасть всі необхідні ресурси для декодування відео. Необхідно перевіряти значення на NULL, так як всі комп'ютери мають різний набір ПЗ (наприклад, може бути відсутнім бібліотека для декодування відео).

Функція cvCreateCameraCapture () приймає ідентифікатор, який вказує до якої камери необхідно отримати доступ. Якщо камера одна, то ідентифікатор можна встановити в 0.

index- це сума з порядкового номера та «домену». «Домен» вказує на тип камери.

Open CV надає швидкий і простий у використанні інтерфейс для виконання морфологічних перетворень над зображенням.

Головні морфологічні перетворення – розширення і розмиття (звуження) Вони застосовуються:

- Для видалення шуму
- Виділення окремих елементів
- З'єднання різнорідних елементів на зображенні.
- Для пошуку нерівностей інтенсивності або отворів
- Градієнта зображення [17].

Розширення – згортка зображень (або області зображення), що має назву A, з деяким ядром, що має назву B. Ядро може бути будь-якої форми і розміру, але зазвичай це невеликий суцільний квадрат або диск з якорем в центрі і має одну певну точку прив'язки (якір). Ядро можна розглядати в якості шаблону або маски і його ефект на розширення залежно від оператора локального максимуму.

При скануванні зображення ядром B відбувається обчислення локального максимуму пікселя, що перекривається B, і потім значення пікселя, що лежить під опорною точкою, замінюється цим максимальним значенням. Це призводить до появи яскравих областей на зображенні; схематично це зростання показаний на рис. 2.2. Це зростання називається «розширенням оператора».

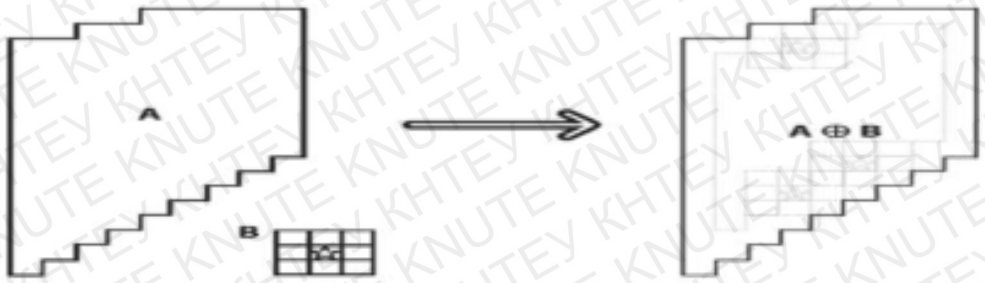

Рис. 2.2 Морфологічне перетворення – розширення

Розмиття – зворотна операція. Дія оператора розмиття полягає в обчисленні локального мінімуму під ядром. Даний оператор створює нове зображення на основі вихідного за наступним алгоритмом:

При скануванні зображення ядром B відбувається обчислення локального мінімуму пікселя, що перекривається B, і потім значення пікселя, що лежить під опорною точкою, замінюється цим мінімальним значенням. Схематично звуження показано на рис. 2.3.

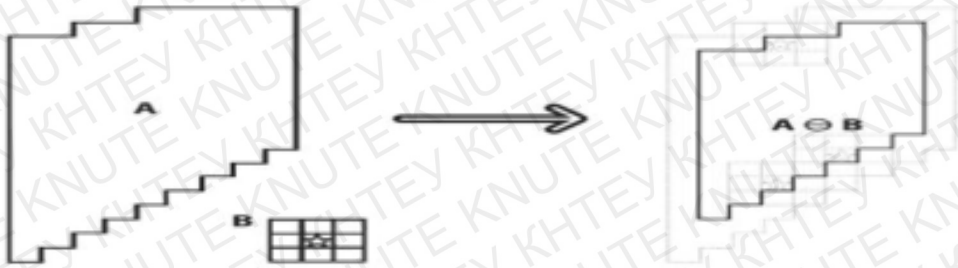

Рис. 2.3 Морфологічне перетворення – розмиття

Морфологічні перетворення виконуються над бінарними зображеннями, які отримуються за допомогою порогового перетворення. Розширення збільшує область A, а розмиття зменшує область A.

Розширення застосовується для того, щоб згладити вкраплення, а розмиття, щоб згладити виступи. Кінцевий результат завжди буде залежати від ядра.

В Open CV для виконання цих перетворень використовуються функції cvErode () і cvDilate () (рис. 2.4).

```
void
cvErode(
IplImage*
src ,
IplImage*
dst ,
IplConvKernel* B =
NULL, int
iterations = 1
);

void
cvDilate(
IplImage*
src ,
IplImage*
dst ,
IplConvKernel* B =
NULL, int
iterations = 1
);
```

Рис. 2.4 Прототипи функцій cvErode () і cvDilate ()

Обидві функції допускають використання одного і того ж зображення в якості вихідного і кінцевого.

- Третій аргумент – ядро, яке за замовчуванням дорівнює NULL. Якщо ядро дорівнює NULL, то використовується ядро 3x3 з якорем в центрі.

- Четвертий аргумент – кількість ітерацій. Якщо вказане значення не дорівнює 1, то операція буде застосовуватися кілька разів протягом одного виклику функції.

Операція розмиття використовується для усунення «вкраплень» шуму на зображенні. Суть операції звуження в тому, що вкраплення і шуми розмиваються, в той час як великі і відповідно більш значущі регіони не будуть зачіпатися.

Операція розширення використовується при спробі знайти пов'язані компоненти, тобто великі окремі області аналогічного кольору або інтенсивності. Корисність розширення виникає багато в чому через те, що, велика область розбита на кілька більш дрібні частини шумами, тінями або

якимись аналогічними ефектами. Застосування невеликого розтягування призводить до того, що ці області «плавляться» в одну.

2.3. Машинне навчання

Машинне навчання (Machine Learning) – великий підрозділ штучного інтелекту, який вивчає методи побудови алгоритмів, здатних навчатися. Машинне навчання знаходиться на стику математичної статистики, методів оптимізації та класичних математичних дисциплін, але має і власну специфіку, пов'язану з проблемами обчислювальної ефективності та перенавчання. Багато методів індуктивного навчання розроблялися як альтернатива класичним статистичним підходам. Багато методів тісно пов'язані з вилученням інформації та інтелектуальним аналізом даних [28].

Найбільш теоретичні розділи машинного навчання об'єднані в окремий напрям, теорію обчислювального навчання.

Машинне навчання – не тільки математична, а й практична, інженерна дисципліна. Чиста теорія, як правило, не призводить відразу до методів і алгоритмів, які можуть застосовуватися на практиці. Щоб змусити їх добре працювати, доводиться винаходити додаткові механізми, що компенсують невідповідність зроблених в теорії припущень до умов реальних завдань. Практично жодне дослідження в машинному навчанні не обходиться без експерименту на модельних або реальних даних, що підтверджує практичну працездатність методу.

Розглянемо докладніше види машинного навчання.

Навчання із учителем. У навчанні з учителем (Supervised learning) ідея полягає в тому, що існує деяка залежність між відповідями і об'єктами, але вона невідома. Відома тільки кінцева сукупність прецедентів – пар «об'єкт, відповідь», яка названа навчальної вибіркою. На основі цих даних потрібно відновити залежність, тобто побудувати алгоритм, здатний для будь-якого об'єкта видати досить точну відповідь. Для вимірювання точності відповідей

певним чином вводиться функціонал якості. Під учителем розуміється або сама навчальна вибірка, або той, хто вказав на заданих об'єктах правильні відповіді (рис. 2.5).

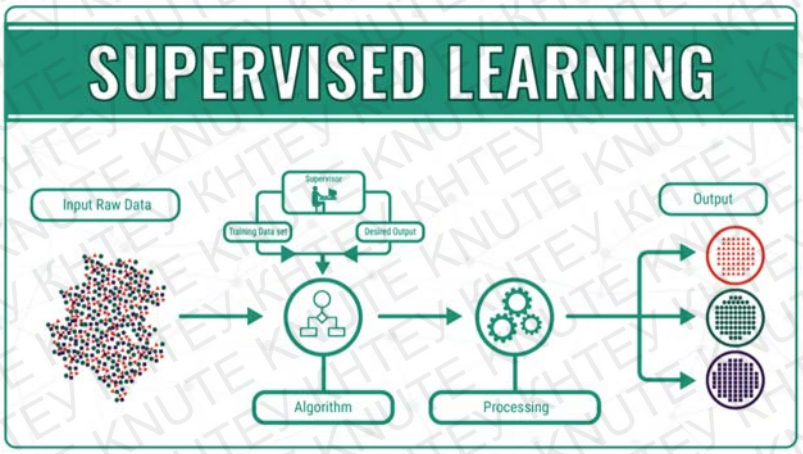

Рис. 2.5 Навчання із учителем

Функціонал якості зазвичай визначається як середня помилка відповідей, виданих алгоритмом, по всіх об'єктах вибірки.

Навчання без вчителя. Навчання без вчителя (Unsupervised learning) вирішує задачі, у яких відомий тільки опис множини об’єктів. І потрібно знайти внутрішні взаємозв’язки, залежності які існують між об’єктами (рис. 2.6). Прикладами таких задач є кластеризація і візуалізація даних.

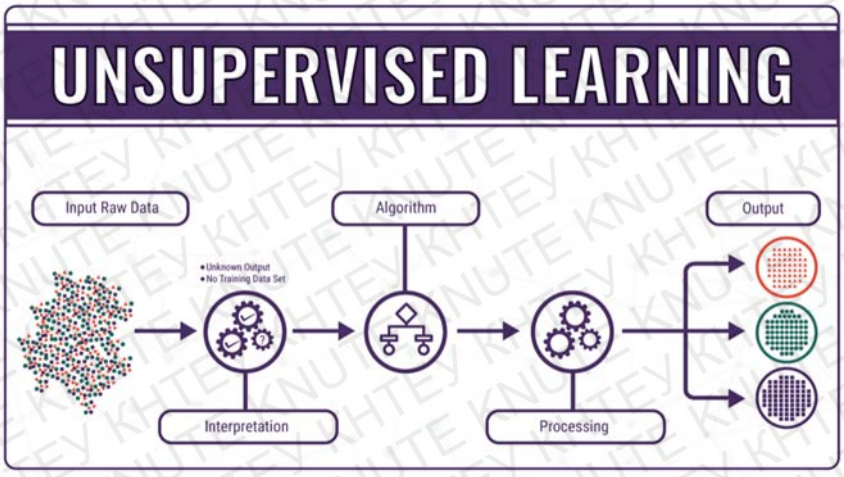

Рис. 2.6 Навчання без вчителя

Часткове навчання. Часткове навчання (semi-supervised learning) використовує при навчанні як розмічені, так і розділеного дані. Зазвичай використовується невелика кількість розмічених і значний обсяг нерозмічену даних. Часткове навчання є компромісом між навчанням без учителя (без будь-яких розмічених навчальних даних) і навчанням з учителем (з повністю розмічені набором навчання). Було відмічено, що якщо використовувати непомічені дані з невеликою кількістю розмічених то це може забезпечити значний приріст якості навчання. Під якістю навчання мається на увазі якийсь функціонал якості, наприклад, середньоквадратична помилка (рис. 2.7).

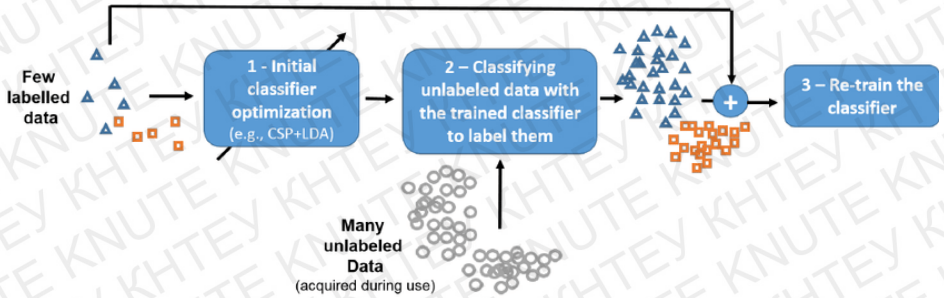

Рис. 2.7 Часткове навчання

Збір розмічених даних для завдання навчання часто вимагає, щоб кваліфікований експерт вручну класифікував об'єкти навчання. Витрати, пов'язані з процесом розмітки, можуть бути настільки великими, що створення повного набору може бути неможливим, в той час як збір нерозмічених даних порівняно недорогий. У подібних ситуаціях цінність часткового навчання складно переоцінити.

Приклад прикладної задачі – автоматична рубрикація великої кількості питань за умови, що деякі з них вже віднесені до якихось рубрик.

Навчання із підкріпленням. Навчання із підкріпленням (reinforcement learning) вивчає, як агент повинен діяти в середовищі, щоб максимізувати деякий довготривалий виграш. Алгоритми з частковим навчанням намагаються знайти стратегію, зробити так, щоби в кожному стані навколишнього середовища агент вибирав найкращу дію. В економіці і теорії

ігор навчання з підкріпленням розглядається в якості інтерпретації того, як може встановитися рівновага (рис. 2.8).

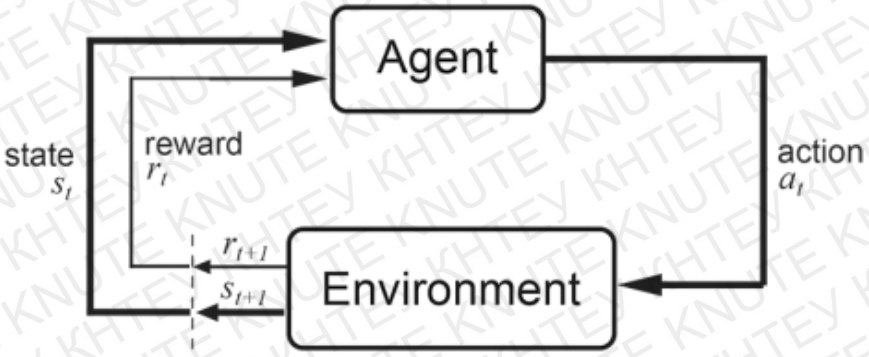

Рис. 2.8 Навчання з підкріпленням

При навчанні з підкріпленням, на відміну від навчання з учителем, не надаються вірні пари «вхідні дані-відповідь», а прийняття майже оптимальних рішень (що дають локальний екстремум) не обмежується явно. Навчання з підкріпленням намагається знайти компроміс між дослідженням невивчених областей і застосуванням наявних знань [32].

Формально найпростіша модель навчання з підкріпленням складається з:

1. безлічі станів оточення S;
2. безлічі дій A;
3. безлічі «виграшів».

Загалом навчання із підкріпленням працює таким чином:

1. На агента приходить стан;
2. Агент вибирає дію;
3. Середовище посилає агенту нагороду і наступний стан;
4. Агент опрацьовує нагороду і коректує свої дії.

Висновки до розділу 2

Отже, в ході дослідження була обрана мова програмування Python – це проста в освоєнні і потужна мова програмування. Вона надає ефективні високорівневі структури даних, а також простий, але ефективний підхід до об'єктно-орієнтованого програмування. Її елегантний синтаксис і динамічна типізація разом з тим, що вона є інтерпретуємою, роблять її ідеальним мовою для написання сценаріїв та швидкої розробки додатків в різних областях і на більшості платформ. Також в процесі розробки використовується ImageAI – бібліотека python, яка дозволяє програмістам і розробникам програмного забезпечення легко інтегрувати новітні технології комп'ютерного зору в свої вже існуючі або нові додатки, використовуючи лише кілька рядків коду. Також в ході розробки використовується tensorflow/keras. Tensorflow (далі – TF) – досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain. Робота с TF ґрунтується навколо побудови і виконання графа обчислень. Граф обчислень – це конструкція, яка описує те, яким чином будуть проводитися обчислення. В TF граф складається з плейсхолдеров, змінних і операцій. З цих елементів можна зібрати граф, в якому будуть обчислюватися тензори. Тензори – багатовимірні масиви, вони служать «паливом» для графа.

Розпізнавання образів у підсистемі безпеки розумного дому буде проводитися за допомогою відкритої бібліотеки Open CV (Open Source Computer Vision Library) це бібліотека комп'ютерного зору з відкритим вихідним кодом, тобто з набором алгоритмів, реалізованих у вигляді функцій CV, написана на C і C ++. Open CV написана на мові високого рівня (C / C ++) і містить алгоритми для: інтерпретації зображень, калібрування камери за зразком, усунення оптичних спотворень, визначення подібності, аналіз переміщення об'єкта, визначення форми об'єкту та стеження за об'єктом, EE-реконструкція, сегментація об'єкта, розпізнавання жестів і т.д.

В ході розробки програмного забезпечення використовується машинне навчання – великий підрозділ штучного інтелекту, який вивчає методи побудови алгоритмів, здатних навчатися. Машинне навчання знаходиться на стику математичної статистики, методів оптимізації та класичних математичних дисциплін, але має і власну специфіку, пов'язану з проблемами обчислювальної ефективності та перенавчання. Багато методів індуктивного навчання розроблялися як альтернатива класичним статистичним підходам. Багато методів тісно пов'язані з вилученням інформації та інтелектуальним аналізом даних. Машинне навчання поділяється на навчання з вчителем, навчання без вчителя, часткове навчання та навчання з підкріпленням.

РОЗДІЛ 3
ЗАГАЛЬНА ХАРАКТЕРИСТИКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ ОБРАЗІВ

3.1. Сегментація зображень та зіставлення шаблонів

Сегментація – це процес поділу цифрового зображення на декілька множин пікселів, або присвоєння кожному пікселю таких міток, щоб пікселі з однаковими мітками мали спільні візуальні характеристики.

Мета сегментації полягає в спрощенні уявлення зображення, щоб його було легше аналізувати. В результаті сегментації зазвичай виділяють границі та об'єкти на зображеннях. Пікселі, що належать до одних і тих же сегментів, схожі за деякою вирахуваною характеристикою. Наприклад, схожі за кольором і яскравістю, а сусідні елементи істотно відрізняються по ній (рис. 3.1).

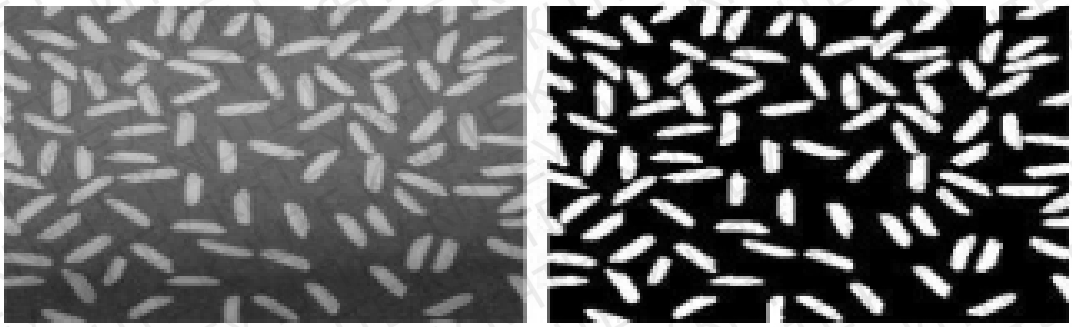

Рис. 3.1 Приклад сегментації зображення

Переваги методу:

Простота використання в тих ситуаціях, коли досліджувані об'єкти значно відрізняються від решти фону по якому-небудь параметру, наприклад, в біоінформатиці або пошук дефектів матеріалів.

Недоліки:

- Низька універсальність методів.
- Необхідність тонкого налаштування параметрів роботи алгоритму розпізнавання в кожному окремому випадку.
- Висока чутливість до змін освітлення сцени [30].

Зіставлення шаблонів (template matching). Ідея пошуку об'єктів на зображенні за шаблоном полягає в порівнянні шаблону, на якому зображений шуканий об'єкт, з підобластями оброблюваного зображення. Метод добре працює з чорно-білими, контрастними зображеннями. При його використанні необхідно враховувати такі моменти як освітленість, можливість об'єкта змінювати своє положення щодо камери та інші особливості, які можуть змінити отримані зображення.

При використанні даного методу необхідно:

1. Зафіксувати об'єкт.

2. Для визначення особи людини на зображенні, особа повинна бути завжди в анфас.

3. Зображення повинно бути якісним, тобто відсутність шумів і т.п.

4. Необхідно отримати шаблон.

Шаблон – це зображення об'єкта, яке потрібно знайти на іншому зображенні де проводиться порівняння інтенсивності. Шаблон попіксельно порівнюється із зображенням, на якому необхідно знайти цей об'єкт і вважається, що об'єкт знайдений якщо він відповідає заданому порогу [25].

Існує кілька метрик, що дозволяють визначити, наскільки якісно вийшло знайти зображення:

Сума абсолютних різниць – SAD (Sum of absolute differences):

$$SAD(x,y) = \sum_x \sum_y |I_1(x,y) - I_2(x,y)| \quad (3.1)$$

I_1 – інтенсивність пікселя в шаблоні;

I_2 – інтенсивність пікселя в зображенні;

(x, y) – координати пікселя.

При найменшому SAD досягається найкращий результат.

Сума квадратів різниць – SSD (Sum of squared differences)

$$SSD(x,y) = \sum_x \sum_y \left(I_1(x,y) - I_2(x,y)\right)^2 \quad (3.2)$$

При SSD бажано отримати результат близький до нуля.

Перехресна кореляція CC (Cross-correlation):

$$CC(x,y) = \left(I_1(x,y) * I_2(x,y)\right) \quad (3.3)$$

При CC найкращий результат повинен дорівнювати одиниці.

Переваги:

- Хороша застосовність при аналізі сцен, в яких камера статична, і всі екземпляри шуканих об'єктів виглядають однаково. Наприклад, при розпізнаванні деталей на складальній лінії.

Недоліки:

- Нестійка робота в разі масштабування, зрушення, повороту зображень, а також у випадках, коли об'єкт, котрий розпізнається, видно не повністю.

3.2. Виділення країв та детектор границь Канні

Виділення країв, границь або контурів є наступним методом знаходження об'єктів на зображенні (Edge detection). Завдяки краям об'єкта, можна спробувати зрозуміти, що за об'єкт представлений на зображенні.

Краї можна знаходити за різкими змінами кольору поверхні, глибини і нормалі поверхні. Коли краї будуть знайдені, далі для знаходження окремих об'єктів на зображенні вже використовується метод зіставлення шаблонів.

Рис. 3.2 Процес пошуку країв

З рис. 3.2 видно, що край – це місце різкої зміни значень функції інтенсивності зображень. Для знаходження краю, беремо функцію інтенсивності зображення і знаходимо точки екстремуму першої похідної, які будуть відповідати границям зображення. Знайти границю зображення можна також за допомогою градієнта (рис. 3.3).

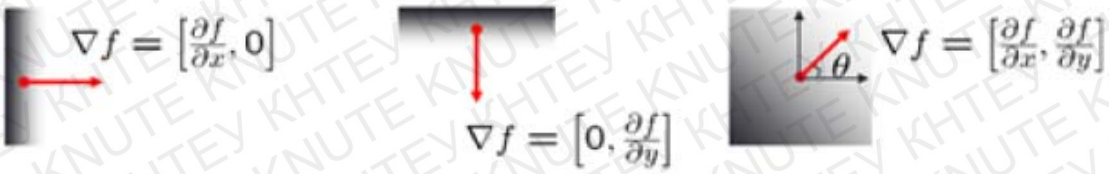

Рис. 3.3 Процес роботи градієнта

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] \tag{3.4}$$

З рис. 3.3 видно, що градієнт направлений в бік найбільшої зміни інтенсивності. Напрямок градієнта задається за допомогою формули:

$$\theta = \tan^{-1}\left(\frac{\left(\frac{\partial f}{\partial x}\right)}{\left(\frac{\partial f}{\partial y}\right)}\right) \tag{3.5}$$

Сила краю задається величиною або нормою градієнта:

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{3.6}$$

Цей метод при великій кількості зашумлення буде працювати погано і для поліпшення і застосування використовують згладжування і фільтрацію. Найвідоміші фільтри зображені на рис. 3.4:

Робертса $\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$

Превитт

$$\begin{bmatrix}-1 & -1 & -1\\0 & 0 & 0\\1 & 1 & 1\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\-1 & 0 & 1\\-1 & 0 & 1\end{bmatrix}$$

Собеля

$$\begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 2 & 1\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}$$

Рис 3.4 Найвідоміші фільтри

Недоліки методу обчислення градієнта:

- Товщина – отримані краї виходять товсті, розмиті.
- Незрозумілість – досить важко визначити які лінії співвідносяться один з одним і де саме знаходяться лінії одного об'єкта.

Краї (межі) – це такі криві на зображенні, уздовж яких відбувається різка зміна яскравості або інших видів неоднорідностей. Краї відображають важливі особливості зображення. Причинами виникнення країв є зміна освітленості, зміна кольору і зміна глибини сцени (орієнтації поверхні). Виділення істотних характеристик зображення і скорочення обсягу інформації для подальшого аналізу є головними цілями перетворення зображення в набір кривих

У 1986 році Джоном Канні був розроблений детектор границь, який став найпопулярнішим методом їх виділення. Канні вивчив математичну проблему отримання фільтра, оптимального за критеріями виділення, локалізації та мінімізації декількох відгуків одного краю. Детектор Канні повинен реагувати на границі і точно визначати їхню лінію, ігноруючи помилкові. На кожну границю необхідно реагувати тільки один раз, що дозволить уникнути сприйняття широких смуг зміни яскравості як сукупності границь.

Пікселями границь оголошуються точки, в яких досягається локальний максимум градієнта в напрямку вектора градієнта.

Алгоритм роботи цього методу наступний:

1. Згладжування. Проводиться розмиття зображення, щоб позбутися від шумів.

2. Пошук градієнтів. Розраховуються градієнти зображення: сила і напрямок. У місцях максимального значення градієнта – границя.

3. Виділення локальних максимумів. Смуги в кілька пікселів зменшують до смуг в один піксель.

4. Подвійна порогова фільтрація. Потенційні межі визначаються двома порогами (верхній і нижній). Верхній поріг необхідний для ініціалізації кривих, нижній – для продовження [15].

3.3. Примітиви Хаара та їхнє математичне визначення

Примітиви Хаара це прямокутники, що складаються з суміжних областей (рис. 3.4 і 3.5).

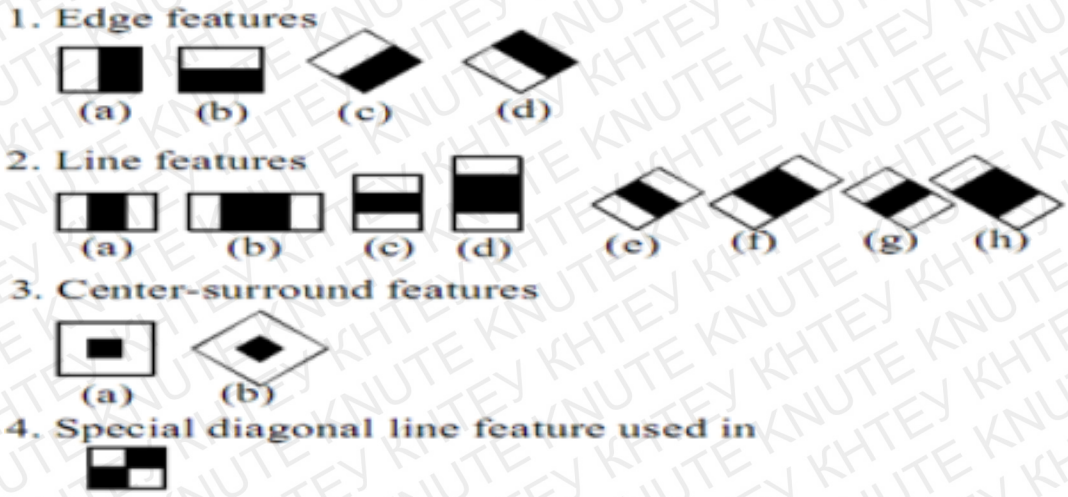

Рис. 3.5 Примітиви Хаара

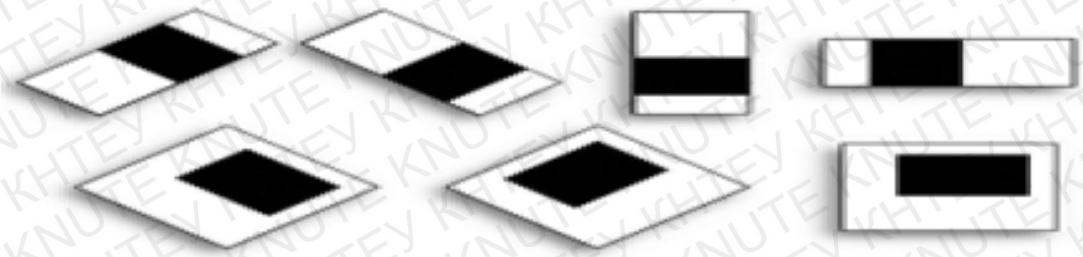

Рис. 3.6 Додаткові примітиви Хаара, котрі використовуються в OpenCV

Примітиви Хаара є набором пікселів, частина з яких білі, а частина чорні. Це перший алгоритм детектування осіб, що працює в реальному часі.

На малюнку 3.6 ми бачимо приклад знаходження ознак Хаара на зображенні.

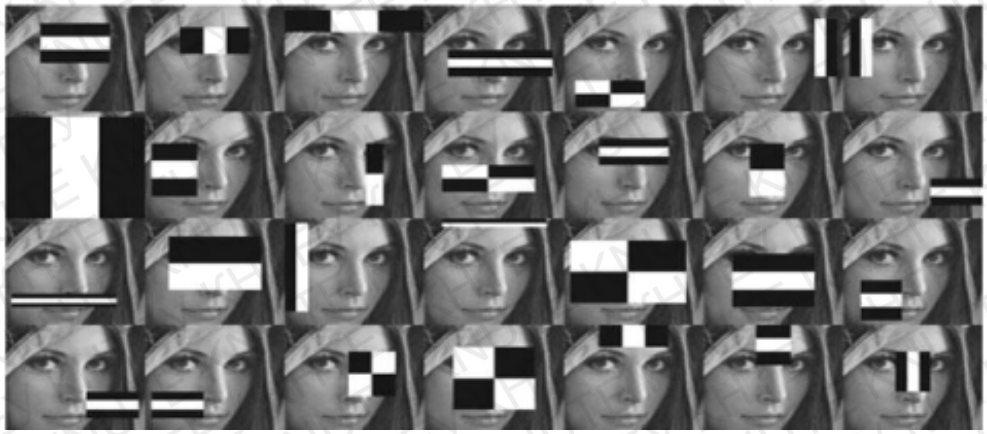

Рис. 3.6 Приклад знаходження ознак Хаара на зображенні

Для порівняння примітивів із зображенням використовується інтегральне представлення зображень. На основі робочого зображення створюється матриця, яка повинна буде збігатися за розмірами з вихідним зображенням. В елементах цієї матриці зберігаються суми інтенсивностей всіх пікселів, що знаходяться лівіше і вище даного елемента [27].

У вигляді формули це виглядає наступним чином:

$$L(x,y) = \sum_{i=0,j=0}^{i\le x,j\le y} I(i,j) \qquad (3.7)$$

Де (i, j) – яскравість пікселя вихідного зображення.

Отримана нова матриця L (x, y) зберігає в собі суму інтенсивностей пікселів в прямокутнику від (0,0) до (x, y) причому значення елемента матриці з координатою (x, y) буде максимальним і є сумою всіх попередніх елементів матриці. Отже, розрахунок нової матриці займає лінійний час, пропорційне числу пікселів в зображенні, тому інтегральне зображення прораховується за один прохід. Розрахунок матриці здійснюється за такою формулою:

$$L(x,y) = I(x,y) - L(x-1,y-1) + L(x,y-1) + L(x-1,y) \quad (3.8)$$

Ознака – це відображення f: x =>Df, де Df – безліч допустимих значень ознаки.

Якщо задані ознаки $f_1,\ldots f_n$, то вектор ознак $f_1(x)$, ... $.f_n(x)$ називається ознаковим описом об'єкта xєX.

Ознакові описи допустимо ототожнювати з самими об'єктами.

При цьому безліч X = Df_t * * Dfn називають простором ознак.

Ознаки поділяються на такі типи в залежності від безлічі Df:

1. Бінарна ознака, Df = {0,1};

2. Номінальна ознака: Df = кінцева множина;

3. Порядкова ознака: Df = кінцева впорядкована множина;

4. Кількісна ознака: Df = безліч дійсних чисел.

Обчислюваним значенням такого показника буде:

$$F = X\text{-}Y \quad (3.9)$$

де X – сума значень яскравості точок, які закриваються світлою частиною ознаки,

Y – сума значень яскравості точок, які закриваються темною частиною ознаки.

Для їх обчислення використовується поняття інтегрального зображення. Ознаки Хаара дають точкове значення перепаду яскравості по осі X і Y, відповідно.

Перевагою ознак Хаара є порівняно висока швидкість обчислення.

3.4. Пошук об’єкту за кольором

Кольорова палітра – це модель представлення кольору, заснована на використанні колірних координат, тобто будь-який колір буде представлений точкою, що має певні координати. Для зберігання цифрових зображень використовується зазвичай колірний простір RGB.

У RGB кожної з трьох осей або каналів присвоюється свій колір: червоний, зелений і синій. На кожен канал виділяється по 8 біт інформації і відповідно інтенсивність кольору на кожній осі буде приймати значення в діапазоні від 0 до 255. Всі кольори в цифровому просторі RGB виходять шляхом змішування трьох основних кольорів: червоного, зеленого та синього.

На рис. 3.7 ми бачимо кольорове зображення, представлене у вигляді 3 матриць: red, green, blue. Кожна невелика коробка являє собою піксель зображення. Експерименти показують, що RGB не завжди добре підходить для аналізу інформації, так як геометрична близькість кольорів досить далека від того, як людина сприймає близькість тих чи інших кольорів один до одного в реальних зображеннях, ці пікселі настільки невеликі, що людське око не може розрізняти їх.

Рис. 3.7 Кольорове зображення представлено з використанням матриць

Існують і інші колірні простори. Найбільш відповідним колірним простором для сегментації зображення на основі кольору є простір HSV.

Простір HSV (англ. Hue, Saturation, Value- тон, насиченість, значення) - це колірна модель, в якій координатами кольору є:

Ось Value (значення кольору) – це яскравість і позначає кількість світла. Варіюється межах 0-100 і 0-1. На відміну від RGB на нього виділено окремий канал і його значення не потрібно обчислювати кожен раз. Це чорно-біла версія зображення і з нею вже можна працювати.

Ось Hue – це колірний тон і представляється у вигляді кута. Варіюється в межах 0-360 °,

Ось Saturation це насиченість кольору і залежить від відстані від центру до краю.

Варіюється в межах 0-100 або 0-1. Чим більше цей параметр, тим «чистіше» колір, тому цей параметр іноді називають чистотою кольору. А чим ближче цей параметр до нуля, тим ближче колір до нейтрального сірого (рис. 3.8).

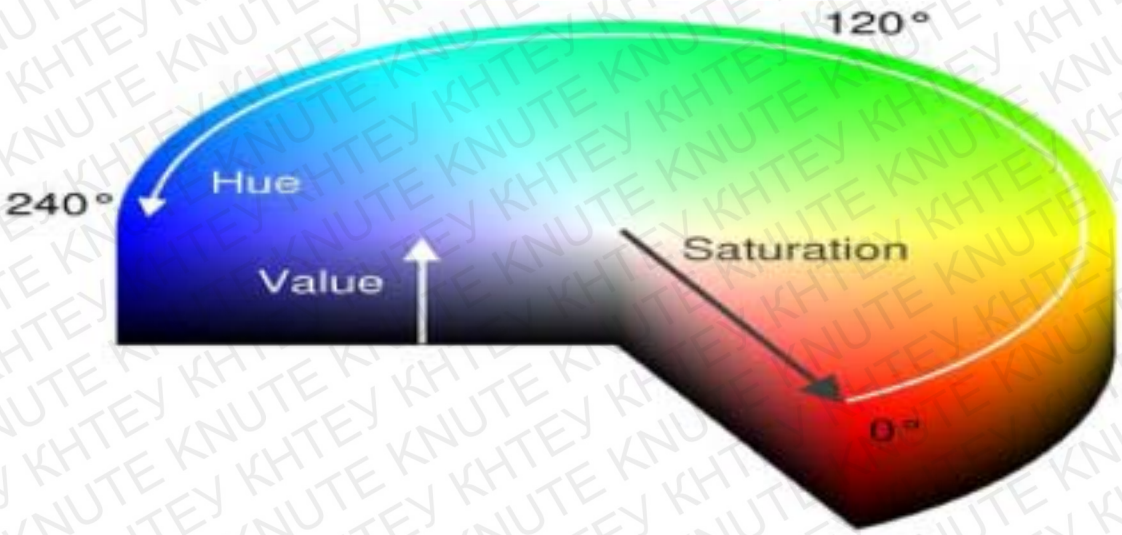

Рис. 3.8 Кольоровий простір HSV

Простір HSV набагато ближче до того, як ми уявляємо собі кольори. Наприклад, якщо показати людині в темряві червоний і зелений об'єкт, то він не зможе розрізнити колір.

У HSV відбувається те ж саме. Чим нижче по осі V ми просуваємося, тим менше стає різниця між відтінками, так як знижується діапазон значень насиченості. На схемі це виглядає як конус, на вершині якого гранично чорна точка (рис. 3.9).

Рис. 3.9 Конічне уявлення моделі HSV

В основі цього методу лежить ідея фільтрації зображення за кольором.

Метод пошуку об'єкта за кольором є одним з найпоширеніших методів. Цей метод можна застосовувати, коли об'єкт суттєво відрізняється за кольором від фону і умови освітлення змінюються мало [30].

Колір – це властивість тіл відбивати або випускати видиме випромінювання певного спектрального складу і інтенсивності. Наприклад, жовтий колір один з найяскравіших і являється останнім кольором, який бачать слабозорі люди. Тому для них приклеюють жовті кола на дверях магазинів, щоб вони розрізняли скляні двері, покривають жовтою фарбою бордюри і смуги на сходах пішохідних переходів.

Відомий колір є результатом взаємодії спектра випромінюваного світла і поверхні. На пошук об'єкта за кольором впливають безліч чинників. Освітленість це один з факторів, що впливає на пошук об'єкта за кольором. Наприклад, якщо ми будемо висвітлювати білий аркуш світлом червоної лампочки, то лист буде здаватися нам червоним.

Тому для пошуку об'єкта з таким кольором потрібні такі умови:

- цільова область повинна заздалегідь виділятися в колірному просторі і реалізуватися тільки в разі пошуку об'єктів заздалегідь відомих кольорів.
- цільова область повинна визначатися вказівками користувача і описуватися колірними термінами природної мови людини. Наприклад, синій, червоний, зелений, яскраво-жовтий, і т.д.

Висновки до розділу 3

Таким чином, перевагами використання методу сегментації є його простота використання в тих ситуаціях, коли досліджувані об'єкти значно відрізняються від решти фону по якому-небудь параметру. В свою чергу сегментація має такі недоліки: низька універсальність методів; необхідність тонкого налаштування параметрів роботи алгоритму розпізнавання в кожному окремому випадку; висока чутливість до змін освітлення сцени.

Метод зіставлення шаблонів володіє такими перевагами: хороша застосовність при аналізі сцен, в яких камера статична, і всі екземпляри шуканих об'єктів виглядають однаково. Недоліком цього методу вважається нестійка робота в разі масштабування, зрушення, повороту зображень, а також у випадках, коли об'єкт, котрий розпізнається, видно не повністю.

Метод виділення країв та детектор границь Канні є досить специфічним, оскільки він провокує такі складнощі як: товщина – отримані краї виходять товсті, розмиті; незрозумілість – досить важко визначити які лінії співвідносяться один з одним і де саме знаходяться лінії одного об'єкта.

Примітиви Хаара використовуються коли необхідно оброблювати зображення в реальному часі, так як їхньою основною перевагою є порівняно висока швидкість обчислення.

Метод пошуку об'єкта за кольором є одним з найпоширеніших методів. Цей метод можна застосовувати, коли об'єкт суттєво відрізняється за кольором від фону і умови освітлення змінюються мало. В свою чергу, для пошуку об'єкта за кольором необхідно виконати такі умови: цільова область повинна заздалегідь виділятися в колірному просторі і реалізуватися тільки в разі пошуку об'єктів заздалегідь відомих кольорів; цільова область повинна визначатися вказівками користувача і описуватися колірними термінами природної мови людини. Наприклад, синій, червоний, зелений, яскраво-жовтий, і т.д.

РОЗДІЛ 4
ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Архітектура програмного забезпечення

Програмне забезпечення було побудоване спираючись на принцип модульної архітектури програмного забезпечення, тому виникає наступний перелік модулів:

- головний модуль – запускається як процес, приймає в параметрах, які модулі будуть використовуватися в процесі роботи за принципом композиції, створює дочірні паралельні процеси для підвищеної швидкодії;
- модуль препроцесінгу – модуль, який займається первинною обробкою зображення, покращує якість вихідного зображення;
- модуль визначення ключових особливостей – займається аналізом та ідентифікацією областей на зображенні, де можуть знаходитися відомі об’єкти, виділяє їх ключові характеристики, проводить порівняння з характеристиками об’єктів зі створеного заздалегідь корпусу даних та маркує область зображення.

В контексті того, що необхідно сприймати та опрацьовувати не фотографії, а потокове відео для системи «розумний будинок», проте неможливо аналізувати власне відеопотік, необхідно доповнити цю систему модулем, який перетворює потокове відео в низку зображень, які проходять через аналіз та збираються назад у те саме відео. За об’єктами, що мають

природу непостійного положення (пересуваються з плином часу) будемо слідкувати за допомогою апарату на основі фільтра Калмана (рис. 4.1).

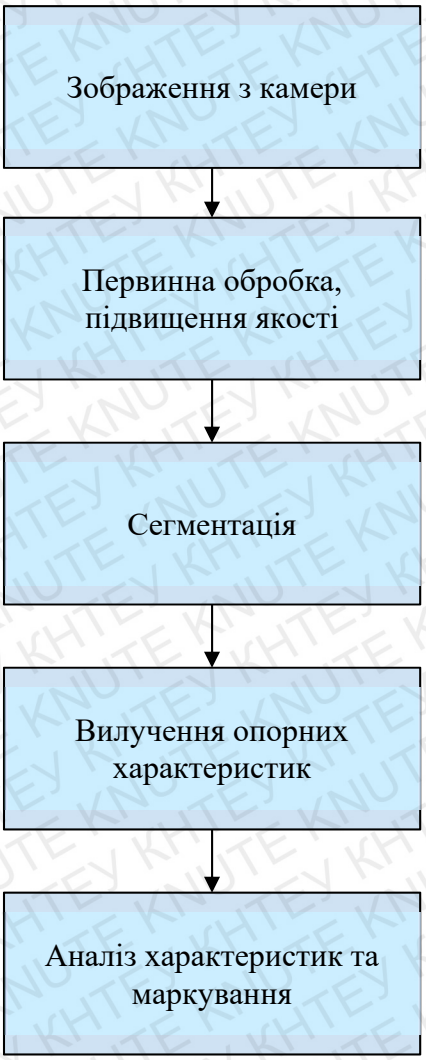

Рисунок 4.1 – Алгоритм роботи програмного забезпечення.

Така архітектура програмного забезпечення абсолютно і повністю відповідає загальній моделі розпізнавання та ідентифікації об’єктів, а на основі такої архітектури можна розробити наступний алгоритм роботи програмного забезпечення.

4.2. Модуль препроцесінгу

Модуль займається тим, що підвищує якість вихідного зображення за допомогою методів, описаних раніше в дослідженні. До того ж, щоб не втрачати результатів, модуль працює з кожним із зображень, тобто з вихідного зображення отримується наступний перелік зображень:

- вихідне зображення;
- зображення з видаленими шумами;
- зображення з підкресленими границями;
- зображення з фільтром;
- зображення з комбінованими викривленнями.

Кожне з цих зображень використовується надалі для виявлення областей, на яких знаходяться обличчя.

За рахунок мультипроцесінгу ми не втрачаємо швидкість видачі результату, тобто аналіз виконується паралельно, а не послідовно. З кожного проаналізованого зображення матимемо перелік областей з ймовірними положеннями об’єктів і в більшості випадків ці області в дійсності співпадають, але в окремих випадках одні методи викривлення оригінального зображення можуть проявити об’єкти там, де їх не було проявлено використовуючи інші методи викривлення. Таким чином підвищується надійність пошуку об’єктів за поганих умов.

4.3. Модуль визначення ключових особливостей

Цей модуль займається виявленням областей з об’єктами використовуючи дані з попереднього модуля в комбінованому вигляді, підрахунком значень ключових особливостей, використовуючи метод підтримуючих (опорних) векторів та порівнює ці дані з даними з корпусу відомих об’єктів.

Після того, як було проведено порівняння, та знайдено відомі об’єкти, області з ними виділяються та маркуються назвою об’єкта та вірогідністю приналежності до визначеного об’єкту.

4.4. Корпус даних

Корпус даних було отримано за допомогою набору зображень об’єктів, що були розташовані у різних положеннях, на різних відстанях та за різних умов освітлення, що дозволяє нам майже однозначно знаходити подібні об’єкти у режимі використання цього корпусу.

З кожного зображення вилучалися характеристики використовуючи вікна розмірами 25 на 25 пікселів, подібні зображення заздалегідь були розподілені по групах, характеристики об’єднувалися та визначалися найбільш впливові показники для кожного з об’єктів. Після цього дані за місцезнаходженням об’єктів валідувалися за допомогою розмітки маркерів, чим ітеративно досягається підвищення точності знань у корпусі. Таким чином було отримано корпус даних, структура вихідних даних зображено на рис 4.2

```
Корінь
--- дані
    --- назва об'єкта
        --- зображення
            --- img1.jpg
            --- img2.jpg
            ..........
        --- маркери
            --- img1.txt
            --- img2.txt
            ..........
        --- train.txt
        --- val.txt
```

Рис 4.2 – Первинна структура даних для корпусу

У роботі також було використано сторонню бібліотеку OpenCV, що дозволяє ідентифікувати області на зображенні, які відповідають певним об’єктам, проте ця бібліотека не має жодних знань про природу походження цих об’єктів, чим і зумовлена необхідність створення корпусу даних.

4.5. Робота програмного забезпечення

Саме програмне забезпечення може працювати незалежно від наявної відеокамери, що підключена до нього, тобто може бути запущене як на стаціонарному ПК, так і на дислокованому сервері.

Програма може працювати в трьох режимах:

- режим аналізу фотографій;
- режим аналізу відеофайлу;
- режим аналізу потокового відео.

Таким чином стає очевидно, що абсолютно немає жодної необхідності мати під’єднане апаратне забезпечення у вигляді відеокамери, оскільки має лише існувати надійний спосіб передачі даних на вхід цієї програми, щоб програма мала змогу провести виявлення областей з об’єктами та подальший аналіз.

Приклад роботи програми із зображенням та визначенням об’єктів на них наведено на рисунках 4.3-4.6.

Рисунок 4.3 – Вихідне зображення (приклад №1)

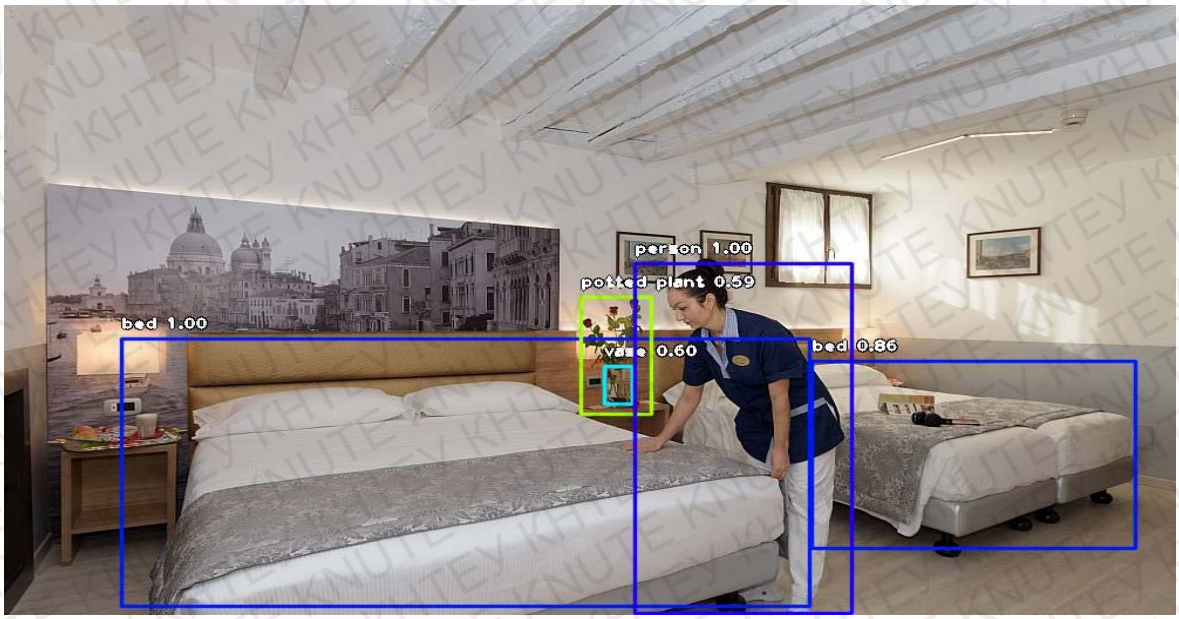

Рисунок 4.4 – Зображення з розміченими об’єктами та вірогідностями їх приналежності цим об’єктам (результат роботи програми)

Рисунок 4.5 – Вихідне зображення (приклад №2)

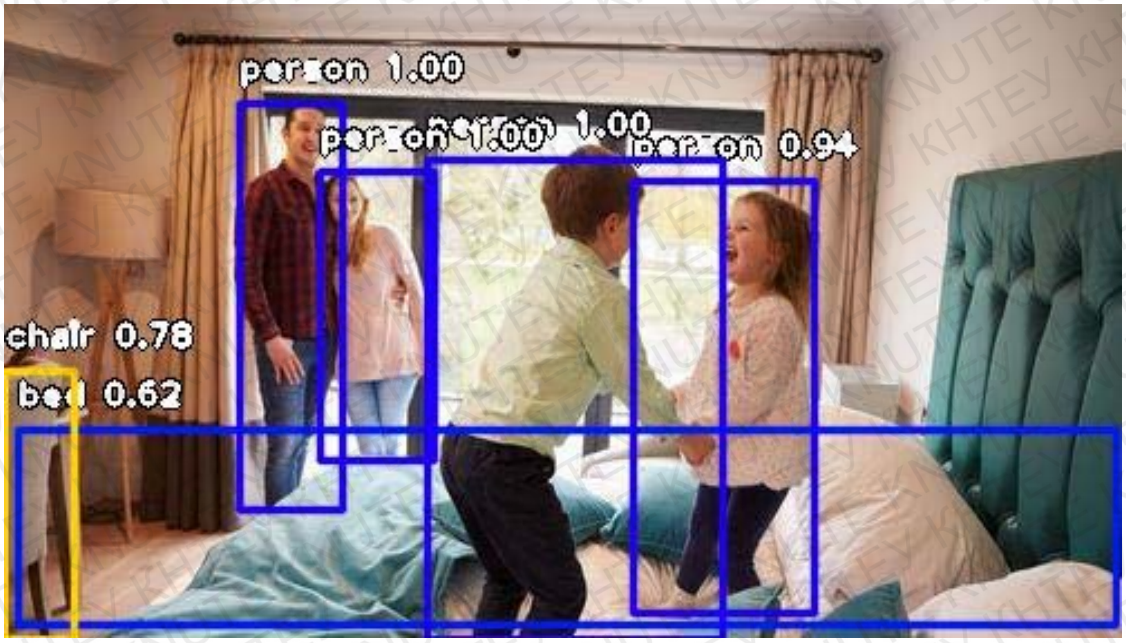

Рисунок 4.6 – Результат роботи програми з прикладом №2

Висновки до розділу 4

В цьому розділі було створено та детально описано архітектуру програмного забезпечення, алгоритм програмного забезпечення, а також наведені результати роботи програми з різними умовами освітлення та різним

наповненням кімнат. Були представлені результати роботи лише із зображеннями, проте робота з відеофайлами передбачає подібні операції лише з передумовою розкадрування відео на окремі зображення.

Кожен з модулів системи є самодостатнім та виконує певну конкретну задачу, тому може бути заміненим на використання, наприклад, іншого методу розпізнавання та ідентифікації, використання інших алгоритмів покращення зображення тощо. Тобто архітектура програмного забезпечення є достатньо гнучкою для її подальшої модифікації та інтеграції з іншими системами, що вже існують.

Корпус даних, що був створений в ході роботи над програмою був наповнений інформацією, що дозволяє доволі точно відображати дійсність та з високою ймовірністю дізнаватися про наявні об’єкти на зображенні. Таким чином можна сказати, що цей корпус даних може бути наповнений додатковими результатами аналізу зображень для виявлення нових об’єктів або тих самих, проте за інших умов освітлення чи положення.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Отже, розумний дім – житло сучасного типу, організоване для проживання людей за допомогою автоматизації і високотехнологічних пристроїв. Під «розумним» будинком слід розуміти систему, яка забезпечує безпеку і ресурсозбереження (в тому числі і комфорт) для всіх користувачів. При розробці програмного забезпечення автоматичного розпізнавання образів у підсистемі безпеки розумного дому використовується комп’ютерний зір – це теорія і технологія створення машин, які здатні виявляти, відстежувати і класифікувати об'єкти. Головною його метою є автоматизація процесу розпізнавання і обробки цифрових зображень і відеофайлів, тобто процес навчання комп'ютера розпізнавати об'єкти на статичному зображенні або в відео. Основними задачами комп’ютерного зору є розпізнавання, рух, відновлення зображень. Основними функціями виступають: отримання зображень, попередня обробка, виділення деталей, детектування/сегментація. Основними проблемами комп’ютерного зору є високорівнева обробка, ракурси, освітлення, масштаб, деформація, перекриття, рух, маскування, внутрішньокласова мінливість, контекст.

В ході дослідження була обрана мова програмування Python – проста в освоєнні і потужна мова програмування. Вона надає ефективні високорівневі структури даних, а також простий, але ефективний підхід до об'єктно-орієнтованого програмування. Також в процесі розробки використовується ImageAI – бібліотека python, яка дозволяє програмістам і розробникам програмного забезпечення легко інтегрувати новітні технології комп'ютерного

зору в свої вже існуючі або нові додатки, використовуючи лише кілька рядків коду. Також в ході розробки використовується tensorflow/keras. Tensorflow – досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain. Розпізнавання образів у підсистемі безпеки розумного дому буде проводитися за допомогою відкритої бібліотеки Open CV (Open Source Computer Vision Library) – бібліотеки комп'ютерного зору з відкритим вихідним кодом, тобто з набором алгоритмів, реалізованих у вигляді функцій CV.

В ході дослідження були проаналізовані переваги та недоліки інформаційного забезпечення автоматичного розпізнавання образів, а саме сегментація зображень та зіставлення шаблонів; виділення країв та детектор границь Канні; примітиви Хаара; пошук об’єкту за кольором.

Результатом дослідження є програмне забезпечення автоматичного розпізнавання образів у підсистемі безпеки розумного дому, Кожен з модулів системи є самодостатнім та виконує певну конкретну задачу, тому може бути заміненим на використання, наприклад, іншого методу розпізнавання та ідентифікації, використання інших алгоритмів покращення зображення тощо. Тобто архітектура програмного забезпечення є достатньо гнучкою для її подальшої модифікації та інтеграції з іншими системами, що вже існують.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bejarano A., Fernandez B., Jimeno M., Salazar A., Wightman P. Towards the Evolution of Smart Home Environments: A Survey. International Journal of Automation and Smart Technology. 2016. No. 6. pp. 105-136.
2. Benjamin Johnston, Philip de Chazal. A review of image-based automatic facial landmark identification techniques // Johnston and Chazal EURASIP Journal on Image and Video Processing. – 2018.
3. Bitterman N., Shach-pinsly D. Smart home – A challenge for architects and designers. Architectural Science Review. 2015. No. 58. pp. 266-274.
4. Dalal N., Triggs B. Histograms of oriented gradients for human detection. // Computer Vision and Pattern Recognition. – 2015. – С. 886–893.
5. General-purpose technique sheds light on inner workings of neural nets trained to process language. Larry Hardesty | MIT News Office, September 8, 2017.
6. H. Li, G. Hua, Z. Lin, J. Brandt, and J. Yang. Probabilistic elastic matching for pose variant face verification // In Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference. – 2013. – С. 3499–3506.
7. H. Liy, Z. Linz, X. Shenz, J. Brandtz, GHua, A Convolutional Neural Network Cascade for Face Detection // IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA. – 2015. – С. 5325 – 5334.
8. H.-W. Ng, S. Winkler. A data-driven approach to cleaning large face datasets // IEEE International Conference on Image Processing (ICIP), Paris, France. – Oct. 27-30/ – 2014.

9. Hamid Ouanan, Mohammed Ouanan, Brahim Aksasse. Facial Landmark Localization: past, present and future // IEEE International Colloquium on Information Science and Technology (CIST). – 2016.
10. Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, Gang Hua. A convolutional neural network cascade for face detection // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2015. – С. 5325-5334.
11. Lobaccaro G., Carlucci S., Löfström E. A Review of Systems and Technologies for Smart Homes and Smart Grids. Energies. 2016. No. 9. art. no. 348.
12. M.Nandini, P.Bhargavi, G.Raja Sekhar, Face Recognition Using Neural Network // International Journal of Scientific and Research Publications, vol. 3, no. 3. – 2013. – С. 1-5.
13. O. M. Parkhi, A. Vedaldi, A. Zisserman. Deep Face Recognition // British Machine Vision Conference. – 2015.
14. OpenCV крок за кроком. Знаходження контурів і операції з ними. [Електронний ресурс] URL: http://robocraft.ru/blog/computervision/640.html
15. OpenCV крок за кроком. Обробка зображення – детектор границь Кенні (Canny). [Електронний ресурс] URL: http://robocraft.ru/blog/computervision/484.html
16. R. F. Inglehart. Cultural evolution people's motivations are changing and reshaping the world. 2018. Cambridge University Press. 274 P.
17. Robocraft. OpenCV крок за кроком. Вступ. [Електронний ресурс] URL: http://robocraft.ru/blog/computervision/264.html
18. Schematic illustration of the convolutional neural network. – [Електронний ресурс] Режим доступу: http://www.nature.com/nature/journal/v518/n7540/fig_tab/nature14236_F1.html.
19. S-H Yooa, S-K Oha, Witold Pedrycz, Optimized face recognition algorithm using radial basis function neural networks and its practical applications // International journal on Neural Networks, volume 69. – 2015. – С. 111-125.

20. This Is What Keras Different, According To Its Author. Режим доступа: http://forbes.com/sites/quora/2016/08/25/this-is-what-makes-keras-different-according-to-its-author/#54d878ae66cf
21. Багриновський К.А., Хрусталєв Е.Ю. «Нові информаційні технології» / К.А. Багриновський, Е.Ю. Хрусталєв. – М.: «ЭКО», 2015.
22. Бродкевич В. М., Ремесло В. Я. Нейронні мережі та нейрокомп'ютери як основа відтворення процесу мислення. / В. М. Бродкевич, В. Я. Ремесло / Міжнародний науковий журнал «Інтернаука». – 2018. – № 5.
23. Візільтер Ю. В., Желтов С. Ю Обробка та аналіз цифрових зображень з прикладами на LabVIEW IMAQ Vision. / Ю. В. Візільтер, С. Ю. Желтов. – К.: 2017. – 464 с.
24. Гонзалес Р., Вудс Р. Цифрова обробка зображень. / Р. Гонзалес, Р. Вудс. – К.: 2016. – 1072 с.
25. Детектування об'єктів - пошук об'єкта за шаблоном (Template matching) [Електронний ресурс] URL: http://robocraft.ru/blog/computervision/3046.html
26. Жук Ю. О. Основи інформаційних технологій: план і програма теоретичного курсу / Ю. О. Жук. – К. : ЦіППО АПН України, 2015. – 17 с.
27. Класифікація даних методом опорних векторів [Електронний ресурс]. – URL: https://habr.com/post/105220/
28. Крилов І. В. «Інформаційні технології: теорія і практика», / І.В. Крилов – М.: «Центр», 2015 р.
29. Маккінні У. Python та аналіз даних. / У. Маккінні. – К.: 2015. – 482 с.
30. Методи комп'ютерної обробки злбражень // Під ред. В. А. Сойфера. – 2-е вид. Вип. – К. : 2016. – 784 с.
31. Виявлення стійких ознак зображення: метод SURF [Електронний ресурс]. – URL: http://twinpeppers.blogspot.com/2012/03/surf-httphabrahabrrupost103107.html
32. Саттон Р.С Навчання з підкріпленням / Саттон Р.С, Е. Г. Барто. – К.: 2014 – С. 42-96.

33. Симонович С., Євсєєв Г. Практична інформатика: універсальний курс. / С. Симонович, Г. Євсєєв. – К.:, 2013.– 480 с.
34. Тропченко А. Ю, Тропченко А. О. Методи вторинної обробки та розпізнавання зображення: навч. посібник / А.Ю. Тропченко, А.О. Тропченко. К:, 2015. – 215 с.
35. Шапіро Л., Стокман Д. Комп'ютерний зір. / Л. Шапіро, Д. Стокман. – К:, 2016 – 752с.
36. Шафрін Ю. Основи комп'ютерних технологій. / Ю. Шафрін. – К.:, 2015. – 656 с.
37. Юрченко І.В. Інформатика та програмування. Частина 1. Навчальний посібник. / І.В. Юрченко. – Чернівці: Книги–ХХІ, 2016. – 203 с.

ДОДАТКИ

Додаток А

Програма та методика тестування

Для того, щоб оцінити якість та надійність розробленого програмного забезпечення, необхідно провести його тестування. Тестування може бути кількох типів:

- мануальне - людина перевіряє самостійно всі можливі функції програми варіюючи вихідні параметри та послідовність дій
- автоматичне - програміст пише скрипти, які подають на вхід програми (або її структурних елементів) параметри та отримують від неї певний результат, який порівнюють з очікуваним.

В цьому випадку було проведено лише мануальне тестування, оскільки програма є доволі невеликою в своїй реалізації, що дозволяє за короткий час перевірити всі можливі комбінації вихідних параметрів.

Також мануальне тестування в цьому випадку виправдане тим, що неможливо було б валідувати якість програми, яка розпізнає образи (об’єкти) на зображенні та відео, за допомогою іншої подібної програми і стверджувати, що програма працює як і було задумано.

Корпус даних - одна з основних компонент програми - був створений за допомогою сету із зображень та розміток об’єктів, що розміщені на цих зображеннях, проте недостатньо лише створити цей корпус, необхідно його також валідувати. Методи валідації корпусів даних може бути абсолютно різними, проте одним з найбільш ефективних вважається крос-валідація, котрий і було використано для підтвердження надійності корпусу даних.

Суть методу крос-валідації полягає у тому, що спочатку береться вся навчальна вибірка і на основі неї будуються дані, які будуть використовуватися надалі у робочих програмах. Вибудовується певна модель, після чого навчальна вибірка поділяється, наприклад, на частини 80% та 20% цієї навчальної вибірки і менша з них використовується як тестова. Тобто

частина даних, яка використовувалася у безпосередньому створенні корпусу даних, використовується для тестування отриманих результатів.

Додаток Б

Керівництво користувача

Керівництво користувача призначено для користувачів, які планують використовувати або вже використовують ПЗ розроблене спеціально для ідентифікації об’єктів для системи “Розумний будинок”. У керівництві містяться відомості, які нададуть користувачеві всі необхідні знання, за допомогою яких користувач зможе беззаперечно користуватися наданим йому ПЗ.

Встановлення. Процес встановлення передбачає у користувача наявність ПК або сервера (або будь-який інший тип ЕОМ, що відповідає подальшим вимогам), на якому є операційна система, встановлено інтерпретатор мови Python версії 3.6 або вищої мінорної версії разом з менеджером встановлення пакетів pip та наявне безперебійне підключення до мережі Інтернет. ПЗ, яке постачається як є можна завантажити з серверу та розташувати в будь-якому каталозі, який буде зручний користувачеві.

У завантаженому каталозі міститься файл requirements.txt, з якого необхідно встановити всі необхідні для коректної роботи програми сторонні бібліотеки, які допомагають в роботі основного ПЗ.

Запуск програми. Програма має три режими роботи:

- робота із зображенням
- робота з відеофайлом
- робота із потоковим відео

Кожен з режимів роботи регулюється вихідними параметрами:

- команда main.py --image - запускає програму в режимі обробки зображень, які вона бере із теки in, що знаходиться за тим самим шляхом, що і сама програма, та розміщує в теці out
- команда main.py --video - запускає програму в режимі обробки відеофайлів, оригінали яких беруться з теки in, а оброблені потрапляють в теку out

- команда main.py --camera <camera_id> - запускає програму в режимі аналізу потокового відео з відеокамери за її ідентифікатором (ідентифікатор камери користувач має дізнатися самостійно виходячи із зовнішніх налаштувань, що не є безпосередньою та опосередкованою складовою цієї програми), результати аналізу якого записує в теку out

Користувацький інтерфейс. Програма не має графічного інтерфейсу, проте використовується інтерфейс командного рядка (cli), що дозволяє користувачеві запустити програму в будь-який зручний йому спосіб, в тому числі як фоновий або періодичний процес, а також це не накладає обмежень на використання цієї програми на ОС, які не мають графічного інтерфейсу.

Додаток В

Лістинг програмного продукту

```
import os
import cv2
import argparse

from imageai.Detection import ObjectDetection, VideoObjectDetection


path = os.getcwd()
parser = argparse.ArgumentParser()
parser.add_argument('--image', help='Image name')
parser.add_argument('--video', help='Video name')
parser.add_argument('--camera', help='From web camera')
args = parser.parse_args()

def get_detector(detection_class):
    detector = detection_class()
    detector.setModelTypeAsYOLOv3()
    detector.setModelPath(os.path.join(path , «yolo.h5»))
    detector.loadModel()
    return detector

if __name__ == «__main__»:
    if args.image:
        detector = get_detector(ObjectDetection)
        detections = detector.detectObjectsFromImage(
            input_image=os.path.join(path , 'in', args.image),
            output_image_path=os.path.join(path , 'out', args.image),
            minimum_percentage_probability=30)
        for eachObject in detections:
            print(«----------------------------------»)
            print(eachObject[«name»] , «: «, eachObject[«percentage_probability»], « : «,
eachObject[«box_points»])

    if args.camera:
        detector = get_detector(VideoObjectDetection)
        video_path = detector.detectObjectsFromVideo(
            camera_input=cv2.VideoCapture(0),
            output_file_path=os.path.join(path, «out», 'camera'),
            frames_per_second=24, log_progress=True, minimum_percentage_probability=30)
        print(video_path)

    if args.video:
        detector = get_detector(VideoObjectDetection)
        video_path = detector.detectObjectsFromVideo(
            input_file_path=os.path.join(path, 'in', args.video),
            output_file_path=os.path.join(path, 'out', args.video),
            frames_per_second=20, log_progress=True)
        print(video_path)


class VideoObjectDetection:

    def __init__(self):
        self.__modelType = ««
        self.modelPath = ««
        self.__modelPathAdded = False
        self.__modelLoaded = False
```

```
        self.__detector = None
        self.__input_image_min = 1333
        self.__input_image_max = 800
        self.__detection_storage = None


        self.numbers_to_names = {0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus',
                                 6: 'train',
                                 7: 'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign',
                                 12: 'parking meter',
                                 13: 'bench', 14: 'bird', 15: 'cat', 16: 'dog', 17: 'horse', 18: 'sheep', 19: 'cow',
                                 20: 'elephant',
                                 21: 'bear', 22: 'zebra', 23: 'giraffe', 24: 'backpack', 25: 'umbrella', 26: 'handbag',
                                 27: 'tie',
                                 28: 'suitcase', 29: 'frisbee', 30: 'skis', 31: 'snowboard', 32: 'sports ball',
                                 33: 'kite',
                                 34: 'baseball bat', 35: 'baseball glove', 36: 'skateboard', 37: 'surfboard',
                                 38: 'tennis racket',
                                 39: 'bottle', 40: 'wine glass', 41: 'cup', 42: 'fork', 43: 'knife', 44: 'spoon',
                                 45: 'bowl',
                                 46: 'banana', 47: 'apple', 48: 'sandwich', 49: 'orange', 50: 'broccoli', 51: 'carrot',
                                 52: 'hot dog',
                                 53: 'pizza', 54: 'donut', 55: 'cake', 56: 'chair', 57: 'couch', 58: 'potted plant',
                                 59: 'bed',
                                 60: 'dining table', 61: 'toilet', 62: 'tv', 63: 'laptop', 64: 'mouse', 65: 'remote',
                                 66: 'keyboard',
                                 67: 'cell phone', 68: 'microwave', 69: 'oven', 70: 'toaster', 71: 'sink',
                                 72: 'refrigerator',
                                 73: 'book', 74: 'clock', 75: 'vase', 76: 'scissors', 77: 'teddy bear',
                                 78: 'hair dryer',
                                 79: 'toothbrush'}

        # Unique instance variables for YOLOv3 model
        self.__yolo_iou = 0.45
        self.__yolo_score = 0.1
        self.__yolo_anchors = np.array(
            [[10., 13.], [16., 30.], [33., 23.], [30., 61.], [62., 45.], [59., 119.], [116., 90.], [156., 198.],
             [373., 326.]])
        self.__yolo_model_image_size = (416, 416)
        self.__yolo_boxes, self.__yolo_scores, self.__yolo_classes = «», «», «»
        self.sess = K.get_session()

        # Unique instance variables for TinyYOLOv3.
        self.__tiny_yolo_anchors = np.array(
            [[10., 14.], [23., 27.], [37., 58.], [81., 82.], [135., 169.], [344., 319.]])

    def setModelTypeAsRetinaNet(self):
        self.__modelType = «retinanet»


    def setModelPath(self, model_path):
        if (self.__modelPathAdded == False):
            self.modelPath = model_path
            self.__modelPathAdded = True

    def loadModel(self, detection_speed=«normal»):
        if (self.__modelLoaded == False):

            frame_detector = ObjectDetection()

            if (self.__modelType == «retinanet»):
                frame_detector.setModelTypeAsRetinaNet()
            elif (self.__modelType == «yolov3»):
```

```
            frame_detector.setModelTypeAsYOLOv3()
        elif (self.__modelType == «tinyyolov3»):
            frame_detector.setModelTypeAsTinyYOLOv3()
        frame_detector.setModelPath(self.modelPath)
        frame_detector.loadModel(detection_speed)
        self.__detector = frame_detector
        self.__modelLoaded = True


    def detectObjectsFromVideo(self, input_file_path=««, camera_input=None, output_file_path=««, frames_per_second=20,
                               frame_detection_interval=1, minimum_percentage_probability=50, log_progress=False,
                               display_percentage_probability=True, display_object_name=True, save_detected_video=True,
                               per_frame_function=None, per_second_function=None, per_minute_function=None,
                               video_complete_function=None, return_detected_frame=False, detection_timeout = None, thread_safe=False):

        if (input_file_path == ««  and camera_input == None):
            raise ValueError(
                «You must set 'input_file_path' to a valid video file, or set 'camera_input' to a valid camera»)
        elif (save_detected_video == True and output_file_path == ««):
            raise ValueError(
                «You must set 'output_video_filepath' to a valid video file name, in which the detected video will be saved. If you don't intend to save the detected video, set 'save_detected_video=False'«)

        else:
            try:

                output_frames_dict = {}
                output_frames_count_dict = {}

                input_video = cv2.VideoCapture(input_file_path)
                if (camera_input != None):
                    input_video = camera_input

                output_video_filepath = output_file_path + '.avi'

                frame_width = int(input_video.get(3))
                frame_height = int(input_video.get(4))
                output_video = cv2.VideoWriter(output_video_filepath, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'),
                                               frames_per_second,
                                               (frame_width, frame_height))

                counting = 0
                predicted_numbers = None
                scores = None
                detections = None

                detection_timeout_count = 0
                video_frames_count = 0


                while (input_video.isOpened()):
                    ret, frame = input_video.read()

                    if (ret == True):

                        video_frames_count += 1
                        if (detection_timeout != None):
                            if ((video_frames_count % frames_per_second) == 0):
                                detection_timeout_count += 1
```

```
    if (detection_timeout_count >= detection_timeout):
        break

output_objects_array = []

counting += 1

if (log_progress == True):
    print(«Processing Frame : «, str(counting))

detected_copy = frame.copy()

check_frame_interval = counting % frame_detection_interval

if (counting == 1 or check_frame_interval == 0):
    try:
        detected_copy, output_objects_array = self.__detector.detectObjectsFromImage(
            input_image=frame, input_type=«array», output_type=«array»,
            minimum_percentage_probability=minimum_percentage_probability,
            display_percentage_probability=display_percentage_probability,
            display_object_name=display_object_name)
    except:
        None

output_frames_dict[counting] = output_objects_array

output_objects_count = {}
for eachItem in output_objects_array:
    eachItemName = eachItem[«name»]
    try:
        output_objects_count[eachItemName] = output_objects_count[eachItemName] + 1
    except:
        output_objects_count[eachItemName] = 1

output_frames_count_dict[counting] = output_objects_count

detected_copy = cv2.cvtColor(detected_copy, cv2.COLOR_BGR2RGB)

if (save_detected_video == True):
    output_video.write(detected_copy)

if (counting == 1 or check_frame_interval == 0):
    if (per_frame_function != None):
        if (return_detected_frame == True):
            per_frame_function(counting, output_objects_array, output_objects_count,
                               detected_copy)
        elif (return_detected_frame == False):
            per_frame_function(counting, output_objects_array, output_objects_count)

if (per_second_function != None):
    if (counting != 1 and (counting % frames_per_second) == 0):

        this_second_output_object_array = []
        this_second_counting_array = []
        this_second_counting = {}

        for aa in range(counting):
            if (aa >= (counting – frames_per_second)):
                this_second_output_object_array.append(output_frames_dict[aa + 1])
                this_second_counting_array.append(output_frames_count_dict[aa + 1])

        for eachCountingDict in this_second_counting_array:
            for eachItem in eachCountingDict:
```

```
                        try:
                            this_second_counting[eachItem] = this_second_counting[eachItem] + \
                                                             eachCountingDict[eachItem]
                        except:
                            this_second_counting[eachItem] = eachCountingDict[eachItem]

                    for eachCountingItem in this_second_counting:
                        this_second_counting[eachCountingItem] =
int(this_second_counting[eachCountingItem] / frames_per_second)

                    if (return_detected_frame == True):
                        per_second_function(int(counting / frames_per_second),
                                            this_second_output_object_array, this_second_counting_array,
                                            this_second_counting, detected_copy)

                    elif (return_detected_frame == False):
                        per_second_function(int(counting / frames_per_second),
                                            this_second_output_object_array, this_second_counting_array,
                                            this_second_counting)

                if (per_minute_function != None):

                    if (counting != 1 and (counting % (frames_per_second * 60)) == 0):

                        this_minute_output_object_array = []
                        this_minute_counting_array = []
                        this_minute_counting = {}

                        for aa in range(counting):
                            if (aa >= (counting – (frames_per_second * 60))):
                                this_minute_output_object_array.append(output_frames_dict[aa + 1])
                                this_minute_counting_array.append(output_frames_count_dict[aa + 1])

                        for eachCountingDict in this_minute_counting_array:
                            for eachItem in eachCountingDict:
                                try:
                                    this_minute_counting[eachItem] = this_minute_counting[eachItem] + \
                                                                     eachCountingDict[eachItem]
                                except:
                                    this_minute_counting[eachItem] = eachCountingDict[eachItem]

                        for eachCountingItem in this_minute_counting:
                            this_minute_counting[eachCountingItem] =
int(this_minute_counting[eachCountingItem] / (frames_per_second * 60))

                        if (return_detected_frame == True):
                            per_minute_function(int(counting / (frames_per_second * 60)),
                                                this_minute_output_object_array, this_minute_counting_array,
                                                this_minute_counting, detected_copy)

                        elif (return_detected_frame == False):
                            per_minute_function(int(counting / (frames_per_second * 60)),
                                                this_minute_output_object_array, this_minute_counting_array,
                                                this_minute_counting)


                else:
                    break

            if (video_complete_function != None):

                this_video_output_object_array = []
                this_video_counting_array = []
```

```
                    this_video_counting = {}

                    for aa in range(counting):
                        this_video_output_object_array.append(output_frames_dict[aa + 1])
                        this_video_counting_array.append(output_frames_count_dict[aa + 1])

                    for eachCountingDict in this_video_counting_array:
                        for eachItem in eachCountingDict:
                            try:
                                this_video_counting[eachItem] = this_video_counting[eachItem] + \
                                                                eachCountingDict[eachItem]
                            except:
                                this_video_counting[eachItem] = eachCountingDict[eachItem]

                    for eachCountingItem in this_video_counting:
                        this_video_counting[eachCountingItem] = int(this_video_counting[eachCountingItem] /
counting)

                    video_complete_function(this_video_output_object_array, this_video_counting_array,
                                            this_video_counting)

                input_video.release()
                output_video.release()

                if (save_detected_video == True):
                    return output_video_filepath

            except:
                raise ValueError(
                    «An error occured. It may be that your input video is invalid. Ensure you specified a proper
string value for 'output_file_path' is 'save_detected_video' is not False. «
                    «Also ensure your per_frame, per_second, per_minute or video_complete_analysis function is
properly configured to receive the right parameters. «)
```