Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бабій Юрій Васильович

«Алгоритми розпізнавання дорожніх знаків для мобільних пристроїв / Road signs recognition algorithms for mobile devices»

Студент групи КІм – 21
Бабій Юрій Васильович
_____

Науковий керівник
к.т.н., доцент Мельник Г.М.
_____

Тернопіль – 2020

РЕЗЮМЕ

Кваліфікаційна робота на тему “ Алгоритми розпізнавання дорожніх знаків для мобільних пристроїв” зі спеціальності 123 «Комп’ютерна інженерія» освітнього ступеня «магістр» написана обсягом 65 сторінок пояснюючої записки, 44 рисунків, 5 таблиць, 3 додатки.

Метою кваліфікаційної роботи є розробка алгоритмів розпізнавання дорожніх знаків та побудова системи автоматизованого аналізу зображень.

Методи дослідження включають методи: штучного інтелекту, комп'ютерного зору.

Результатом кваліфікаційної роботи є система розпізнавання дорожніх знаків для мобільних пристроїв. На першому (етап сегментації) було створено два каскадні класифікатори, один для круглих знаків а інший для трикутних та знаку «пішохідний перехід». Для збільшення швидкості локалізації було обрано зону інтересу, це дозволило скороти час виконання даного етапу у більш ніж два рази.

Розроблено та реалізовано алгоритми сегментації, на основі каскаду класифікаторів, та оптимізовано його для роботи із мобільними пристроями. Реалізовано систему розпізнавання дорожніх знаків для мобільних пристроїв на Android.

Орієнтовні напрямки розвитку досліджень: подальша оптимізація коду для використання в мобільних пристроях.

Ключові слова: АНАЛІЗ ЗОБРАЖЕНЬ, СЕГМЕНТАЦІЯ, КЛАСИФІКАЦЯ, НЕЙРОННА МЕРЕЖА, ДОРОЖНІ ЗНАКИ, РОЗПІЗНАВАННЯ, КАСКАД КЛАСИФІКАТОРІВ.

RESUME

Qualification work «Road signs recognition algorithms for mobile devices» in the specialty 123 "Computer Engineering" of the Master degree contains 65 pages note, 44 figures, 5 tables, 3 appendices.

The purpose of the final qualification work is to develop algorithms for recognizing road signs and build a system of automated image analysis.

Research methods include methods: artificial intelligence, computer vision.

The result of this final qualification work is a system of road sign recognition for mobile devices. At the first (segmentation stage) two cascade classifiers were created, one for round signs and the other for triangular signs and the pedestrian crossing sign. To increase the speed of localization, the area of interest was selected, which reduced the execution time of this stage by more than two times.

Segmentation algorithms based on a cascade of classifiers have been developed and implemented, and it has been optimized for work with mobile devices. Classification algorithms have been developed and implemented using a trained convolutional neural network. Implemented a road sign recognition system for mobile devices on Android.

Further research: optimization of code for use in mobile devices.

Keywords: IMAGE ANALYSIS, SEGMENTATION, CLASSIFICATION, NEURAL NETWORK, ROAD SIGNS, RECOGNITION, CASCADE OF CLASSIFIERS.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі обов'язковим атрибутом в організації дорожнього руху є дорожні знаки. Вони інформують водіїв про небезпечні ділянки дороги, вказують напрямок руху, забороняють або дають право проїзду, зобов'язують знизити швидкість, а також виконують безліч інших корисних завдань.

З кожним роком число водіїв постійно збільшується. У зв'язку із великою кількістю автомобілів необхідно, щоб дорожній рух став безпечнішим. Для цієї мети розробляються просунуті системи допомоги водієві "Advanced Driving Assistance Systems" (ADAS), які поступово вбудовуються в деякі сучасні високотехнологічні серії автомобілів.

Система автоматичного розпізнавання дорожніх знаків є найважливішою частиною ADAS. Вона покликана повідомляти водія про наявність дорожніх знаків на дорозі. Система може допомогти водієві дотримуватися встановленого на ділянці обмеження на проїзд, обгін і т.д. Подібні системи вперше з'явилися в кінці 2008 року в автомобілях BMW 7Series, а через рік і в MercedesBenz S Class. Ці системи вміли розпізнавати тільки знаки обмеження швидкості. Пізніше їх вбудували в такі серії як Volkswagen Phaeton і Opel Astra з додатковою функцією розпізнавання знаку "обгін заборонений" [1]. Загалом, всі провідні світові виробники автомобілів намагаються вмонтувати подібні технології в свої продукти. Однак більшість людей не може дозволити собі ціну подібних автомобілів і, як наслідок, не може скористатися даною технологією. З іншого боку, на сьогоднішній день практично у кожного є смартфон з камерою. До того ж обчислювальні потужності телефонів з кожним днем збільшуються і вже наближаються до комп'ютерних. Тому наявність системи автоматичного розпізнавання дорожніх знаків в телефоні може вирішити цю проблему.

Завданням роботи є розробка системи автоматичного розпізнавання попереджувальних, заборонних знаків та знаку "Пішохідний перехід", для

мобільних пристроїв, на платформі Android, здатної працювати у режимі реального часу. Було обрані ці знаки, тому що саме вони позначають найбільш важливу інформацію для водія, і їх порушення призводить до найбільш небезпечних ситуацій на дорозі.

Метою кваліфікаційної роботи є розробка алгоритмів розпізнавання дорожніх знаків на основі штучних нейронних мереж.

Об’єкт дослідження – процес аналізу зображень дорожніх знаків

Предмет дослідження – методи і алгоритми аналізу зображень на основі загорткових нейронних мереж.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

- провести огляд систем розпізнавання дорожніх знаків;
- провести огляд методів та алгоритмів розпізнавання зображень;
- розробити узагальнений алгоритм роботи системи;
- розробити структуру загорткової нейронної мережі;
- програмно реалізувати розроблені алгоритми;
- провести експериментальне дослідження розроблених алгоритмів.

Методи досліджень базуються на використанні методів гістограмного аналізу зображень, методів комп’ютерного зору, положень теорії алгоритмів і аналітичної геометрії.

Наукова новизна одержаних результатів. Розроблено алгоритми розпізнавання зображень дорожніх знаків на основі штучних нейронних мереж, що дозволило підвищити точність розпізнавання.

Практичне значення отриманих результатів. Спроектовано систему розпізнавання дорожних знаків у вигляді мобільного додатку для систем на базі Andriod.

Публікації та апробація випускної кваліфікаційної роботи. Отримані результати апробовані в межах ІІІ науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» Західноукраїнського національного економічного університету та опубліковано дві тези доповіді по темі роботи [2,3].

Кваліфікаційна робота складається із трьох розділів, висновків, списку використаної літератури та додатків. У першому розділі сформовано основні вимоги до системи розпізнавання, розглянуті найбільш популярні методи розпізнавання, такі як: метод нейронних мереж; метод SURF; метод опорних векторів; метод Random forest та каскад класифікаторів. Розглянуті технології розробки для мобільних пристроїв.

У другому розділі сформовано основні вимоги до системи розпізнавання, розглянуті найбільш популярні методи розпізнавання, такі як: метод нейронних мереж; метод SURF, метод опорних векторів, метод Random forest та каскад класифікаторів. Розглянуто їх недоліки, відповідність поставленим вимогам. Розглянуті технології розробки для мобільних пристроїв.

У третьому розділі розроблено та реалізовано алгоритми в складі системі розпізнавання дорожніх знаків для мобільних пристроїв на Android. Проведено експериментальне дослідження розроблених алгоритмів.

1 АНАЛІЗ АЛГОРИТМІВ І ПРОГРАМНО-АПАРАТНИХ ЗАСОБІВ РОЗПІЗНАВАННЯ

1.1 Вимоги до системи розпізнавання дорожніх знаків

Завдання розпізнавання дорожніх знаків активно досліджується вже довгий час. Якщо буде створено метод, який працює з достатньою точністю, його можна використовувати в системах допомоги водієві, інвентаризації об'єктів придорожньої інфраструктури або для автоматизованого створення навігаційних карт.

Дорожні знаки зроблені, щоб бути помітними і легко розпізнаватися і це робить їх хороших об'єктом для автоматичного розпізнавання. У той же час автоматичні алгоритми повинні розпізнавати знаки в умовах внутрішньокласовій мінливості, зміни освітленості, положення точки спостереження, розмиття і перекриттів. Кращі на сьогоднішній день алгоритми спираються на машинне навчання і вимагають наявності великої і репрезентативною навчальної колекції, щоб обійти всі перераховані вище проблеми.

Розробка призначена для збільшення безпеки на дорогах, а також полегшення процесу водіння. Інженери створюють рішення, які будуть автоматично розпізнавати дорожні знаки, фіксувати інформацію про допустимі швидкостях і обмеженнях, включаючи напрямок руху, наявність перехресть, поїзних перегонів та інших даних [4].

Чим більше попереджень отримує система від зовнішнього середовища, тим надійніше стає автомобіль і процес водіння. Водієві фізично важко стежити за всіма параметрами дороги, особливо в тривалих поїздках. Програмне рішення здатне вирішити проблему з неуважністю і зменшити вплив людського фактору під час руху.

Оскільки розпізнавання дорожніх знаків є однією зі основних складових, необхідних для безпілотних автомобілів та коректної роботи системи, повинна

бути розвинута транспортна інфраструктура. Автомобіль повинен самостійно визначати розмітку, обмеження, знаки і умови руху. За відсутності дорожніх знаків, система не буде працювати.

На рисунку 1.1 зображено розпізнавання системою знаку обмеження швидкості.

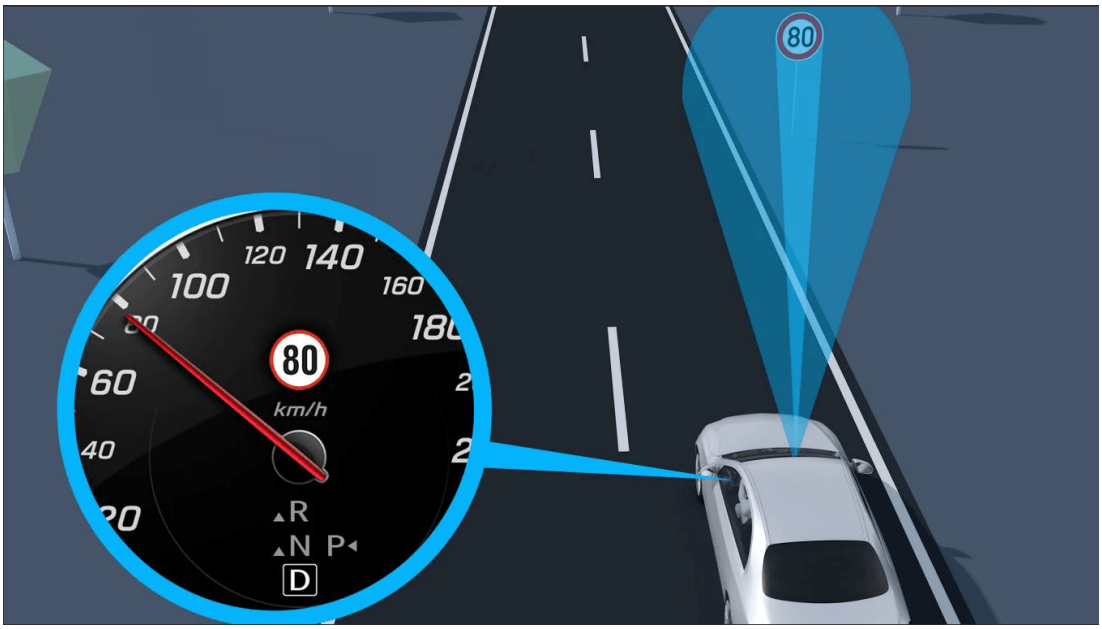

Рисунок 1.1 – Розпізнавання знаку обмеження швидкості

Оскільки причиною більшості аварій вважається порушення швидкісного режиму, інженери автомобільних компаній поставили собі за мету викорінити цю проблему. Для цього в машину встановлюється система розпізнавання знаків. Основні функції якої:

– визначення та підтвердження інформації про дорожні знаки;

– пошук інформації в базі даних і повідомлення водія;

– попередження за допомогою світлового або звукового сигналу, якщо швидкість руху не змінюється.

Можливості систем залежать від поколінь розробки. Початкові рішення могли розпізнавати тільки обмежувачі швидкості, заборони на обгін і деякі додаткові знаки. Сучасні системи можуть розшифровувати інформацію про

житлові зони, початок та кінець населеного пункту, кінець зони обмежень, в'їзд заборонений і багато іншого [5].

Покроковий опис логіки виявлення об'єктів:

– камера аналізує навколишнє середовище і зчитує дані про дорожні знаки;

– система виявляє форму, схожу на знак;

– розпізнавання кольору і наявності додаткових символів;

– пошук відповідності в базі даних;

– інформування водія.

Послідовність розпізнавання типу знаку:

– визначення форми: коло, прямокутник, квадрат;

– аналіз колірної гами;

– зчитування символів або написів на знаку.

1.2 Існуючі методи розпізнавання

Існує безліч методів розпізнавання образів на зображеннях. Розуміння даних методів важливо для вирішення різного роду завдань. По-перше, важливо розуміння теорії розпізнавання образів. Основними термінами є: клас – множина об'єктів, що мають загальні властивості, класів може бути необмежена кількість; класифікація – процес призначення міток класу об'єктів, відповідно до деякого опису властивостей цих об'єктів; класифікатор – пристрій, який в якості вхідних даних отримує набір ознак об'єкта, а в якості результату видає мітку класу; верифікація – процес зіставлення примірника об'єкта з однією моделлю об'єкта або описом класу; ознака – кількісний опис тієї чи іншої властивості досліджуваного предмета або явища; простір ознак – це N-мірний простір, визначене для даної задачі розпізнавання, де N – фіксоване число вимірюваних ознак для будь-яких об'єктів. Вектор з простору ознак x, відповідний об'єкту

завдання розпізнавання це N-мірний вектор з компонентами (x_1, x_2, ..., x_N), які є значеннями ознак для даного об'єкта.

Таким чином, вся задача розпізнавання зводиться до виділення істотних ознак для кожного класу і, в кінцевому підсумку, віднесення вхідних даних до одного з них за допомогою виявлення ключових ознак в оригінальному зображенні.

Тобто розпізнавання образів можна розділити на кілька завдань, таких, як:

– отримання вхідних даних, за допомогою сенсорів, камер відеоспостереження, добірок даних;

– первинна обробка зображень така, як нормалізація даних, фільтрація шумів, виявлення ознак;

– формування векторів ознак, за допомогою вибору найбільш значущих ознак, за допомогою яких можна виділити непересічні безлічі класів;

– класифікація чи пророцтво на основі отриманих даних про класи.

Дані етапи представлені на рисунку 1.2.

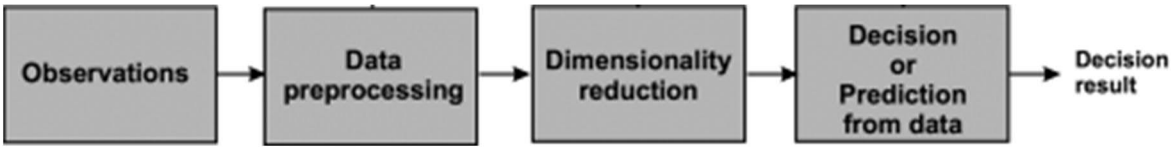

Рисунок 1.2 – Етапи розпізнавання образів

Метод нейронних мереж. Нейронні мережі дозволяють вирішувати широке коло завдань і представляють із себе структуру з декількох шарів – штучних нейронів (обчислювальних елементів) і зв’язків між ними. Структура імітує структуру і властивості організації нервової системи живих організмів. Нейронна мережа отримує на вхід набір сигналів і на виході видає відповідну відповідь (вихідні сигнали), які описують рішення деякої задачі.

На рисунку 1.3 представлена схема штучної нейронної мережі.

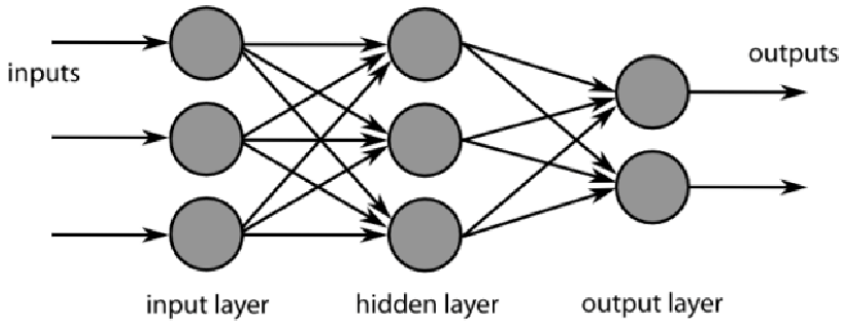

Рисунок 1.3 – Схема нейронної мережі

Щоб описати принцип роботи мережі, уявімо штучний нейрон, схема якого зображена на рисунку 1.4.

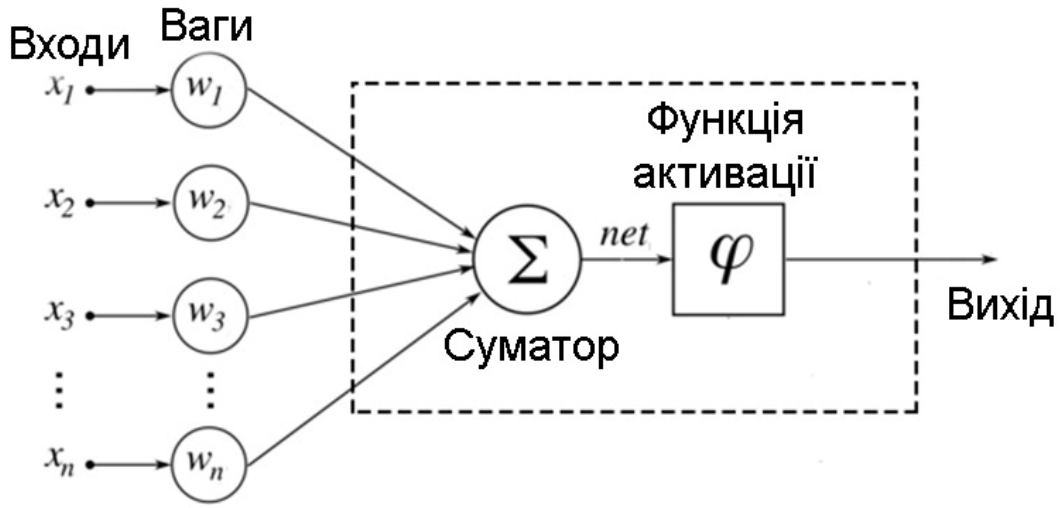

Рисунок 1.4 – Структура штучного нейрона

Відповідно на кожен нейрон вхідного шару надходить сигнал, який множиться на відповідну йому вагу. Функцією активації є або порогове значення, передає на вихід одиничний сигнал, або сигмоїдальною функцією, яка перетворює значення суми всіх, хто прийде сигналів, в число, що знаходиться в діапазоні від 0 до 1 [6].

Таким чином на виході мережі виходить розподіл усіх значення, схоже на результат методу Баєса. У порівнянні з лінійними методами статистики,

нейронні мережі дозволяють ефективно будувати нелінійні залежності, точніше описують набори даних. Що ж стосується байєсівського класифікатора, який будує квадратичну розділяючу поверхню, нейронна мережа може побудувати поверхню більш високого порядку. Висока нелінійність розділяє поверхні наївного байєсівського класифікатора (він не використовує коваріаційні матриці класів, як класичний Баєс, а аналізує локальні щільності ймовірності) вимагає значного сумарного числа прикладів для можливості оцінювання ймовірностей при кожному поєднанні інтервалів значень змінних – тоді як нейронна мережа навчається на всій вибірці даних, не фрагментуючи її, що підвищує адекватність налаштування мережі.

Метод Speeded Up Robust Features (SURF) – дозволяє виконувати пошук особливих точок на зображенні і вираховувати їх дескриптори, які будуть інваріантні до масштабу і обертанню. Пошук здійснюється з допомогою матриці Гессе. Гессіан зображення, який вираховується як детермінант матриці Гессе, в точках максимального перепаду яскравості досягає екстремуму. Це дозволяє знайти на зображенні такі особливі точки, як кути, краї ліній та інше.

Для обчислення матриці Гессе і фільтру Харара зручніше перейти до інтегрального представлення зображення. Інтегральне представлення зображення являє собою матрицю, розмірність якої співпадає із розмірністю вхідного зображення. Елементи матриці розраховуються за наступною формулою:

$$D(x,y) = \sum_{i,j=0}^{x,y} S(i,j), \qquad (1.1)$$

де $S(i,j)$ – яскравість пікселя вхідного зображення.

Кожний елемент матриці представляє собою суму яскравостей в прямокутнику від (0,0) до (x,y). Перейшови до такого представлення, зручно і легко можна вирахувати загальну яскравість довільної прямокутної області ABCE:

$$I_{ABCE} = D(A) + D(B) - D(C) - D(E), \tag{1.2}$$

де *ABCE* – вершини прямокутника.

Матриця Гессе для двовимірної функції і її детермінанта вираховуються наступною формулою:

$$H(S(x,y)) = \begin{bmatrix} \frac{\partial^2 S}{\partial x^2} & \frac{\partial^2 S}{\partial x \partial y} \\ \frac{\partial^2 S}{\partial x \partial y} & \frac{\partial^2 S}{\partial y^2} \end{bmatrix}, \tag{1.3}$$

$$\det(H) = \frac{\partial^2 S}{\partial x^2}\frac{\partial^2 S}{\partial y^2} - \left(\frac{\partial^2 S}{\partial x \partial y}\right)^2. \tag{1.4}$$

Значення визначника цієї матриці досягає екстремуму у точках локального максимуму і мінімуму яскравості зображення. Елементи матриці Гессе вираховуються, по суті, як згортка пікселів зображення і маски фільтра. Проте, на думку авторів, використання обчислень лапласіан гауссіана для цього у початковому виді неефективно, тому в SURF використовується бінарний аналог фільтрів для обчислення матриці Гессе. Ці маски більш стійкі до обертів і їх можна легко вирахувати з допомогою інтегральної матриці:

$$\det\left(H_{approx}\right) = S_{xx}S_{yy} - \left(0.9D_{xy}\right)^2, \tag{1.5}$$

де S_{xx}, S_{yy}, S_{xy} – згортки по фільтрах.

Для віднесення точки до розряду особливих, до гауссіану застосовується поріг, вище якого, точка вважається особливою. Також слід зазначити, що гауссіан є величиною похідною і не залежить від абсолютного значення яскравості, що дозволяє виявляти ті ж особливості на зображеннях з іншим

рівнем освітленості. Як вже говорилося, гессіан залежить від масштабу зображення, тому для пошуку особливих точок по черзі перебираються різні масштаби фільтрів. Розміри таких фільтрів не можуть приймати довільні значення, інакше їх було б дуже багато. Допустимі розміри: 9, 15, 21, 27 і т.д. Для оптимізації перебору фільтрів в методі застосовується принцип поділу всього діапазону масштабів на октави. Причому, на октаву приходиться кілька фільтрів, так як серед різних масштабів і октав одна точка може мати кілька локальних максимумів гессіана.

Виходячи з цього, крок розміру фільтра в першій октаві становить 6, у другій 12, у третій 24 пікселя і т.д., причому октави перекривають один одного для збільшення надійності знаходження точок. На думку авторів, для зображення з роздільною здатністю 1024x768 пікселів потрібно близько 5-6 октав.

Для обчислення локального екстремуму гессіана використовується метод сусідніх точок 3x3x3. Суть методу полягає в тому, що точка є локальним максимумом, якщо її гессіан більше гессіана сусідніх точок за принципом 8-пов'язаності в тому ж масштабі, а також для сусідніх точок масштабом менше і більше в даній октаві. При цьому зрозуміло, що октава повинна містити не менше трьох фільтрів.

Для підвищення швидкості роботи методу фільтри октави обчислюються не для всіх пікселів зображення. Перша октава рахується для кожного другого пікселя, друга – для кожного четвертого і т.д., так як розмір максимумів пропорційний масштабу фільтра.

Так як алгоритм перебирає не всі пікселі зображення, реальний максимум і обчислений можуть розходитися. Для знаходження точного максимуму використовується інтерполяція знайдених гессіанів куба 3x3x3 квадратичною функцією [7].

Для того щоб дескриптори особливих точок були інваріантні до повороту, необхідно визначити переважну орієнтацію їх градієнта або вектор орієнтації. Для початку обчислюються градієнти в пікселях в околиці особливої точки. Для цього застосовується фільтр Хаара. Він дає точкове значення перепаду

яскравості по осях dX і dY. Дані значення досить легко обчислюються за допомогою інтегральної матриці. Далі всі знайдені значення в вигляді точок наносяться на площину dXdY і обчислюється вектор, що зображає пріоритетний напрямок в області особливої точки. Слід зазначити, що інваріантність щодо обертання не завжди потрібна, і метод допускає не розраховувати її.

Для обчислення дескриптора вибирається прямокутна область навколо особливою точки. Причому, квадрат орієнтується вздовж напрямку особливості. Дескриптор формується з опису 16 квадрантів навколо особливою точки. Потрібно відзначити, що саме зображення при розрахунку не перевертається, фільтр вважається в звичайних координатах, а вже отримані координати градієнта, повертаються відповідно до отриманої раніше орієнтацією.

Дескриптор особливості зображення складається з чотирьох величин:

$\sum_i^w dX_i$, $\sum_i^w |dX_i|$, $\sum_i^w dY_i$, $\sum_i^w |dY_i|$, які є сумарними градієнтами за кожним з 16 квадрантів. Для всіх 16 квадрантів зображення отримуємо 64 компонента дескриптора. На додаток до дескриптора для опису точки обчислюється величина:

$$sign\left(S_{xx} + S_{yy}\right), \tag{1.6}$$

яка дозволяє розрізняти темні і білі плями.

Метод опорних векторів. Даний метод спочатку відноситься до бінарних класифікаторів, хоча існують способи змусити його працювати і для задач мультикласифікації. Ідея методу зручно проілюстровано на наступному простому прикладі: дані точки на площині, розбиті на два класи (рисунок 1.5). Проведемо лінію, що розділяє ці два класи (червона лінія на рисунку 1.5). Далі, все нові точки (не з навчальної вибірки) автоматично класифікуються наступним чином:

– точка яка вище прямої потрапляє в клас A;

– точка яка нижче прямої – в клас B.

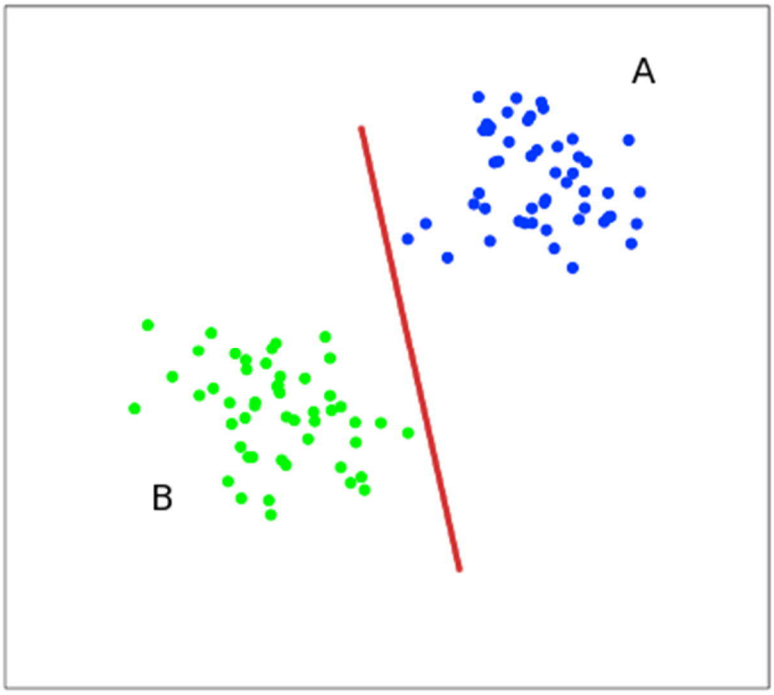

Рисунок 1.5 – Приклад методу опорних векторів

Таку пряму назвемо розділовою прямою. Однак, в просторах високих розмірностей пряма вже не буде розділяти наші класи, так як поняття «нижче прямої» або «вище прямої» втрачає будь-який сенс. Тому замість прямих необхідно розглядати гіперплощини – простори, розмірність яких на одиницю менше, ніж розмірність початкового простору. В $Ў^3$ наприклад, гіперплощина – це звичайна двовимірна площина. У даному прикладі існує кілька прямих, які поділяють два класи (рисунок 1.6).

З точки зору точності класифікації найкраще вибрати пряму, відстань від якої до кожного класу максимально велика. Іншими словами, виберемо ту пряму, яка розділяє класи найкращим чином (червона пряма на рисунку 1.6). Така пряма, а в загальному випадку – гіперплощина, називається оптимальною розділовою гіперплощиною.

Вектори, що лежать ближче всіх до розділової гіперплощини, називаються опорними векторами (support vectors). На рисунку 1.6 вони позначені червоним.

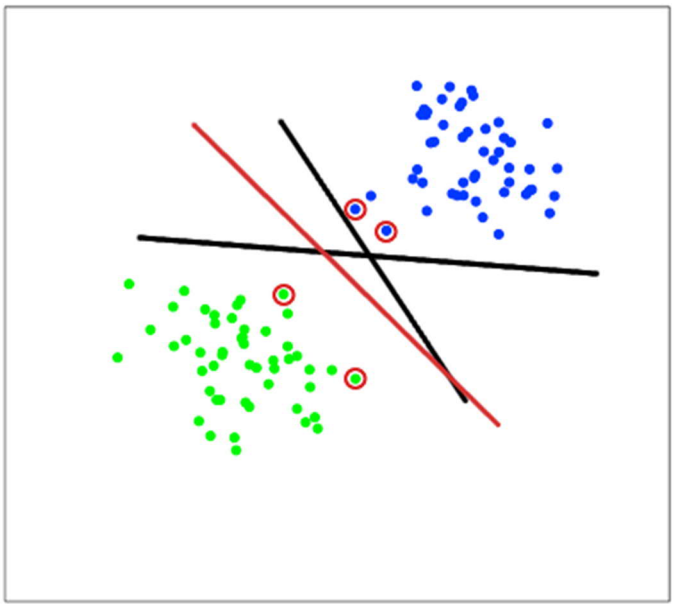

Рисунок 1.6 – Приклад методу опорних векторів

Нехай є вибірка:

$$(x_1, y_1), \ldots, (x_m, y_m), \in \text{Ў}^n, y_i \in \{-1, 1\}. \quad (1.7)$$

Метод опорних векторів будує функцію F, що класифікується, у вигляді:

$$F(x) = sign(\langle w, x \rangle + b), \quad (1.8)$$

де $\langle , \rangle$ – скалярний добуток,

w – нормальний вектор до гіперплощини що розділяє,

b – допоміжний параметр.

Ті об'єкти, для яких $F(x) = 1$ потрапляють в один клас, а об'єкти з $F(x) = -1$ – в інший. Вибір саме такої функції не випадковий: будь-яка гіперплощина може бути задана у вигляді $\langle w, x \rangle + b = 0$ для деяких w і b.

Далі, ми хочемо вибрати такі w і b які максимізують відстань до кожного класу. Можна підрахувати, що дана відстань дорівнює $\frac{1}{\|w\|}$ (рисунок 1.7).

Проблема знаходження максимуму $\frac{1}{\|w\|}$ еквівалентна проблемі знаходження мінімуму $\|w\|^2$.

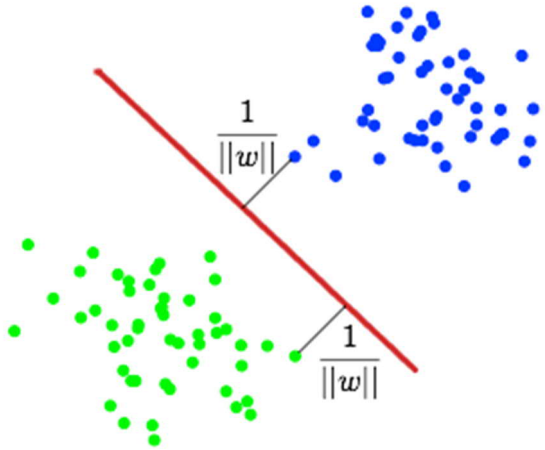

Рисунок 1.7 – Визначення відстані до класу

Формула у вигляді завдання оптимізації:

$$\begin{cases} \arg\min\limits_{w,b} \|\omega\|^2, \\ y_i\left(\langle w, x_i\rangle + b\right) \geq 1, i = 1,\ldots,m \end{cases}, \tag{1.9}$$

яка є стандартною задачею квадратичного програмування і вирішується за допомогою множників Лагранжа [8].

Метод Random forest. По суті своїй модель випадкового лісу побудована на ідеї усереднення великої кількості відповідей [9]. Нехай на вході є певна кількість об'єктів, представимо їх у вигляді таблиці (рисунок 1.8).

Стовпці в таблиці даних це параметри, а рядки – об'єкти навчальної вибірки. Для побудови дерева потрібно виконати наступні кроки:

1) випадковим чином вибрати декілька стовпців (деяке число параметрів $\approx \sqrt{m}$) – рисунок 1.9;

	x_1	x_2	...	x_m
1				
2				
...				
N				

Рисунок 1.8 – Таблиця об’єктів

	x_1	x_2	...	x_t	...	x_m
1						
2						
...						
N						

Рисунок 1.9 – Таблиця об’єктів, вибір стовпців

2) побудувати під-вибірку з повтореннями розміру N з числа об'єктів навчальної вибірки. Може вийти що деякі об'єкти вибірки не потраплять в під-вибірку, і таких в середньому $N\left(1-\frac{1}{N}\right)^N = \frac{N}{e}$ об'єктів;

3) побудувати T_i бінарні дерева (дерева рішень) до кінця, без відсікання, за допомогою якого-небудь алгоритму побудови вирішальних дерев. Це може бути критерій Джіні, критерій приросту інформації та інші (рисунок 1.10).

Повторюючи процедуру 1-3 можна побудувати велику кількість дерев. Оптимальне число дерев підбирається таким чином, щоб мінімізувати помилку класифікатора на тестовій вибірці. Кінцевим кроком буде побудова ансамблю з вирішальних дерев. Ансамбль в даному випадку – це комітет, який проводить просте голосування. Це означає, що перемагає той клас, який отримав найбільшу кількість голосів. Недоліком цього методу для розпізнавання зображень є достатня складність отримання навчальної вибірки. Потрібна наявність великої

кількості зображень, де гарантовано присутній об'єкт який розпізнається і також потрібна величезна кількість зображень де гарантовано відсутні об'єкти що розпізнаються.

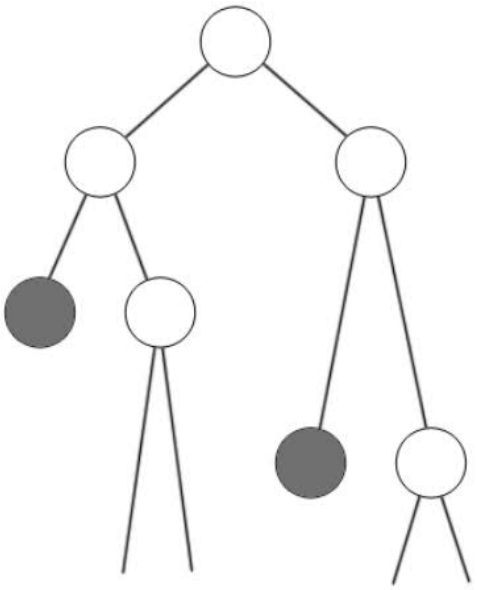

Рисунок 1.10 – Бінарне дерево прийнятих рішень

Каскад класифікаторів. Алгоритмічна композиція, адаптивний гібридний алгоритм – спосіб побудови «сильного» класифікатора, на основі «слабких» [10]. Під терміном сила розуміється якість роботи класифікатора в рішенні поставленого завдання класифікації.

Потрібно побудувати функцію що буде класифікувати – $F : X \to Y$, де X – простір векторів ознак, Y – простір міток класів. Нехай існує навчальна вибірка $(x_1, y_1), \text{K}, (x_N, y_N)$, де $x_i \in X$, а $y_i \in X$. Для простоти будемо вважати що $Y = \{-1,1\}$. Нехай також існує набір алгоритмів що класифікують, які будуть використовуватися для побудови композиції.

Їх будемо називати базовим набором. $H : X \to Y$ – базовий набір. Побудуємо гібридний класифікатор в наступнім вигляді:

$$F(x) = sign\left[\sum_{m=0}^{M} a_m h_m(x)\right], \tag{1.10}$$

де $a_m \in R$ – вага, $h_m \in H$ – алгоритм з базового набору. Будується процес, при якому на кожному кроці додається доданок $f_m = a_m h_m(x)$, який обчислюється з урахуванням роботи вже побудованої частини композиції, наступним чином:

1) початковий розподіл $D_0(i) = \frac{1}{N}$;

2) для кожного $m = 1, 2, ..., M$:

а) вибирається найкращий на даному розподілі $D_m(i)$ класифікатор, що входить до складу базового набору $h_m(x) \in H$:

$$h_m = \arg\min\left(e_m = \sum_{i=1}^{N} D_m(i) \cdot h_m(x_m) \neq y_i\right); \qquad (1.11)$$

b) обчислюється коефіцієнт $a_m = \frac{1}{2}\log\left(\frac{1-e_m}{e_m}\right)$;

c) запам’ятовується $f_m = a_m h_m(x)$ і перераховується розподіл:

$$D_{m+1}(i) = \frac{D_m(i) e^{-y_i f_m(x_i)}}{Z_i}, \qquad (1.12)$$

де Z_i – коефіцієнт нормалізації, такий що $\sum_{i=1}^{N} D_{m+1}(i) = 1$.

3) складається гібридний класифікатор:

$$F(x) = sign\left[\sum_{m=0}^{M} f_m(x)\right]. \qquad (1.13)$$

Вага кожного елемента навчальної вибірки на даному кроці задає важливість даного прикладу для чергового кроку навчання алгоритму. Чим більша вага, тим більше алгоритм буде старатися класифікувати даний приклад на даному кроці правильно. Чим краще приклад розпізнається попередніми

кроками, тим його вага менша – таким чином, найбільші ваги получать приклади які попередніми кроками були класифіковані невірно. Інакше кажучи, підраховуються ваги таким чином, щоб класифікатор, включений до комітету на даному кроці, звертав увагу на ті приклади, з яким попередні кроки не впоралися. Таким чином на кожному кроці ми працюємо з певною частиною даних, які погано класифікувалися попередніми кроками, в результаті об'єднуємо проміжні результати.

1.3 Огляд існуючих розробок

Існуючі рішення можна умовно розбити на дві категорії: впроваджені в комерційне виробництво системи розпізнавання дорожніх знаків і науково-дослідні роботи.

Системи розпізнавання, що розробляються і впроваджуються різними автовиробниками. Приклад такої системи – «OpelEye», яка спрямована на розпізнавання знаків обмеження швидкості. Так як, подібні системи є закритими комерційними проектами, складно отримати достовірні дані про результати роботи і надійності цих систем.

Науково-дослідні роботи. В даному розділі розглянута робота «Integrated Speed Limit Detection And Recognition from Real-Time Video» (Детектування і розпізнавання на відео потоці знаків обмеження швидкості в режимі реального часу), виконана Marcin L. Eichner і Toby P. Breckon в 2008 р [11]. Підхід в їхній роботі являє собою модель алгоритму, що дозволяє розпізнати як знаки, що вводять обмеження на максимальну швидкість, так і знаки що знімають обмеження, крім того, відслідковується поворот автомобіля на іншу дорогу, що згідно ПДР, означає скасування введеного раніше обмеження. Розглянемо лише частину розпізнавання знаків (обмеження швидкості).

Етапи алгоритму:

– етап виявлення. Використовуючи кольоровий простір YCrCb, з вихідного зображення витягується канал Cr, потім отримане півтонування адаптивно бінарізується, що забезпечує стійкість до перепадів яскравості в межах знака. Освічені на бінарному зображенні пов'язані групи пікселів використовуються для детектування кіл на основі RANSAC-технології (метод оцінки параметрів моделі на основі випадкових вибірок), яка дозволяє визначити, чи відповідає пов'язаний компонент заданої моделі геометричній фігурі, використовуючи для перевірки лише частину точок компонента. Таким чином, сильне зашумлення зображення або часткове перекриття знаку не перешкоджає його виявленню;

– етап розпізнавання. З вихідного зображення, перетвореного в градації сірого, витягується фрагмент, що обмежується облямівкою, виконується його бінаризація з граничним значенням, рівним середньому значенню пікселів вхідного зображення. Даний підхід дає хороше виявлення темних цифр на світлому фоні знаку. Далі отримана картинка масштабується до розміру 20 на 20 пікселів і подається на вхід нейронної мережі. Для класифікації використовується класична багатошарова мережа з прямим розповсюдженням сигналів. Структура мережі включає 400 у вхідному шарі (тому що розмір нормалізованого зображення 20x20), 30 нейронів в прихованому і 12 у вихідному шарі. Кількість нейронів у прихованому шарі підбиралося досвідченим шляхом. Перші десять виходів відповідають значенням максимальної швидкості: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. 11 вихід приписаний знаку скасування обмежень швидкості, 12 використовується для оповіщення про те, що об'єкт на зображенні це не знак. Сигнали на виходах мережі приймають значення від 0 до 1 і вказують на ймовірність відповідності вхідного об'єкта відповідного класу. Класифікація вважається успішною, якщо різниця між двома найбільшими вихідними значеннями більше 0.5, в іншому випадку результат ігнорується. Продуктивність даного методу дозволяє обробляти до 27 кадрів в секунду при використанні одноядерного процесора з тактовою частотою 1.6 GHz.

При випробуваннях в умовах міського руху системою було виконано виявлення 92% зустрінутих знаків, відсоток помилкової класифікації склав 0.7%. Розглянутий в розділі підхід має достатню стійкість до шумів і часткового перекриття знаків сторонніми об'єктами, при цьому демонструючи прийнятну продуктивність. Однак він має свої недоліки: не забезпечує інваріантності повороту знаків в площині зображення; не забезпечує стійкість до змін ракурсу; алгоритм розпізнає тільки знаки з формою кола.

1.4 Технології розробки додатків для мобільних пристроїв

Android – операційна система для смартфонів, планшетів, електронних книг, цифрових програвачів, наручних годинників, фітнес-браслетів, ігрових приставок, ноутбуків, нетбуків, смартбуків, окулярів Google Glass, телевізорів, проекторів і інших пристроїв (в 2015 році з'явилася підтримка автомобільних систем і побутових роботів).

Створена на ядрі Linux і власній реалізації віртуальної машини Java від Google. Спочатку розроблялася компанією Android, Inc., яку потім купила Google. Згодом Google ініціювала створення альянсу Open Handset Alliance (OHA), який зараз займається підтримкою і подальшим розвитком платформи. Android дозволяє створювати Java додатка, що керують пристроєм через розроблені Google бібліотеки. Android Native Development Kit дозволяє перенести бібліотеки і компоненти додатків, написані на C та інших мовах.

Основні переваги розробки додатку на базі Android:

– платформа від Google є найбільш відкритою і розвинутою, орієнтованою на кінцевого користувача;

– компанія надає величезний вибір пристроїв за різними цінами, охоплюючи ринок практично повністю.

У 86% смартфонів, проданих в усьому світі у другому кварталі 2014 року, була встановлена операційна система Android. На конференції розробників травнем 2017 року Google оголосила, що за всю історію Android було активовано більше 2 млрд Android пристроїв.

Для розробки додатків існує безкоштовна Android Studio платформа. Це інтегроване середовище розробки (IDE) для роботи з платформою Android, анонсована 16 травня 2013 року на конференції Google I/O. Перша стабільна версія 1.0 була випущена в грудні 2014 року, тоді ж припинилася підтримка плагіна Android Development Tools (ADT) для Eclipse. Android Studio розроблена на програмному забезпеченні IntelliJ IDEA від компанії JetBrains, – офіційний засіб розробки Android додатків. Дане середовище розробки доступне для Windows, macOS і Linux.

Для створення додатків під Android використовується мова програмування Java. Програми на Java транслюються в байт-код, що виконується віртуальною машиною Java (JVM) – програмою, що обробляє байтовий код і передає інструкції обладнанню як інтерпретатор.

Переваги подібного способу виконання програм – в повній незалежності байт-коду від операційної системи і устаткування, що дозволяє виконувати Java додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером) викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять те, що виконання байт-коду віртуальною машиною може знижувати продуктивність програм і алгоритмів, реалізованих на мові Java. Останнім часом було внесено ряд удосконалень, які дещо збільшили швидкість виконання програм на Java:

– застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді;

– широке використання переносних орієнтованого коду (native-код) в стандартних бібліотеках;

– апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM).

За даними сайту shootout.alioth.debian.org, для семи різних завдань час виконання на Java становить в середньому в півтора-два рази більше, ніж для C/C++, в деяких випадках Java швидше, а в окремих випадках в 7 разів повільніше. З іншого боку, для більшості з них споживання пам'яті Java-машиною було в 10-30 разів більше, ніж програмою на C/C++. Також цікаве дослідження, проведене компанією Google, згідно з яким відзначається істотно нижча продуктивність і більше споживання пам'яті в тестових прикладах на Java в порівнянні з аналогічними програмами на C ++.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли також вираз в специфікації загальномовна інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

Основні можливості:

– автоматичне керування пам'яттю;

– розширені можливості обробки виняткових ситуацій;

– багатий набір засобів фільтрації введення / виведення;

– набір стандартних колекцій, таких як масив, список, стек;

– наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);

– наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;

– вбудовані в мову засоби створення багатопоточних додатків;

– уніфікований доступ до баз даних;

– на рівні окремих SQL-запитів – на основі JDBC, SQLJ;

– на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних – на основі Java Data Objects і Java Persistence API;

– підтримка шаблонів (починаючи з версії 1.5);

– паралельне виконання програм.

OpenCV (Open Source Computer Vision Library) являє собою бібліотеку програмного забезпечення для комп'ютерного зору та комп'ютерного навчання з відкритим вихідним кодом. OpenCV створена для забезпечення загальної інфраструктури додатків для комп'ютерного зору і прискорення використання сприйняття машини в комерційних продуктах. Будучи ліцензованим BSD продуктом, OpenCV спрощує бізнес для використання і модифікації коду [12].

Основною метою для розробки даної бібліотеки було підвищення ефективності в додатках реального часу. Функціонал бібліотеки реалізований на мові C. OpenCV має можливість використовувати багатоядерні процесори. При необхідності автоматичної оптимізації на платформах Intel, можливо додаткове придбання та інтеграція з бібліотекою IPP.

До складу бібліотеки входить більше 2500 оптимізованих алгоритмів, які включають в себе повний набір класичних і сучасних алгоритмів комп'ютерного зору і машинного навчання. Дані алгоритми можуть використовуватися для виявлення і розпізнавання осіб, ідентифікації об'єктів, класифікації дій людини на відео, відстеження рухомих об'єктів і рухів самої камери і т.д. OpenCV має більше 47 тисяч користувачів спільноти, а передбачувана кількість завантажень перевищує 14 мільйонів. Бібліотека широко використовується в компаніях, дослідницьких групах і в урядових органах.

Крім таких відомих компаній як Google, Yahoo, Microsoft, Intel, IBM, є безліч різноманітних стартапів, таких як Applied Minds, VideoSurf і Zeitera, які так само використовують функціонал OpenCV.

Бібліотека має інтерфейси для C ++, Python, Java і MATLAB і підтримує Windows, Linux, Android і Mac OS. OpenCV орієнтується в основному на додатки в режимі реального часу і використовує переваги команд MMX і SSE, коли вони доступні. В даний час активно розвиваються повнофункціональні інтерфейси CUDA і OpenCL.

1.5 Висновки до розділу

У даному розділі сформовано основні вимоги до системи розпізнавання, розглянуті найбільш популярні методи розпізнавання, такі як: метод нейронних мереж; метод SURF; метод опорних векторів; метод Random forest та каскад класифікаторів. Розглянуто їх недоліки, відповідність поставленим вимогам. Розглянуті технології розробки для мобільних пристроїв.

2 РОЗРОБЛЕННЯ АЛГОРИТМІВ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

2.1 Узагальнений алгоритм аналізу відеопотоку

Нехай X – множина зображень, Y – множина непересічних класів (класів дорожніх знаків), f^* – цільова функція, що відображає множину X на множину Y, $f^*: X \to Y$. Значення цільової функції y^* відомі тільки на кінцевій множині пар прецедентів (x_i, y_i) – навчальній вибірці. Запис $f^*(x_k) = y_k$ означатиме, що на зображенні x_k присутній дорожній знак y_k. Необхідно відновити функціональну залежність між зображеннями і відповідями, побудувати алгоритм $A: X \to Y$, що володіє наступними властивостями:

– відображення A повинно допускати фіктивну чисельну реалізацію;

– на навчальній вибірці A повинен видавати правильні відповіді;

– A повинен володіти узагальнюючої здатністю, тобто наближати цільову функцію не тільки на об'єктах навчальної вибірки, а й на всій множині X.

Розглянуті в попередньому розділі методики розпізнавання не відповідають встановленим вимогам, тому потрібно розробити алгоритм, який буде вирішувати поставлену задачу розпізнавання, та задовольняти дані вимоги.

Алгоритм розпізнавання складається із наступних етапів:

1) покадрове зчитування зображення з відеопотоку;

2) сегментація знаків – виявлення місця розташування знаку на зображенні з метою подальшого його розпізнавання нейронною мережею;

3) попередня обробка знайденого знаку;

4) розпізнавання знаку – використання навченої згорткової нейронної мережі для розпізнавання дорожніх знаків.

На рисунку 2.1 зображено блок схему алгоритму розпізнавання дорожнього знаку.

Так як нейронна мережа приймає на вхід зображення розмірністю 32x32, то потрібно витягти з відеопотоку кадр, і сегментувати його таким чином, щоб

дорожній знак максимально повно займав зображення такої розмірності. Іншими словами потрібно знайти прямокутник, в який вписується дорожній знак. Ця процедура називається детектуванням (рисунок 2.2). Знайдений знак при необхідності масштабується, сегментується і подається на вхід нейронної мережі.

Рисунок 2.1 – Блок-схема алгоритму

Вхідний відеопотік є послідовністю RGB зображень. RGB зображення, як відомо, містить в собі інформацію про інтенсивності червоного, зеленого і синього кольору відповідно. Кожен канал RGB зображення має глибину 8 біт, тобто інтенсивність пікселя може мати значення від 0 до 255. Для того щоб алгоритм працював в режимі реального часу доцільно працювати не з усіма трьома каналами зображення, а обмежитися лише одним.

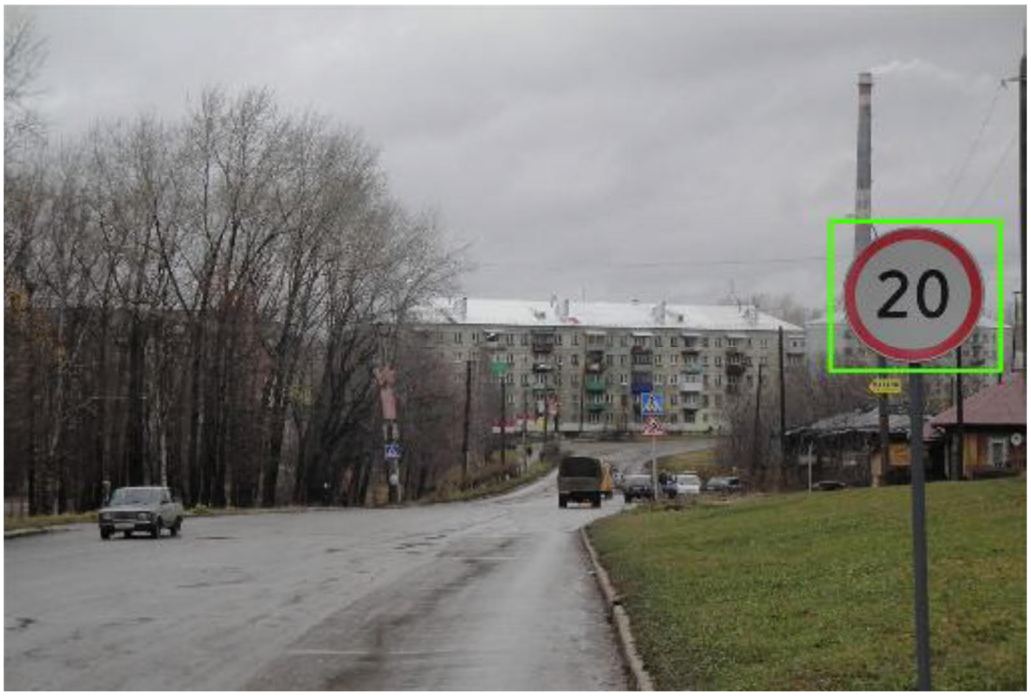

Рисунок 2.2 – Детектування об'єкту

Так як алгоритм сегментації базується на пошуку меж дорожнього знаку, потрібно, щоб на початковому зображенні знак виділявся на своєму фоні. Виходячи з цих тверджень, використовувати будь-який з каналів RGB зображення буде не ефективним з кількох причин: по-перше, різні відеокамери мають різну кольорову передачу і, якщо передача вибраного каналу слабка, то ефективність знизиться; по-друге, колір меж дорожніх знаків різний, а це означає, що визначення знаку одного кольору на зображенні, що відображає інший канал, може бути складним. Рішенням даної проблеми стало використання каналу *Y* колірного простору YCrCb, Який на практиці найкраще підходить для вилучення меж об'єктів.

Локалізація дорожніх знаків. Щоб розпізнавати дорожні знаки, насамперед необхідно вміти їх локалізувати. Оскільки для досягнення цієї мети був обраний метод Віоли-Джонса, то для цього потрібно сформувати базу для навчання і навчити класифікатор.

У вільному доступі немає бази українських дорожніх знаків, яку можна використовувати для навчання класифікатора. Однак існують бази інших країн,

які використовують для цих цілей. Тому було вирішено взяти за основу базу німецьких знаків GTSRB [13], так як в ній більшість знаків зовні ідентичні російським. База містить знаки 43 видів і близько 50000 їх реальних зображень. Для використання в рамках роботи база була відредагована.

З бази були видалені знаки, відсутні на українських дорогах, і знаки, розпізнавання яких в рамках даної роботи розглянута не буде (рисунок 2.3).

Рисунок 2.3 – Приклад знаків які видалено з бази

Додані деякі відсутні знаки (3 види): пішохідний перехід, зупинка заборонена, стоянка заборонена. Було взято по 12 реальних зображень кожного знаку як шаблон, після чого до ним були застосовані наступні перетворення (рисунок 2.4):

– спотворення: шаблони оберталися під випадковим кутом;

– яскравість: шаблони затемнювалися або освітлювалися на випадкову величину [14];

– розпливчастість: застосовувався "motion" фільтр [15] зі випадковими параметрами для створення ефекту, що зображення розмилося при зйомці;

– шум: до шаблонів застосовувався випадковий гаусів шум [16].

У підсумку, було створено по 900 штучних примірників кожного дорожнього знаку.

Рисунок 2.4 – Послідовність перетворень при формуванні бази знаків

Разом, з бази GTSRB було видалено 20 класів дорожніх знаків і додано 3 нових класу. У новій базі вийшло близько 35000 зображень 26 класів дорожніх знаків.

Навчання каскаду класифікаторів. Для збільшення точності локалізації дорожніх знаків було вирішено розділити їх на 2 класу: трикутні (попереджувальні і "Пішохідний перехід "), круглі (забороняють), і навчити окремі каскади класифікаторів для кожного класу.

Для навчання кожного каскаду була сформована своя позитивна і негативна вибірка. Позитивна вибірка містила тільки зображення зі знаком. Негативна вибірка містила зображення без знаків: зображення вулиць, природи, неба і т.д. від різноманітності негативної вибірки залежить якість роботи каскаду в різних умовах. У підсумку, розмір кожної позитивної вибірки склав 1000 зображень, а негативною – 1500 зображень.

Для навчання каскаду класифікаторів в пакеті OpenCV є програма "opencv_traincascade.exe" яка на вхід приймає такі дані:

– data – адреса папки, куди класти отримані результати;

– numStages 18 – кількість рівнів каскаду, які програма буде навчати. Чим більше рівнів, тим точніше, але довше він працює. Нормальне їх кількість від 16 до 25;

– minhitrate 0,999 – коефіцієнт, що визначає якість навчання. По суті (1-0,999) – кількість пропущених об'єктів на шарі;

– numPos 1000 – кількість позитивних зразків;

– numNeg 1500 – кількість негативних зразків.

Після навчання каскадів була отримана наступна точність локалізації (тестування проводилося на тестовій базі GTSDB) при коефіцієнті збільшення скануючого вікна рівного 1,1:

– трикутні знаки – 88,5%;

– круглі знаки – 84%.

Дана точність влаштовує, тому що класифікатори працюють з даними з камери, і якщо об'єкт не був знайдений на одному кадрі, він може бути знайдений на наступному. Було визначено, що більшість помилок пов'язано з маленьким розміром дорожнього знаку (менше 30x30). Приклад успішної локалізації представлений на рисунку 2.5.

Рисунок 2.5 – Локалізований знак

Далі, зображення масштабується до розміру 32x32 і подається на вхід нейронної мережі. Алгоритм попередньої обробки зображення і алгоритм сегментації реалізовані на мові Java з використанням бібліотеки з відкритим вихідним кодом для комп'ютерного зору – OpenCV.

Попередня обробка зображення. Щоб істотно зменшити подальші обчислення з зображення витягується область інтересів – частина зображення з найбільш імовірною появою знаків. Для України це права частина дорожньої смуги, при цьому дорожній символ не розташовується на зображенні занадто високо або занадто низько (рисунок 2.6).

Таким чином, якщо вхідне зображення з роздільною здатністю 1920 × 1080, а розмір області інтересів 625 × 400, то обчислення скорочуються приблизно в 8 разів. Перетворення зображення RGB в зображення YCrCb [17].

Рисунок 2.6 – Область інтересів

Перетворення колірного простору відбувається наступним чином:

$$Y = K_r \cdot R + (1 - K_r - K_b) \cdot G + K_b \cdot B, \tag{2.1}$$

де R, G, B – канали RGB зображення, K_r, K_b – визначені коефіцієнти, отримані вченими на основі колірного сприйняття людського ока. $K_r = 0.2126$, $K_b = 0.0722$,

$$C_r = \frac{1}{2} \cdot \frac{R - Y}{1 - K_r}, \tag{2.2}$$

$$C_b = \frac{1}{2} \cdot \frac{B - Y}{1 - K_b}. \tag{2.3}$$

Витягуючи канал Y, отримуємо одноканальне зображення, тобто зображення в градаціях сірого, яке придатне для ефективного використання в подальших обчисленнях (рисунок 2.7).

Наступним кроком попередньої обробки є підвищення контрастності методом еквалізації гістограм [18].

Рисунок 2.7 – *Y*-канал зображення в YCrCb

Підрахунок гістограми зображення (для кожного значення яскравості порахувати кількість пікселів з даної яскравістю):

$$hist(x) = \left\{ \sum_{i=0}^{w} \sum_{j=0}^{h} I(i, j) : I(i, j) = x \right\}, x \in [0, 255], \tag{2.4}$$

де $I(i, j)$ – значення яскравості i, j-ого пікселя вихідного зображення,

h – висота зображення, w – ширина зображення.

Побудова інтегральної гістограми:

$$\begin{cases} cumulativeHist(x) = hist(x) + cumulativeHist(x-1), x > 0 \\ cumulativeHist(x) = hist(x), x = 0 \end{cases}. \tag{2.5}$$

Піксель нового зображення розраховується за допомогою інтегральної гістограми і старого зображення:

$$New\_Image(i, j) = \frac{cumulativeHist(I(i, j))}{h \cdot w} \cdot 255. \quad (2.6)$$

Гістограма вхідного зображення виглядає наступним чином (рисунок 2.8).

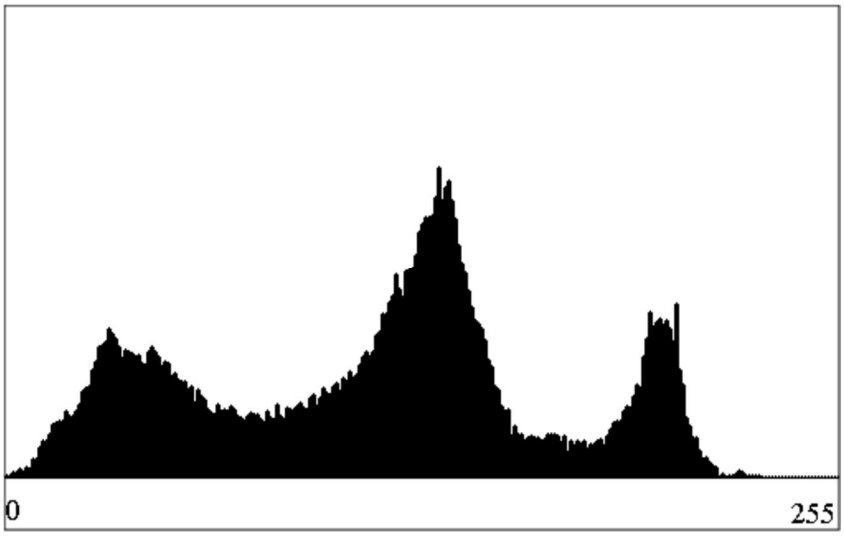

Рисунок 2.8 – Гістограма яскравості

Після процедури виконання еквалізації, гістограма ніби розтягується і використовує всі значення яскравості рівномірно, намагаючись не накопичуватися в одних значеннях.

На рисунку 2.9 зображено еквалізовану гістограму яскравості.

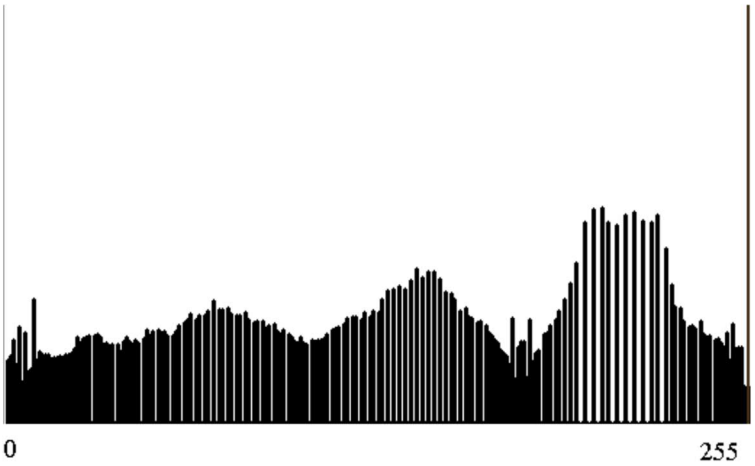

Рисунок 2.9 – Еквалізована гістограма яскравості

На рисунку 2.10 зображено різницю між вихідним зображенням і зображенням з рівною гістограмою.

Рисунок 2.10 – Порівняння зображень до і після нормалізації гістограми

2.2 Топологія та навчання нейронної мережі

Вибір топології нейронної мережі завдання не просте. До цих пір в теорії нейронних мереж не існує чіткого, визначеного алгоритму для вибору структури мережі, а саме: числа прихованих шарів, числа нейронів в прихованих шарах. Звичайно, існують деякі рекомендації по вибору структури мережі, але теоретичного обґрунтування немає. Практично всі подібні рекомендації виявляються експериментальним шляхом.

Таким чином, в якості структури мережі для розпізнавання дорожніх знаків стала структура мережі LeNet, запропонована творцем згорткової нейронної мережі Яном Лекуном в 1989 році (рисунок 2.11).

Основною складністю в роботі з нейронними мережами є підбір параметрів мережі. За основу була взята типова структура для згорткових нейронних мереж з такою послідовністю шарів: вхідний – згортковий – субдискретизующий –

згортковий – субдискретизующий – повнозв’язний – повнозв’язний – повнозв'язний (вихідний).

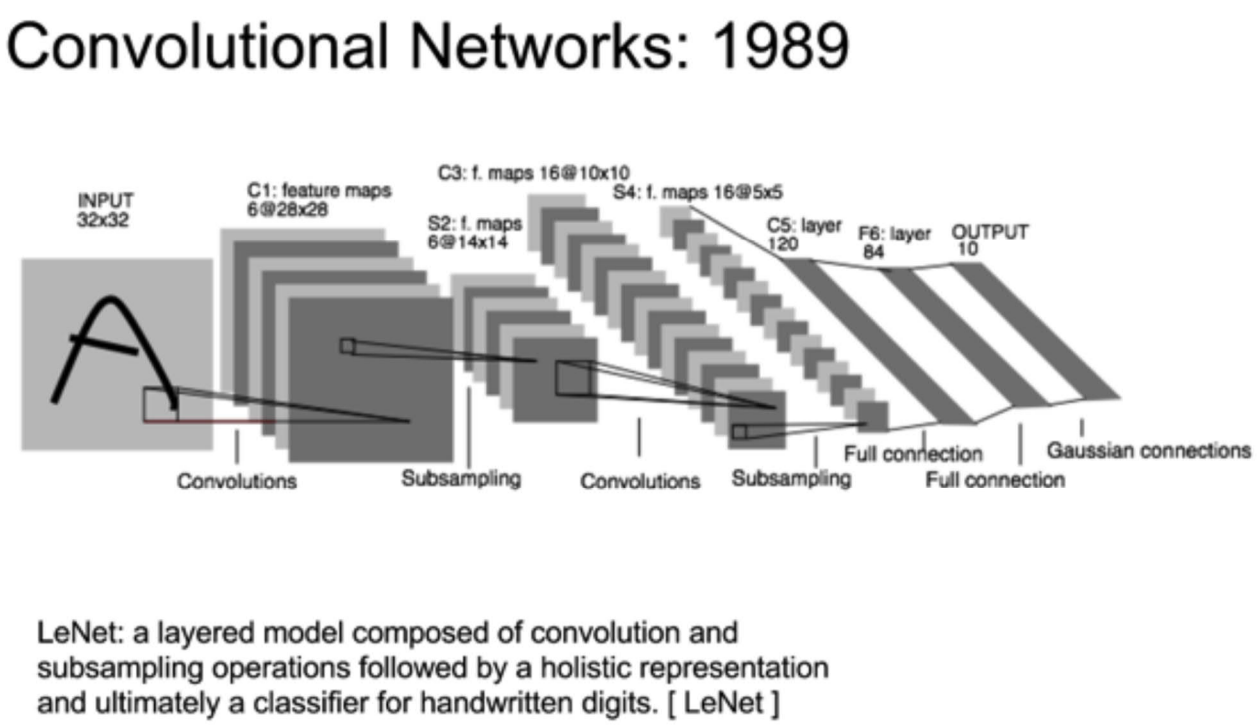

Рисунок 2.11 – Структура мережі LeNet

Кількість карт на шарах підбирались експериментально. Результати експериментів представлені в таблиці 2.1.

Таблиця 2.1 – Результати експериментів

№	Архітектура	Точність	Час проходу (мс)
1	1-16-16-32-32-200-100-26	98,18%	1,86
2	1-12-12-24-24-150-100-26	97,89%	0,9
3	1-8-8-16-16-100-80-26	95,43%	0,45
4	1-6-6-12-12-80-60-26	90,12%	0,29

З розглянутих архітектур була обрана друга, тому що швидкість її роботи вище в 2 рази в порівнянні з першою, а точністю вона практично не поступається. Її детальний опис представлено в таблиці 2.2.

Таблиця 2.2 – Опис архітектури мережі

Шар	Тип	Кількість карт і нейронів	Ядро
0	Вхідний	1 карта 32 × 32 нейрона	
1	Згортковий	12 карт 28 × 28 нейронів	5 × 5
2	Субдискретизующий	12 карт 14 × 14 нейронів	2 × 2
3	Згортковий	24 карти 10 × 10 нейронів	5 × 5
4	Субдискретизующий	24 карти 5 × 5 нейронів	2 × 2
5	Повнозв'язний	150 нейронів	
6	"dropout" шар (0,5)		
7	Повнозв'язний	100 нейронів	
8	"dropout" шар (0,5)		
9	Повнозв'язний	26 нейронів	
10	"softmax" шар		

Помилка згорткових шарів. Нехай E – функція помилки виходів згорткового шару, і нам відомі значення цієї помилки. Потрібно визначити внесок кожного попереднього нейрона в помилку E, тобто $\frac{\partial E}{\partial y_{ij}^t}$ [20]. Для початку визначимо внесок градієнта в кожну вагу:

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^t} \frac{\partial x_{ij}^t}{\partial \omega_{ab}}, \tag{2.17}$$

де $N \times N$ – розмір карти ознак,

$m \times m$ – розмір ядра фільтра (розмір матриці ваг).

В силу того, що $\frac{\partial x_{ij}^t}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{t-1}$ (з рівняння поширення сигналу) отримуємо:

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{t}} y_{(i+a)(j+b)}^{t-1}. \tag{2.18}$$

Тепер обчислимо «дельту» (поправку ваг):

$$\frac{\partial E}{\partial x_{ij}^{t}} = \frac{\partial E}{\partial y_{ij}^{t}} \frac{\partial y_{ij}^{t}}{\partial x_{ij}^{t}} = \frac{\partial E}{\partial y_{ij}^{t}} \frac{\partial}{\partial x_{ij}^{t}} \left(\sigma \left(x_{ij}^{t} \right) \right) = \frac{\partial E}{\partial y_{ij}^{t}} \sigma' \left(x_{ij}^{t} \right). \tag{2.19}$$

Внаслідок того, що помилка шару $\frac{\partial E}{\partial y_{ij}^{t}}$ нам відома, можна знайти значення «дельт» $\frac{\partial E}{\partial y_{ij}^{t}}$. Помилка попереднього шару буде обчислюватися як:

$$\frac{\partial E}{\partial y_{ij}^{t-1}} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{t-1}} \frac{\partial x_{(i-a)(j-b)}^{t-1}}{\partial y_{ij}^{t-1}} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{t}} \omega_{ab}. \tag{2.20}$$

Так як навчання шарів субдискретизації не потрібне, немає сенсу шукати на них помилку.

Помилка повнозв’язних шарів. Знаходження помилки повнозв’язних шарів схоже з знаходженням помилки згорткових шарів. Помилка вихідного шару:

$$\frac{\partial E}{\partial y_{i}^{t}} = \frac{d}{dy_{i}^{t}} E \left(y^{t} \right). \tag{2.21}$$

«Дельти» знайдемо так:

$$\frac{\partial E}{\partial x_{i}^{t}} = \sigma' \left(x_{i}^{t} \right) \frac{\partial E}{\partial y_{i}^{t}}. \tag{2.22}$$

Помилка на попередніх шарах (обчислюється, поки не досягнемо вхідного шару):

$$\frac{\partial E}{\partial y_i^t} = \sum_j \omega_{ij}^t \frac{\partial E}{\partial x_j^{t+1}}. \quad (2.23)$$

Градієнт помилки:

$$\frac{\partial E}{\partial \omega_{ij}^t} = y_i^t \frac{\partial E}{\partial x_j^{t+1}}. \quad (2.24)$$

Навчання мережі проходить коригуванням ваг на значення, яке обчислюється за формулами зазначеним вище.

2.3 Специфіка реалізації для Android

Для написання Android додатку з використанням бібліотеки OpenCV потрібно спочатку завантажити OpenCV SDK та установити його в проект. Для цього переходимо на офіційний сайт та завантажуємо потрібну нам версію SDK (рисунок 2.12).

До архіву OpenCV SDK для Android входять виконавчі файли бібліотек під різні архітектури мікропроцесорів, обгортка на мові програмування Java, яка використовує Java Native Interface (JNI) для виконання функцій з платформо-залежних бібліотек, вихідні коди прикладів програм і файли .apk для установки OpenCV Manager .

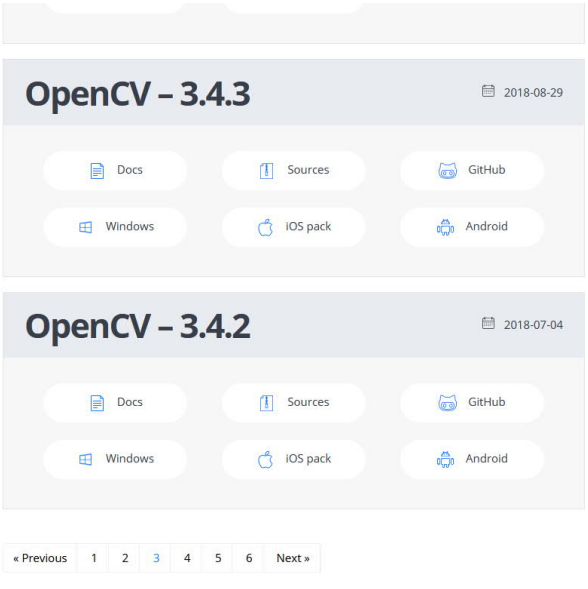

Рисунок 2.12 – Завантаження OpenCV SDK

Далі потрібно додати модуль OpenCV в проект Android Studio. Для початку створюємо проект, або відкриваємо уже створений, до якого потрібно додати модуль. Вибираємо меню File > New > Import Module і вказуємо шлях до \sdk\java (рисунок 2.13).

Після імпорту модуля в його build.gradle потрібно обов'язково оновити параметри compileSdkVersion, buildToolsVersion, minSdkVersion і targetSdkVersion, щоб вони збігалися з відповідними параметрами з модуля програми (рисунок 2.14).

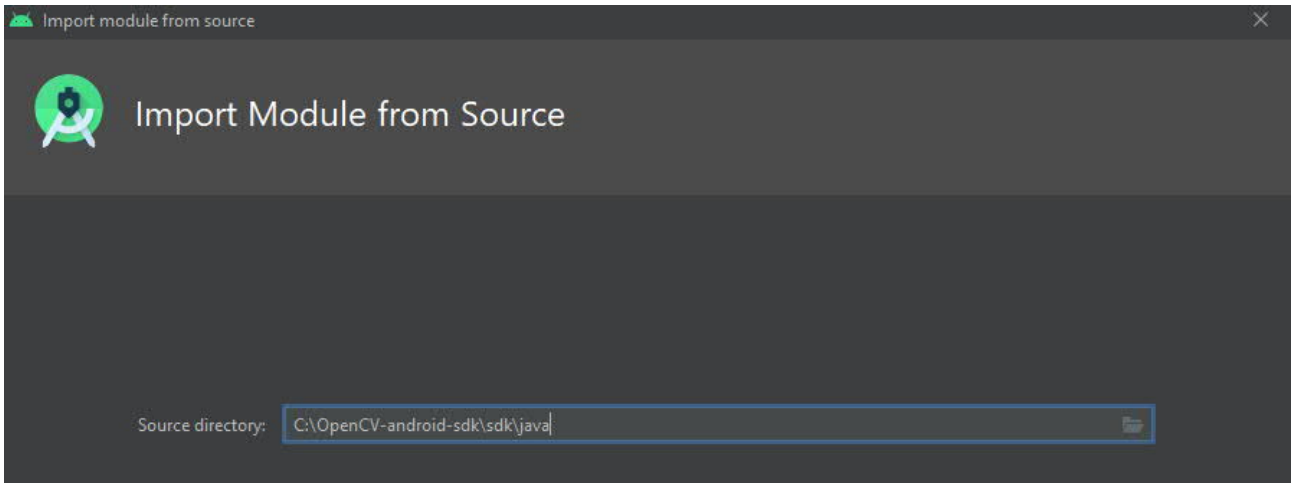

Рисунок 2.13 – Додавання OpenCV модуля в проект

```
android {
    compileSdkVersion 30
    buildToolsVersion "29.0.2"

    defaultConfig {
        minSdkVersion 8
        targetSdkVersion 30
    }
```

Рисунок 2.14 – Параметри build.gradle

Після того, як модуль OpenCV був доданий, його потрібно приєднати як залежності до модуля програми. Для цього потрібно вибрати меню File > Project Structure і для модуля app вказати залежність від opencv.

На рисунку 2.15 зображено вікно Dependencies, де потрібно обрати + > Module dependency > OpenCVLibrary343 > OK.

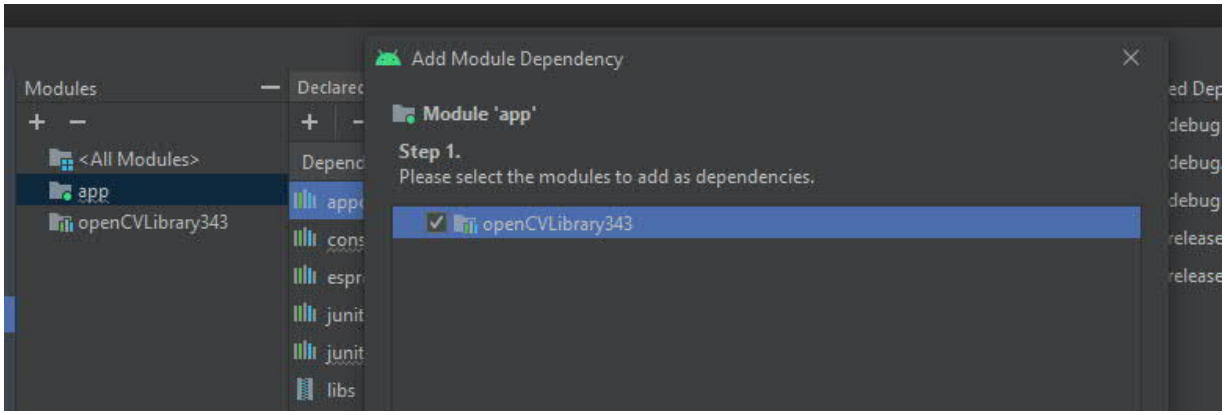

Рисунок 2.15 – Приєднання модуля OpenCV до модуля програми

Останнє, що залишилося зробити – додати виконавчі бібліотеки OpenCV в проект, щоб при складанні вони були упаковані в .apk додаток. Для цього копіюю папку \sdk\native\libs з OpenCV SDK в папку проекту \app\src\main. Перейменовую скопійовану папку libs в jniLibs. Установка OpenCV завершена.

Структура Android включає підтримку різних камер та функцій камери, доступних на пристроях, що дозволяє робити знімки та відео у своїх програмах. Пакет `android.hardware.camera2` надає інтерфейс для окремих пристроїв камер, підключених до Android пристрою, та замінює застарілий клас `Camera`.

Цей пакет моделює пристрій камери як конвеєр, який приймає вхідні запити для зйомки одного кадру, знімає одне зображення на запит, а потім виводить один пакет метаданих результату захоплення, та набір буферів вихідних зображень для запиту. Запити обробляються по порядку, і одночасно можуть виконуватися декілька запитів. Оскільки пристрій камери являє собою конвеєр з декількома етапами, наявність декількох запитів під час виконання необхідне для підтримки повної частоти кадрів на більшості пристроїв Android.

На рисунку 2.16 зображена схема роботи камери [22].

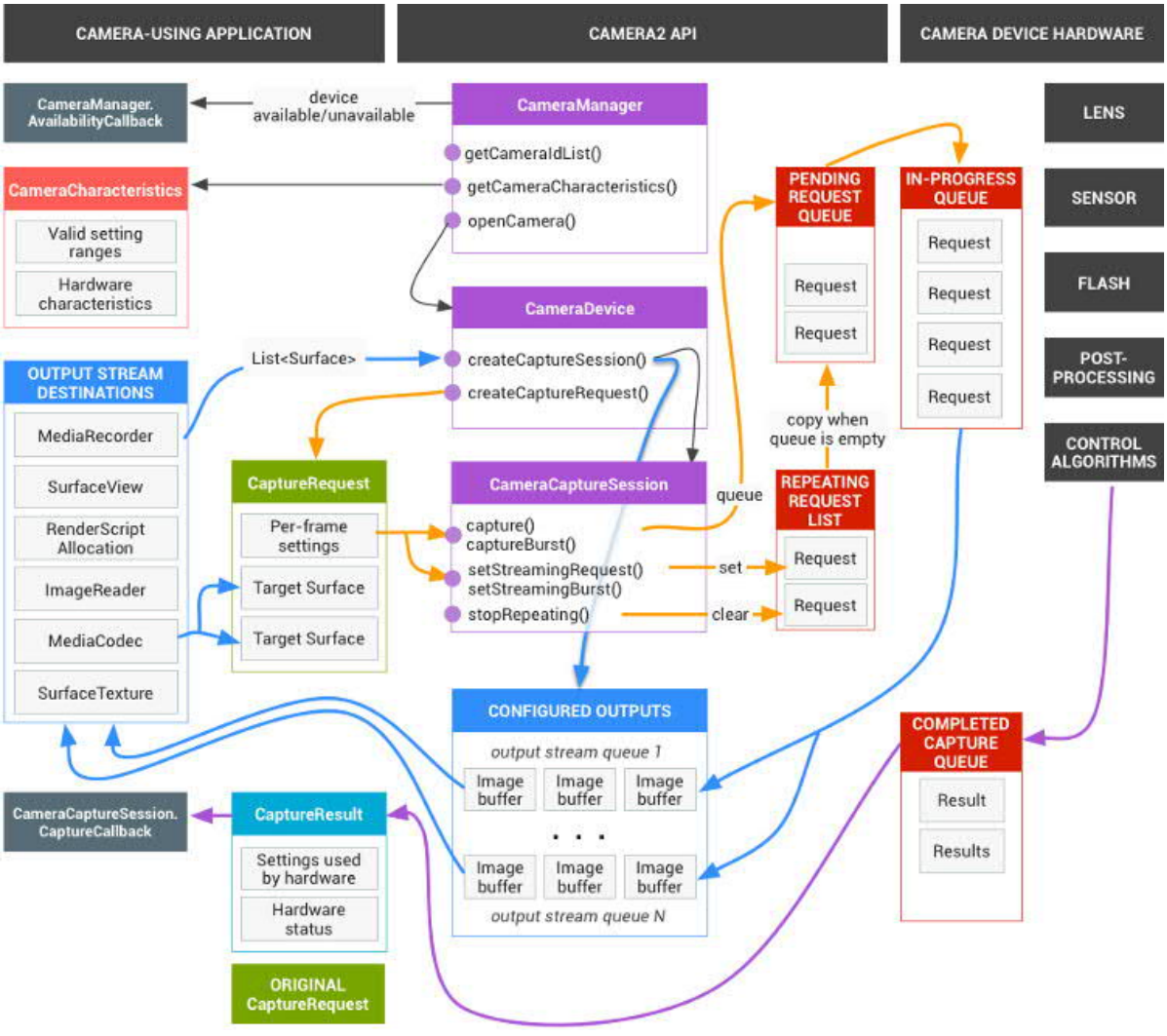

Рисунок 2.16 – Повна схема роботи камери

Щоб перерахувати, запитати та відкрити доступні пристрої камери, викликаємо об’єкт `CameraManager`. Окремі пристрої `CameraDevices` надають

набір інформації про статичні властивості, які описують апаратний пристрій, а також доступні налаштування та вихідні параметри пристрою. Ця інформація надається через об'єкт `CameraCharacteristics` і доступна через `getCameraCharacteristics(String)` [21].

Для запису відео та аудіо використовується Media Recorder, діаграма станів зображена на рисунку 2.17 [23].

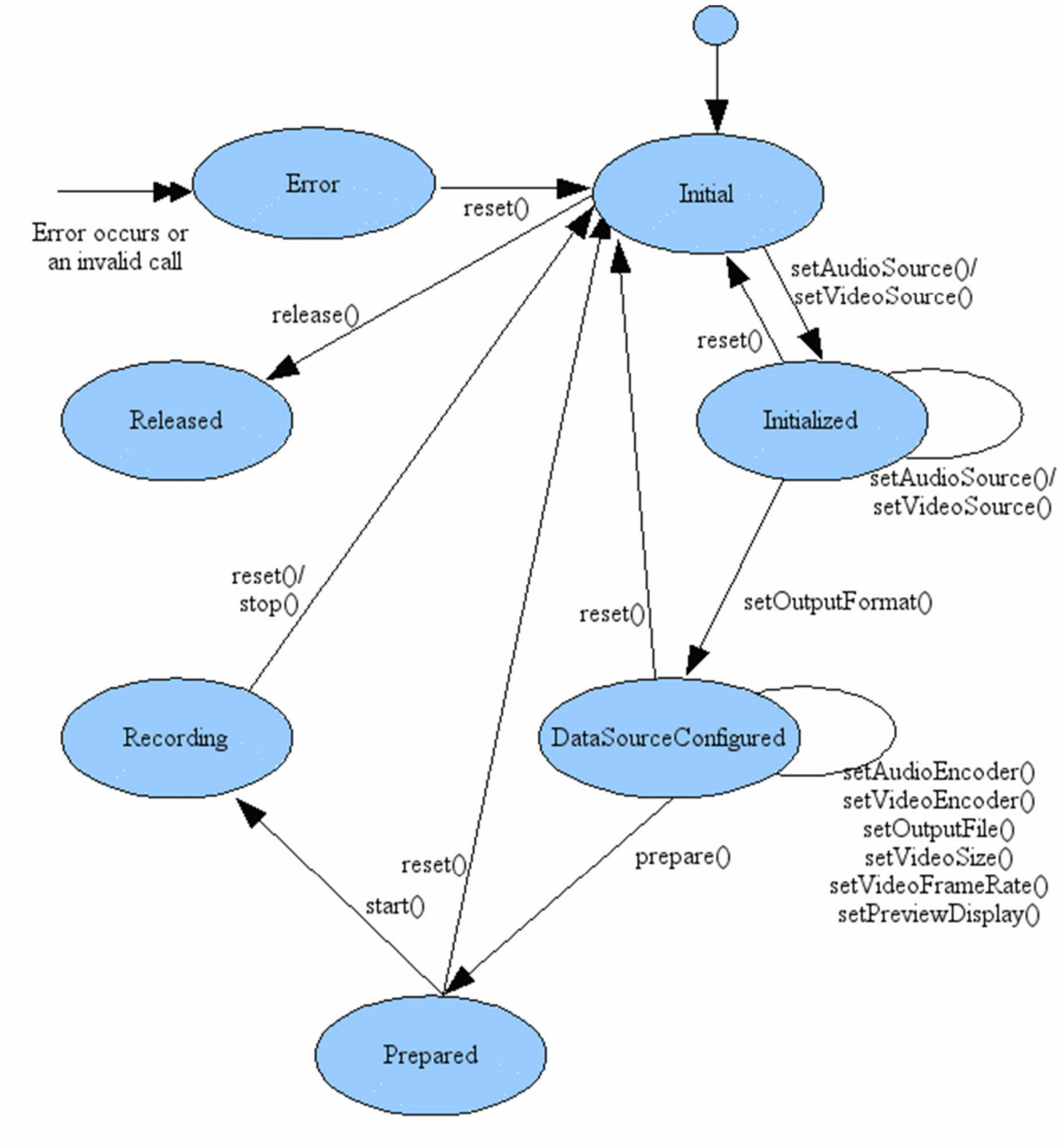

Рисунок 2.17 – Діаграма станів Media Recorder

2.4 Висновки до розділу

В даному розділі розроблено математичний опис алгоритму, його етапи сегментації та розпізнавання. Обрано нейронну мережу, експериментально підібрано кількість карт на шарах та реалізовано на мові Matlab. Здійснено установку OpenCV в проект Android Studio, та розглянуто роботу Android камери.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ

3.1 Структура системи

Класи в OpenCV в Java розділені на наступні пакети:

– org.opencv.imgcodecs – включає клас Imgcodecs, за допомогою якого можна завантажити зображення з файлу або буфера, а також зберегти зображення в файл або в буфер в різних форматах (наприклад, в JPEG);

– org.opencv.core – містить основні класи бібліотеки, реалізують базові структури (вектори, матриці та т. д.). Крім того, пакет включає допоміжні класи (наприклад, класи Point, Rect і ін.). Клас Core з цього пакета містить статичні методи, за допомогою яких можна виконати різні операції з матрицями;

– org.opencv.imgproc – включає класи, призначені для обробки і аналізу зображень;

– org.opencv.features2d – містить класи, за допомогою яких можна знаходити і порівнювати особливі точки;

– org.opencv.photo – включає класи, призначені для створення HDR-зображень;

– org.opencv.video – містить класи, призначені для роботи з відеоданими (аналіз руху і відстеження об'єктів);

– org.opencv.videoio – за допомогою класу VideoCapture з цього пакета можна завантажувати кадри з відеофайлу або послідовності кадрів, а також захоплювати кадри в режимі реального часу з камер зовнішнього відеоспостереження, веб-камер і ін.;

– org.opencv.calib3d – містить класи, за допомогою яких можна виконати калібрування камери, працювати зі стереокамерами і обробляти тривимірні дані;

– org.opencv.objdetect – включає класи для пошуку об'єктів на зображенні. За допомогою навчених класифікаторів можна шукати людей, обличчя, очі, ніс, дізнатися, посміхається людина чи ні, і т. д. При великому бажанні можна

навчити власний класифікатор з довільним призначенням або завантажити вже навчений з Інтернету;

– org.opencv.ml – містить класи, призначені для машинного навчання;

– org.opencv.dnn – включає класи для роботи з нейронними мережами. Можна, можливо завантажувати моделі, навчені в популярних бібліотеках Caffe, TensorFlow і Torch;

– org.opencv.utils – містить допоміжний клас Converters, який в основному використовується для внутрішніх потреб бібліотеки;

– org.opencv.android – включає класи для роботи з ОС Android. Пакет доступний тільки в складі дистрибутива під Android.

Для початку здійснюється запуск класу MainActivity (рисунок 3.1). В даному класі здійснюється ініціалізація кнопок та виклик файлу activity_main.xml.

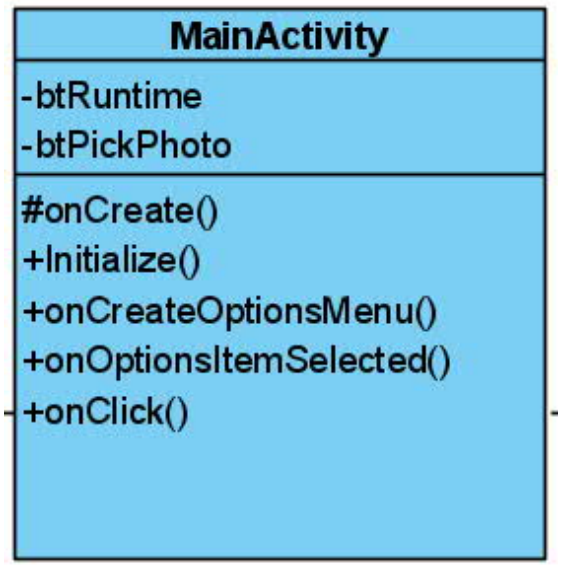

Рисунок 3.1 – Клас MainActivity

Метод onCreate() – задає початкову установку параметрів при ініціалізації активності. При реалізації цього методу необхідно завжди спочатку викликати версію цього методу з суперкласу.

Сім перерахованих методів визначають весь життєвий цикл активності. Є три вкладених цикли, які відстежуються в класі активності:

– повний час життя (entire lifetime) – час з моменту першого виклику методу onCreate() до виклику onDestroy(). Активність робить всю початкову установку свого глобального стану в методі onCreate() і звільняє усі ресурси, які залишилися в onDestroy(). Наприклад, якщо активність породжує додатковий потік, що виконується в фоновому режимі, можна створити цей потік в методі onCreate() і потім зупинити потік в методі onDestroy();

– видимий час життя (visible lifetime) – час між викликом методу onStart() і викликом onStop(). У цей час користувач може бачити вікно активності на екрані, хоча вікно може не бути на передньому плані і може не взаємодіяти з користувачем. Між цими двома методами ви можете підтримувати в коді ресурси, які необхідні, щоб відображати активність користувачеві;

– активний час життя (foreground lifetime) – час між викликами onResume() і onPause(). В цей час вікно активності знаходиться на передньому плані і взаємодіє з користувачем. Активність в процесі роботи програми може часто переходити між станами active і paused, тому код в цих двох методах повинен бути або невеликим за обсягом (щоб не сповільнювати роботу програми під час виконання), або породжувати додаткові потоки, якщо потрібне виконання завдань, які займають тривалий час.

Щоб звернутися до елементу екрану з коду, нам потрібен його ID. Він прописується або в Properties, або в layout-файлах. Для ID існує чіткий формат - @ + id / name, де + означає, що це новий ресурс і він повинен додатися в R.java клас, якщо він там ще не існує

Метод setContentView() – встановлює вміст Activity з layout-файлу. Але як аргумент ми вказуємо не шлях до layout-файлу (res / layout / activity_main.xml), а константу, яка є ID файлу. Ця константа генерується автоматично в файлі R.java. В цьому класі будуть зберігатися згенеровані ID для всіх ресурсів проекту (з папки res / *), щоб ми могли до них звертатися. Імена цих ID-констант збігаються з іменами файлів ресурсів (без розширень).

Файл res / layout / activity_main.xml був створений середовищем розробки разом з Activity. Метод findViewById() – по ID повертає View.

Етап сегментації реалізований у класі Detector (рисунок 3.2). Для сегментації було створено два каскадні класифікатори, один для круглих знаків а інший для трикутних.

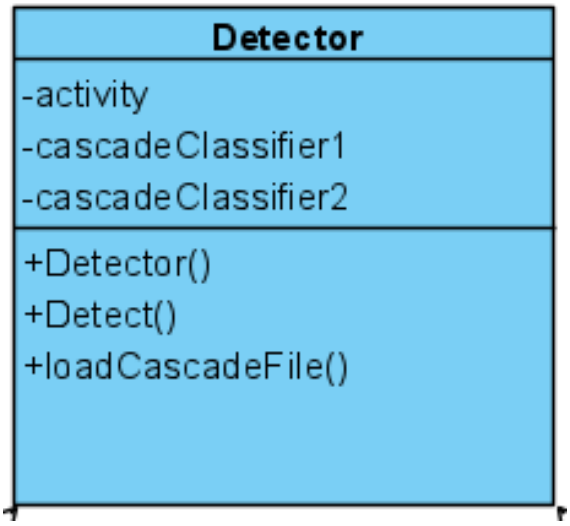

Рисунок 3.2 – Клас Detector

Клас FileOutputStream призначений для запису байтів в файл. Він є похідним від класу OutputStream, тому успадковує всю його функціональність. Через конструктор класу FileOutputStream задається файл, в який проводиться запис. Клас підтримує кілька конструкторів. Файл задається або через строковий шлях, або через об'єкт File. Другий параметр – append задає спосіб запису: якщо він дорівнює true, то дані дозаписуватимуться в кінець файлу, а при false – файл повністю перезаписується.

Для зчитування даних з файлу призначений клас FileInputStream, який є спадкоємцем класу InputStream і тому реалізує всі його методи.

Для створення об'єкта FileInputStream ми можемо використовувати ряд конструкторів. Найбільш використовувана версія конструктора в якості параметра приймає шлях до зчитувати файлу.

Бібліотека комп'ютерного зору OpenCV надає функцію для використання вже навченого каскаду detectMultiScale() зі наступними параметрами:

– const Mat & image – зображення, на якому буде проведено пошук об'єктів;

– vector <Rect> & objects – вектор, для зберігання кордонів знайдених об'єктів;

– double scaleFactor – коефіцієнт збільшення скануючого вікна;

– int minNeighbors – кількість сусідів. Цей коефіцієнт використовується для того, щоб каскад не видавав як відповідь кілька областей які знаходяться поруч (на відстані декількох пікселів);

– Size minSize – мінімальний розмір скануючого вікна;

– Size maxSize – максимальний розмір скануючого вікна.

Так як OpenCV надає інтерфейс для платформи Android, можна скористатися цією функцією для локалізації знака. Оскільки дорожні знаки стоять на правій стороні дороги і розташовані досить високо, то не має сенсу їх шукати на всьому зображенні з камери телефону, можна звузити область до правої верхньої частини зображення. Це дозволило прискорити процес локалізації більш ніж в 2 рази.

Час, необхідний для пошуку дорожніх знаків одним каскадом з використанням смартфона Xiaomi Redmi 3 Pro на зображенні розміру 1280 × 720 при параметрах scaleFactor = 1,1; minSize = 30 × 30; maxSize = 70 × 70:

– на повному зображенні – 530-550 мс;

– в правій верхній частині зображення – 160-170 мс.

Зазначені параметри були обрані в якості значень по замовчуванням. ScaleFactor фактор був підібраний експериментально, як компромісний варіант між швидкістю і точністю роботи. Параметри minSize і maxSize можуть бути обраними користувачем у налаштуваннях програми.

На рисунку 3.3 зображено зону інтересів.

Рисунок 3.3 – Зона пошуку

Процес локалізації може бути описаний наступною послідовністю кроків:

1) отримати зображення з камери телефону;

2) вирізати праву верхню частину (рисунок 3.3);

3) застосувати навчені каскадні класифікатори за допомогою функції `DetectMultiScale()`. Для більш швидкої обробки кадрів і, як наслідок, більш плавною їх зміни в відео вирішено застосовувати каскади не до одного і того ж кадру, а по черзі, тобто один каскад – до першого кадру, другий – до наступного і т.д;

4) запам'ятати і намалювати отримані межі дорожніх знаків.

Клас призначений для збереження списку знаків itemAdapter – (рисунок 3.4).

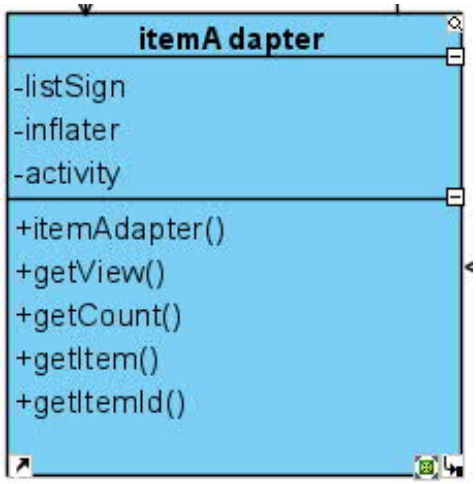

Рисунок 3.4 – Клас itemAdapter

LayoutInflater – це клас, який вміє з вмісту layout-файлу створити View-елемент. Метод який це робить називається inflate(). Є кілька реалізацій цього методу з різними параметрами. Але всі вони використовують один одного і результат їх виконання один – View.

– resource – ID layout-файлу, який буде використаний для створення View.

– R.layout.main.root – батьківський ViewGroup-елемент для створюваного View. LayoutParams від цього ViewGroup присвоюються створюваному View.

– attachToRoot – приєднувати створюваний View до root. Якщо true, то root стає батьком створюваного View. Тобто це рівносильно команді root.addView(View). Якщо false – то створюваний View просто отримує LayoutParams від root, але його дочірнім елементом не стає.

Клас RegconitionActivity призначений для збереження списку розпізнатих знаків (рисунок 3.5).

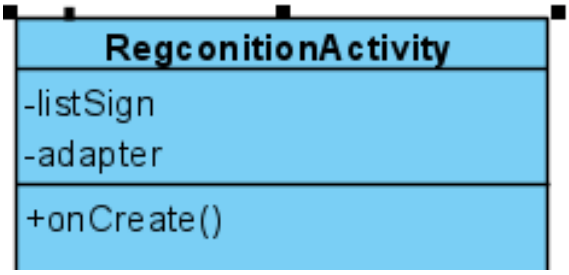

Рисунок 3.5 – Клас `RegconitionActivity`

В класі Utilities знаходяться налаштування, тобто вибір розширення, та вибір minSize і maxSize (рисунок 3.6).

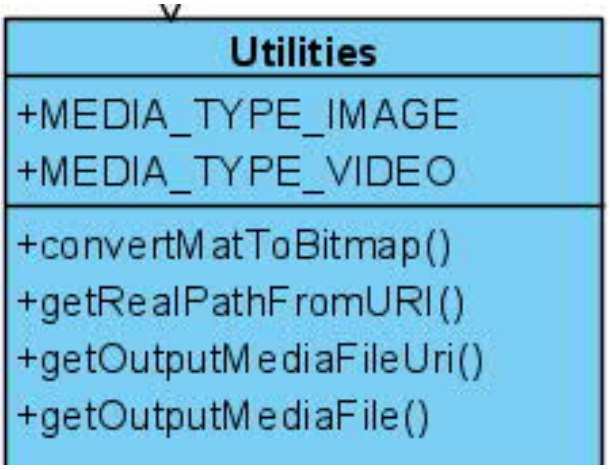

Рисунок 3.6 – Клас Utilities

У класі CameraActivity реалізована робота із камерою та покрокове зчитування кадрів із відео потоку.

CameraBridgeViewBase – це базовий клас, що реалізує взаємодію з камерою та бібліотекою OpenCV. Основний обов'язок – це контролювати, коли камеру можна ввімкнути, обробляти кадр, викликати зовнішнього слухача, щоб зробити будь-які налаштування кадру, а потім намалювати отриманий кадр на екран. Клієнти повинні впровадити CvCameraViewListener.

На рисунку 3.7 зображено клас CameraActivity.

Бібліотека OpenCV дозволяє не тільки читати кадри з відеофайлу, а й захоплювати кадри з зовнішньої камери (наприклад, з веб-камери) в режимі реального часу. Для підключення до камери використовується наступний формат конструктора класу VideoCapture(int index).

В якості значення параметра index вказується індекс камери в системі. Якщо камера одна, то слід вказати значення 0. Для перевірки успішного підключення використовується метод isOpened().

Для підключення до камери також можна скористатися методом `open()`. Метод `open()` спочатку викликає метод `release()` для відключення з'єднання, а потім підключається до камери з вказаним індексом і повертає значення true при успішному підключенні.

Захопити кадр з камери дозволяє метод `grab()`. Якщо кадр успішно захоплений, то метод повертає значення true.

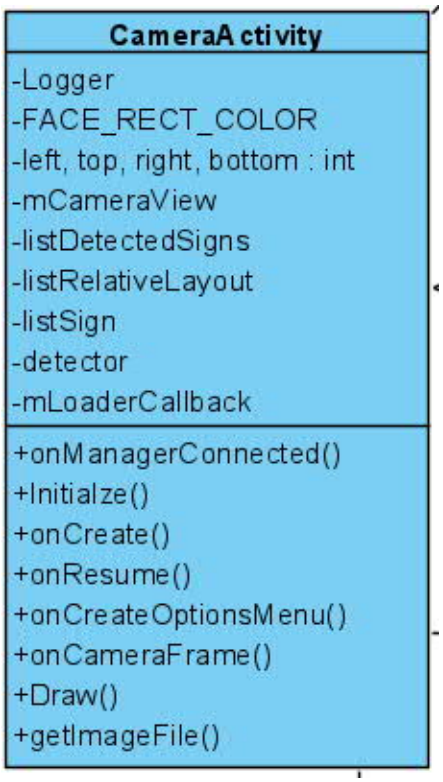

Рисунок 3.7 – Клас `CameraActivity`

Після успішного захоплення викликається метод retrieve() для обробки захопленого кадру і запису його в матрицю image. Метод retrieve() дозволяє також отримати кадр з камери, що має кілька каналів, – наприклад, зі стереокамери. Якщо операція успішно виконана, метод поверне значення true.

Виконати відразу захоплення кадру і його обробку дозволяє метод read(). Якщо операція успішно виконана, метод поверне значення true.

ListView є прокручуваний список елементів. Дуже популярний на мобільні пристрої через свою зручність.

Компонент ListView складніший в застосуванні в порівнянні з TextView і іншим простими елементами. Робота зі списком складається з двох частин. Спочатку ми додаємо на форму сам ListView, а потім заповнюємо його елементами списку.

Компоненту ListView потрібні дані для наповнення. Джерелом наповнення можуть бути масиви, бази даних. Щоб зв'язати дані зі списком, використовується так званий адаптер.

Адаптер для стандартного списку зазвичай створюється за допомогою конструкції new ArrayAdapter(Context context, int textViewResourceId, String [] objects):

– context – поточний контекст;

– textViewResourceId – ідентифікатор ресурсу з розміткою для кожного рядка. Можна використовувати системну розмітку з ідентифікатором android.R.layout.simple_list_item_1 або створити власну розмітку;

– objects – масив рядків;

Метод setAdapter(ListAdapter) пов'язує підготовлений список з адаптером.

Виконати автоматичне вирівнювання гістограми для 8-бітного зображення в градаціях сірого можна за допомогою методу equalizeHist() або класу CLAHE.

Клас CLAHE виконує адаптивне вирівнювання гістограми за допомогою алгоритму Contrast Limited Adaptive Histogram Equalization. Для імпорту використовується бібліотека: import org.opencv.imgproc.CLAHE.

Даний метод покращує контраст напівтонового зображення шляхом перетворення значень його пікселів методом контрастно обмеженої адаптивної еквалізації гістограми. Він працює з невеликими областями зображення. Контраст кожної частини зображення підвищується, що пов'язано з зміною форми гістограми. Після виконання вирівнювання (екалізації) метод об’єднує краї локальних областей із застосуванням білінійної інтерполяції, виключаючи штучно створені межі. Щоб застосувати алгоритм до зображення, слід викликати метод apply()[24].

Клас Mat описує матрицю, яка, наприклад, служить для зберігання зображень. Інструкція імпорту – import org.opencv.core.Mat.

Створити матрицю дозволяють наступні конструктори:

– Mat();

– Mat(int rows, int cols, int type);

– Mat(int rows, int cols, int type, Scalar s);

– Mat(Size size, int type);

– Mat(Size size, int type, Scalar s);

– Mat(long addr).

Параметр rows задає кількість рядків (висоту для зображення), а параметр cols - кількість стовпців (ширину для зображення). Задати розміри можна також за допомогою параметра size. У параметрі type вказується тип елементів матриці. За допомогою параметра s можна заповнити матрицю однаковим значенням. Якщо значення не присвоїли, то елементи будуть мати довільні значення.

Отримати інформацію про розміри матриці дозволяють наступні методи з класу Mat:

– rows() - повертає кількість рядків (висоту для зображення). Формат методу: public int rows();

– cols() - повертає кількість стовпців (ширину для зображення). Формат методу: public int cols();

– width() - повертає ширину для зображення. Формат методу: public int width();

– height() - повертає висоту для зображення. Формат методу: public int height();

– size() - повертає розміри зображення (Size (cols, rows)). Формат методу: public Size size().

submat() - повертає матрицю, що містить елементи з прямокутного діапазону. Зміна значень в цій матриці торкнеться і значення вихідної матриці.

ReiativeLayout (відносна розмітка) знаходиться в розділі Layouts і дозволяє

дочірнім компонентів визначати свою позицію щодо батьківського компонента або щодо сусідніх дочірніх елементів (ідентифікатора елемента). У RelativeLayout дочірні елементи розташовані так, що якщо перший елемент розташований по центру екрана, інші елементи, вирівняні щодо першого елемента, будуть вирівняні відносно центру екрана. При такому розташуванні, при оголошенні розмітки в XML-файлі, елемент, на який будуть посилатися для позиціонування інші об'єкти уявлення, повинен бути оголошений раніше, ніж інші елементи, які звертаються до нього за його ідентифікатором.

Атрибути:

– android:layout_alignParentBottom – вирівнювання щодо нижнього краю;

– android:layout_alignParentLeft – вирівнювання щодо лівого краю;

– android:layout_alignParentRight – вирівнювання щодо правого краю;

– android:layout_alignParentTop – вирівнювання щодо верхнього краю;

– android:layout_centerInParent – вирівнювання по центру по вертикалі і горизонталі;

– android:layout_centerHorizontal – вирівнювання по центру по горизонталі;

– android:layout_centerVertical – вирівнювання по центру по вертикалі;

У класі PhotoActivity реалізована функція роботи із фото, що дозволяє загрузити фото із телефона (рисунок 3.8).

Компонент ImageView призначений для відображення зображень. Знаходиться в розділі Widgets.

Для завантаження зображення в XML-файлі використовується атрибут android:src.

ImageView є базовим елементом-контейнером для використання графіки. Дозволяє завантажувати зображення з різних джерел, наприклад, з ресурсів програми, контент-провайдерів. У класі ImageView існує кілька методів для завантаження зображень:

– setImageResource (int resId) – завантажує зображення за ідентифікатором ресурсу;

– setImageBitmap (Bitmap bitmap) – завантажує растрове зображення;

– setImageDrawable (Drawable drawable) – завантажує готове зображення;

– setImageURI (Uri uri) – завантажує зображення по його URI;

Намалювати прямокутник дозволяє статичний метод rectangle() з класу Imgproc. Формат метода: public static void rectangle(Mat img, Point pt1, Point pt2, Scalar color, int thickness).

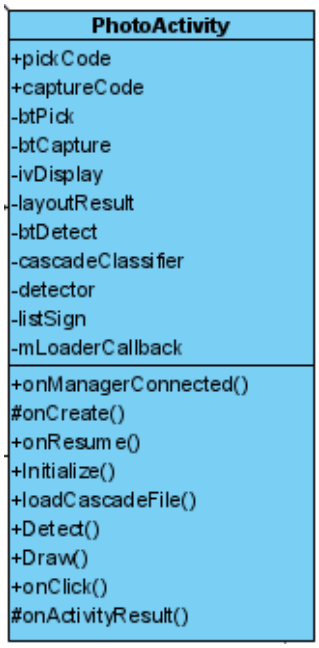

Рисунок 3.8 – Клас `PhotoActivity`

У першому параметрі вказується посилання на зображення, у другому – координати лівого верхнього кута прямокутника, в третьому – координати правого нижнього кута, а в четвертому – колір в форматі BGR або Gray (для зображень в градаціях сірого). За замовчуванням лінія обведення малюється товщиною в один піксель. За допомогою параметра thickness можна вказати іншу товщину. Якщо в параметрі thickness вказати константу FILLED з класу Core, то прямокутник буде малюватися з заливкою без обведення.

У параметрі lineType вказується тип лінії: константи LINE_4, LINE_8 (значення за замовчуванням) або LINE_AA (зі згладжуванням), а в параметрі shift - зсув (значення за замовчуванням: 0).

UML діаграма класів наведена в додатку А на рисунку А.1.

3.2 Програмна реалізація системи

На рисунку 3.9 зображено ініціалізацію кнопок та виклик файлу activity_main.xml в класі `MainActivity` (додаток Б).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Initialize();
    btRuntime.setOnClickListener(this);
    btPickPhoto.setOnClickListener(this);
}
public void Initialize(){
    btRuntime = (Button)findViewById(R.id.btRuntime);
    btPickPhoto = (Button)findViewById(R.id.btPickPhoto);
}
```

Рисунок 3.9 – Ініціалізація кнопок та виклик activity_main.xml

На рисунку 3.10 зображено запуск файлу camera_preview.xml у класі `CameraActivity`, тобто активності на якій буде відображено зображення із камери.

```
private void Initialze(){
    mCameraView = (CameraBridgeViewBase)findViewById(R.id.mCameraView);
    listDetectedSigns = (ListView)findViewById(R.id.listView1);
    listRelativeLayout = (RelativeLayout)findViewById(R.id.listViewLayout);
    mCameraView.setCvCameraViewListener(this);
    //listRelativeLayout.setVisibility(View.GONE);
    mCameraView.setVisibility(SurfaceView.VISIBLE);
    mCameraView.setCvCameraViewListener(this);
}
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    setContentView(R.layout.camera_preview);
    Initialze();
}
```

Рисунок 3.10 – Запуск файлу camera_preview.xml

У класі `Detector` метод `Detect()` виконує етап сегментації, використовуючи каскади класифікаторів (рисунок 3.11).

Для загрузки каскадів класифікаторів створено метод `loadCascadeFile()` (рисунок 3.12)

```
public void Detect(Mat mGray,MatOfRect signs,int type){
    switch (type) {
        case 1:
            if (cascadeClassifier1 != null && !cascadeClassifier1.empty()) {
                cascadeClassifier1.detectMultiScale(mGray, signs,  scaleFactor: 1.1,  minNeighbors: 3,  flags: 0
                        , new Size(minSize1, minSize1), new Size(maxSize1, maxSize1));
            } else {
                Log.e( tag: "s",  msg: "cascade");
            }
            break;
        case 2:
        default:
            if (cascadeClassifier2 != null && !cascadeClassifier2.empty()) {
                cascadeClassifier2.detectMultiScale(mGray, signs,  scaleFactor: 1.1,  minNeighbors: 5,  flags: 0
                        , new Size(minSize2, minSize2), new Size(maxSize2, maxSize2));
            } else {
                Log.e( tag: "s",  msg: "cascade");
            }
    }
}
```

Рисунок 3.11 – метод `Detect()`

```
private void loadCascadeFile(int type){
    try {
        InputStream is;
        File cascadeDir = activity.getDir( name: "cascade", Context.MODE_PRIVATE);
        File cascadeFile;
        switch (type) {
        case 1:
            is = activity.getResources().openRawResource(R.raw.circle);
            cascadeFile = new File(cascadeDir,  name: "circle.xml");
            break;
        case 2:
        default:
            is = activity.getResources().openRawResource(R.raw.triangle);
            cascadeFile = new File(cascadeDir,  name: "triangle.xml");
            break;
        }
        FileOutputStream os = new FileOutputStream(cascadeFile);
        byte[] buffer = new byte[4096];
        int bytesRead;
        while ((bytesRead = is.read(buffer)) != -1) {
            os.write(buffer,  byteOffset: 0, bytesRead);
        }
        is.close();
        os.close();
        switch (type) {
            case 1:
                cascadeClassifier1 = new CascadeClassifier(cascadeFile.getAbsolutePath());
                break;
            case 2:
            default:
                cascadeClassifier2 = new CascadeClassifier(cascadeFile.getAbsolutePath());
                break;
        }
```

Рисунок 3.12 – метод `loadCascadeFile()`

В класі CameraActivity здійснюється обрізання знаку та його обробка, перетворення в градієнт сірого, масштабування та вирівнювання гістограми.

На рисунку 3.13 зображено масштабування та вирівнювання гістограми (додаток Б).

```
for (int i = 0; i <facesArray.length; i++){
    final int ii = i;
    Log.e( tag: "RES",  msg: "start");
    Mat subMat;
    subMat = mGray.submat(facesArray[i]);
    Mat resizeMat = new Mat();
    Imgproc.resize(subMat, resizeMat, new Size( width: 32,  height: 32),  fx: 0,  fy: 0, Imgproc.INTER_CUBIC);
    clahe.apply(resizeMat, resizeMat);
```

Рисунок 3.13 – масштабування та вирівнювання гістограми

На рисунку 3.14 наведено приклад роботи етапу сегментації.

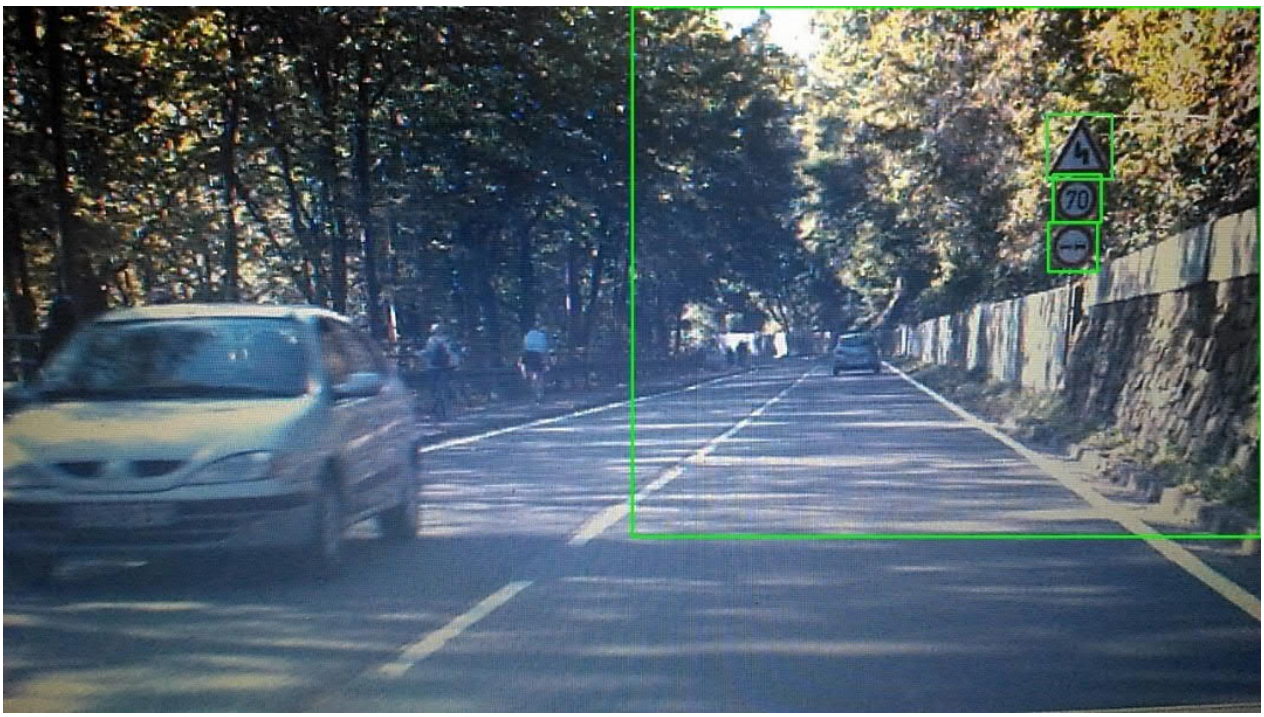

Рисунок 3.14 – Приклад роботи каскадів класифікаторів

Процес класифікації можна описати наступною послідовністю кроків:

1) вирізати отриману після етапу локалізації область зображення;

2) перетворити її в відтінки сірого;

3) змінити розмір області до розміру 32 × 32 методом бікубічної інтерполяції;

4) застосувати контрастне вирівнювання;

5) застосувати навчену нейронну мережу;

6) вивести результат.

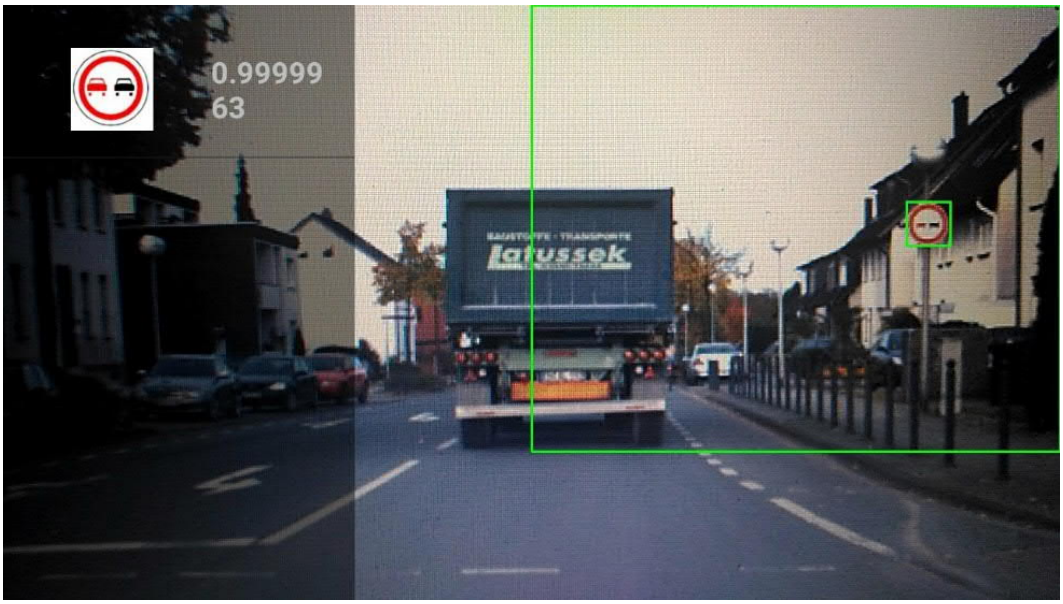

Рисунок 3.15 – Приклад роботи

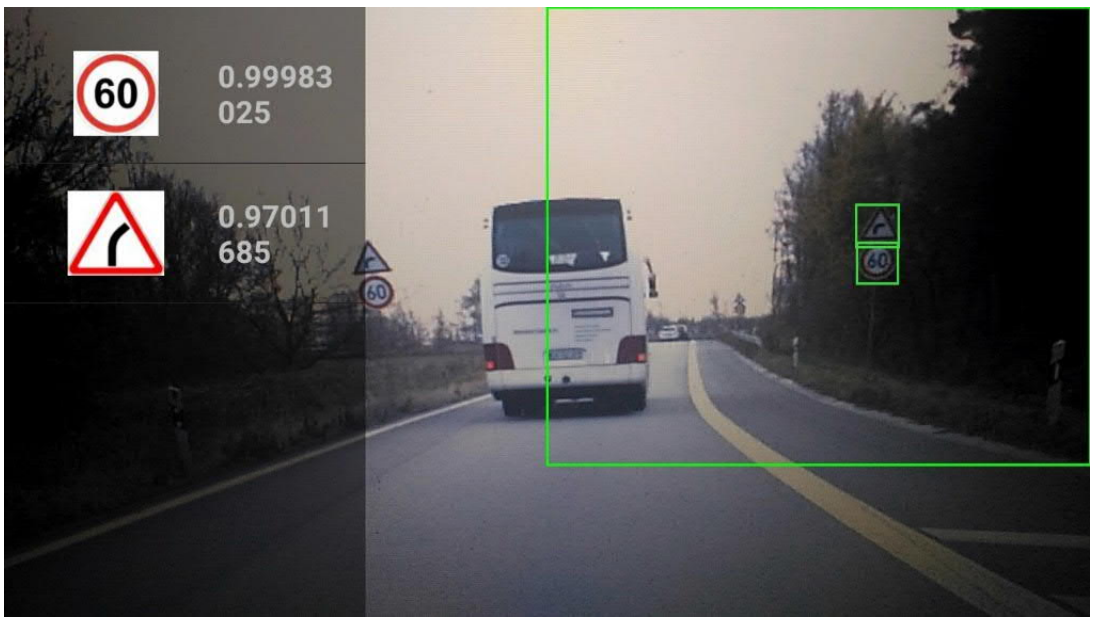

Рисунок 3.16 – Приклад роботи

Час, що витрачається на класифікацію одного знака на смартфоні Xiaomi Redmi 3 Pro, становить в середньому – 10 мс. На рисунку 3.15 і 3.16 наведені приклади роботи.

3.3 Експериментальне дослідження ефективності алгоритмів

Система тестувалася в світлий час доби на смартфоні Xiaomi Redmi 3 Pro. Були підраховані наступні дані: кількість знайдених знаків (вірно локалізованих), кількість незнайдених знаків, кількість помилкових спрацьовувань на етапі локалізації (об'єкт, який не є знаком, визначався як знак), кількість вірно класифікованих, кількість невірно класифікованих. Всього на шляху проходження було 55 знаків. Розглядалися тільки знаки, присутні в навчальній базі. Результати наведені у таблиці 3.1 та 3.2.

Таблиця 3.1 – Результати тестування

Детектування (точність локалізації)	Не детектовані	Хибне детектування	Класифікація (точність класифікації)	Невірно класифіковані
50 (90,90%)	5	3	51 (92,72%)	4

Таблиця 3.2 – Результати тестування (класифікація)

Тип знаку	% розпізнавання
Обмеження швидкості	92,96
Обгін заборонено	91,47
Пішохідний перехід	91,64

Для порівняння систем було обрано RoadAR, OpelEye, Speed limit assist, Road sign information (таблиця 3.3).

Таблиця 3.3 – Порівняння систем

	RoadAR	OpelEye	Speed limit assist	Road sign information	Розроблений алгоритм
Заявлена точність розпізнавання	95%	90%	95%	96%	
Розпізнавання знаків обмеження швидкості	+	+	+	+	+
Розпізнавання інших забороняють знаків	+	+	+	+	+
Реальна точність розпізнавання	85%	75%	70%	75%	91%

З таблиці видно, що окремо існуючі програмні продукти складають рідкість. Більшість комплексів не є окремо встановлюваним програмним забезпеченням, а входять до складу заводських опцій виробника автомобіля. Таким чином, їх комерційна вартість виявляється занадто високою. Всі подібні продукти мають невисокий відсоток точності розпізнавання, що падає ще сильніше при нахилі, частковому перекритті або забрудненості знаку, його повороті, так як зображення зазнають проектні та афінні спотворення. При випробуваннях надійно розпізнаються лише чисті знаки високої контрастності.

У результаті на етапі локалізації дорожніх знаків навчено два каскади класифікаторів: один – для круглих знаків (які забороняють), другий – для трикутних (попереджувальних і знаку "Пішохідний перехід"). Точність їх роботи 85,5% і 83% відповідно. Оскільки класифікатори працюють із зображеннями, отриманими з камери, то якщо об'єкт не буде знайдений на одному кадрі, він може бути знайдений на наступному, тому дана точність прийнятна. Середній час, необхідний одному каскаду для обробки зображення розміру 1280 × 720 на смартфоні становить 530-550 мс. З огляду на те, що обчислення розглядаються на мобільній платформі, зазначений час задовільний, проте для роботи в режимі

реального часу цього недостатньо. Тому було оптимізовано даний метод визначенням зони інтересів, це дозволило скоротити до 160-170 мс.

3.4 Висновки до розділу

Розроблено та реалізовано алгоритми в складі системі розпізнавання дорожніх знаків для мобільних пристроїв на Android.

На етапі класифікації дорожніх знаків була обрана структура згорткової нейронної мережі, що дає кращий результат з точки зору співвідношення точності і швидкості роботи. Її точність – 92%, а час, необхідний на розпізнавання одного знаку на смартфоні складає близько 10 мс. Дані показники досить високі і дозволяють успішно використовувати мережу в режимі реального часу. Об'єднавши результати роботи сегментування і розпізнавання, можна виявити загальний відсоток розпізнавання розробленого алгоритму: ≈ 91%.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи отримано наступні результати.

1. Проведено огляд систем розпізнавання дорожніх знаків, сформульовано вимоги до їх структури і функцій.

2. Проаналізовано наступі методи розпізнавання: метод нейронних мереж, метод SURF, метод опорних векторів, метод Random fores, каскад класифікаторів

3. Розроблено узагальнений алгоритм роботи системи із трьох етапів. На першому етапі сегментації застосовано два каскадні класифікатори, один для круглих знаків і для трикутного та знаку «пішохідний перехід». Для збільшення швидкості локалізації було обрано зону інтересі, що дозволило скороти час виконання етапу у більш ніж два рази. На другому етапі відбувається обрізання знайденого знаку, його масштабування, перетворення в градієнт сірого та вирівнювання гістограми. На третьому етапі відбувається класифікація знаку з допомогою штучної нейронної мережі.

4. Реалізовано згорткову нейронну мережу та реалізовано алгоритми класифікації на її основі. Експериментально підібрано кількість карт на шарах, що дозволило підвищити точність розпізнавання.

5. Програмно реалізовано систему розпізнавання дорожніх знаків для мобільних пристроїв на Android на мові Java.

6. Проведено тестування розробленої системи на смартфоні Xiaomi Redmi 3 Pro. Показано, що реальна точність класифікації дорожніх знаків склала приблизно 91%, що є кращим результатом у порівнянні із існуючими аналогами RoadAR, OpelEye, Speed limit assist, Road sign information.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Принципы распознавания дорожных знаков: веб-сайт. URL: https://studfile.net/preview/8166351/ (дата звернення: 02.07.2020).

2. Баган Я.А., Бабій Ю.В. Алгоритми текстурної сегментації на основі статистичних ознак. III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі». 26 листопада 2020, Тернопіль, Україна. Тернопіль: ЗУНУ, 2020. с.23

3. Бабій Ю.В., Баган Я.А. Алгоритми розпізнавання дорожніх знаків для мобільних пристроїв. III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі». 26 листопада 2020, Тернопіль, Україна. Тернопіль: ЗУНУ, 2020. с.24

4. Как работает система распознавания дорожных знаков: веб-сайт. URL: https://techautoport.ru/sistemy-bezopasnosti/aktivnaya/sistema-raspoznavaniya-dorozhnyh-znakov.html (дата звернення: 15.07.2020).

5. Анализ методов распознавания образов: веб-сайт. URL: https://moluch.ru/archive/150/42393/ (дата звернення: 04.08.2020).

6. Minsky M. An Introduction to Computational Geometry. Cambridge: MiT, 1972. 214p.

7. Speeded-Up Robust Features (SURF) / B. Herbert, E. Andreas, T. Tinne, L. Gool. Computer Vision and Image Understanding. 2008. №110. P. 346-359.

8. Классификация данных методом опорных векторов: веб-сайт. URL: https://habr.com/ru/post/105220/ (дата звернення: 08.08.2020).

9. The Elements of Statistical Learning / Hastie, T and etc. Data Mining, Inference, and Prediction. 2009. №2. P. 746.

10. Freund Y. A. Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence. 1999. №5. P. 771-780.

11. Eichner M. Integrated speed limit detection and recognition from real-time video. IEEE International Intelligent Vehicles Symposium. 2009, №8, P. 626-631.

12. OpenCV library: веб-сайт. URL: https://opencv.org/ (дата звернення: 22.08.2020).

13. The german traffic sign recognition benchmark (GTSRB) dataset: веб-сайт. URL: http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset (дата звернення: 29.08.2020).

14. Image Processing Toolbox: управление яркостью палитры: веб-сайт. URL: http://matlab.exponenta.ru/imageprocess/book3/10/brighten.php (дата звернення: 07.09.2020).

15. Image Processing Toolbox: create predefined 2D filter: веб-сайт. URL: http://www.mathworks.com/help/images/ref/fspecial.html (дата звернення: 08.09.2020).

16. Image Processing Toolbox: добавление шума: веб-сайт. URL: http://matlab.exponenta.ru/imageprocess/book3/10/imnoise.php (дата звернення: 09.09.2020).

17. ColorConversion: веб-сайт. URL: http://www.equasys.de/colorconversion.html (дата звернення: 10.09.2020).

18. Hum, Y. C. Multiobjectives bihistogram equalization for image contrast enhancement. Complexity. 2014. №20. P. 22-36.

19. Krizhevsky, A. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems. 2012, №15, P. 1097-1105.

20. Rumelhart, D.E. Learning Internal Representations by Error Propagation. In: Parallel Distributed Processing, vol. 1. 1986. P. 318-362.

21. Pratt W. K. Digital Image Processing: PIKS Scientific Inside. NY, USA: John Wiley and Sons, Inc., 2007. 782 p.

22. android.hardware.camera2: веб-сайт. URL: https://developer.android.com/reference/android/hardware/camera2/package-summary.html (дата звернення 25.09.2020).

23. Android Camera2 API: веб-сайт. URL: https://habr.com/ru/post/468083/ (дата звернення 26.09.2020).

24. MediaRecorder: веб-сайт. URL: https://developer.android.com/reference/android/media/MediaRecorder?hl=en (дата звернення 26.09.2020).

25. Прохоренок Н. А. OpenCV и Java. Обработка изображений и компьютерное зрение. БХВ-Петербург, 2018. 320 с.

26. Powers L., Snell M. Microsoft visual studio 2008 unleashed. Indianapolis, IN, USA: Sams, 2008. 1248 p.

27. Bradski G., Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library.Cambridge, MA: O'Reilly, 2008. 577 p.

28. Ma W. Y., Manjunath B. S. NeTra: a toolbox for navigating large image databases.Multimedia Systems archive. 1999. V. 3, № 7. P.184-198

29. An efficiently computable metric for comparing polygonal shapes / E. M. Arkin, L. Chew, D. Huttenlocher et al. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1991. Vol. 13. p. 209-216.

30. Fan S. Shape representation and retrieval using distance histograms: Tech. Rep. 01-14: Department of Computing Science, University of Alberta, 2001.

31. M. F. Zakaria, L. J. Vroomen, P. J. A. Zsombor-Murray, J. M. H. M. van Kessel. Fast algorithm for the computation of moment invariants. Pattern Recognition. 1987. Vol. 20, no. 6. p. 639.

32. Field D. J. Relations between the statistics of natural images and the response properties of cortical cells. Journal of the Optical Society of America. 1987. Vol. 4, no. 12. - p. 2370-2393.

33. Fox E. A. Combination of multiple searches. 2nd Text REtrieval Conference (TREC-2). National Institute of Standards and Technology Special Publication 500-215, 1994, p. 243-252.

34. Freeman H. In computer processing of line-drawing images. ACM Computing Surveys (CSUR). 1974. March. Vol. 6. - Pp. 57-97.

35. Ghafoor A., Iqbal R. NKhan S. A. Modified chamfer matching algorithm // Lecture Notes in Computer Science. 2003. Vol. 2690. p. 1102-1106.

36. Gotlieb C. CKreyszig H. E. Texture descriptors based on co-occurrcnce matrices. Computer Vision, Graphics and Image Processing.1990.V.51,no.1,p.70-86.

37. Grosky W., Stanchev P. An image data model. In Proceedings of Advances in Visual Information Systems: 4th International Conference. 2000. p. 227-243.

38. Guerin-Dugue A., Ayache S., Berrut C. Image retrieval: a first step for a human centered approach. Joint Conference of ICI, CSP and PRCM. 2003. p. 21-25.

39. Guironnet M., Pellerin D., Ladret P. Combinaison de descripteurs flous de couleur et d'activité pour le résumé de vidéos // 14éme congrès de Reconnaissance des Formes et Intelligence Artificielle RFIA. 2004.

40. Haddadnia J., Ahmadi M., Faez K. An efficient feature extraction method with pseudo-zernike moment in RBF neural network-based human face recognition system. EURASIP Journal on Applied Signal Processing. 2003.p. 890-901.

41. Haralick R. M., Shanmugam K.7 Dienstein I. Textural features for image classification. IEEE Transactions on Systems, Man and Cybernetics. 1973. November. Vol. 3, no. 6. p. 610-621.

42. Hateren J. H. V., der Schaaf A. V. Independent component filters of natural images compared with simple cells in visual cortex. Transactions of Royal Society of London. 1998. Vol. B265. p. 359-366.

43. Heller K. A., Ghahramani Z. simple bayesian framework for content- based image retrieval. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006). 2006. p. 2110-2117.

44. Hew P. Geometric and zernike moments. Diary, Department of Mathematics, The University of Western Australia. 1996. Octobcr. URL: http://citeseer.ist.psu.cdu/hew96geometric.html.

45. Hoqve S. K., Sirlantzis M. C. F. A new chain-code quantization approach enabling high performance handwriting recognition based on multi- classifier schemes . Seventh International Conference on Document Analysis and Recognition (ICDAR'03). Vol. 2. 2003. 843 p.

46. Howarth P., Riiger S. Evaluation of texture features for content-based image retrieval. Proceedings of CIVR'04. 2004. p. 326-334.

47. Howarth P., Riiger S. Robust texture features for still image retrieval. IEEE Proceedings of Vision, Image and Signal Processing. Vol. 152. 2005. p.868-874.

48. *Efficient* and effective querying by image content: Tech. rep. / C. Faloutsos, W. Equitz, M. Flickner et al.: IBM Research, 1993. *p. 179-187*

49. Березький О.М., Дубчак Л.О., Мельник Г.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма - Комп’ютерна інженерія"/ Під ред. О.М. Березького. Тернопіль:ЗУНУ,2020.32 с.

50. Гураль І.В., Дубчак Л.О. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» /Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.