Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: **Модель технічного зору для розпізнавання дорожніх знаків для безпілотного автомобіля**

Спеціальність 123 – Комп'ютерна інженерія.
Галузь знань: 12 – Інформаційні технології.
Освітня програма «Комп'ютерна інженерія».

Захищено на засіданні
Екзаменаційної комісії № 44
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**.

Виконав:
студент 4 курсу, групи КІ–41
КРИЛОВ Дмитро Андрійович

Керівник: к.т.н., доцент кафедри комп'ютерних систем та робототехніки
БИКОВА Тетяна Володимирівна

Рецензент: к.т.н., доцент кафедри математичного моделювання та аналізу даних
МАЗОРЧУК Марія Сергіївна

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до бакалаврської кваліфікаційної роботи складається з трьох основних розділів, висновків, списку використаних джерел та двох додатків. Обсяг роботи становить **104** сторінок, з яких **46** сторінок є основною частиною, що включає **15** рисунків, **4** таблиці та **11** найменувань у списку використаних джерел, **4** додатки і **5** листингів.

Об'єкт дослідження - це процес розпізнавання дорожніх знаків за допомогою моделі технічного зору, яка інтегрує камеру, алгоритми нейронні мережі та механізми аналізу навколишнього середовища для забезпечення функціонування безпілотного автомобіля.

Предмет дослідження - принципи побудови та функціонування моделі технічного зору в аспекті розпізнавання дорожніх знаків, зокрема технології обробки зображень, алгоритми класифікації в реальному часі та засоби інтеграції системи в архітектуру автопілоту.

Проблема, яку вирішує кваліфікаційна робота - це зазвичай безпілотні автомобілі, які стикаються з проблемами до яких можна віднести розпізнавання дорожніх знаків, виявлення перешкод та адаптація до умов навколишнього середовища. Моделі автопілотів які існують на даний час, стикаються з поганою точністю розпізнавання об'єктів, що може призводити до помилкових рішень під час руху. Кваліфікаційна робота присвячена розробці моделі технічного зору для розпізнавання дорожніх знаків, спрямована на вирішення цих проблем.

Область застосування - автономний транспорт, системи допомоги водієві ADAS. Розроблена модель може бути використана для підвищення безпеки дорожнього руху та автоматизації транспортних засобів.

Методи дослідження - було використано методи аналізу принципів технічного зору, оцінки сучасних технологій розпізнавання об'єктів, розробки та тестування моделі на основі глибокого навчання із застосуванням аугментації даних.

Ключові слова: *датасет, автопілот, модель, система, точність, транспорт, технічний зір, обробка дани*

ABSTRACT

The explanatory note to the bachelor's qualification work consists of three main sections, conclusions, a list of references and two appendices. The volume of the work is **104** pages, of which **46** pages are the main part, including **15** figures, **4** tables, **11** references in the list of used sources and **4** appendices and **5** listings

The object of study is a technical vision system for an unmanned vehicle, which is a set of methods for image processing and road sign recognition algorithms.

The subject of the study consists of the principles of technical vision, road sign recognition algorithms and the possibility of their use in unmanned vehicle systems.

The problem solved by the qualification work is usually unmanned vehicles that face problems that include road sign recognition, obstacle detection and adaptation to environmental conditions. Currently available autopilot models face poor object recognition accuracy, which can lead to erroneous decisions while driving. The qualification work, dedicated to the development of a vision model for road sign recognition, aims to solve these problems.

Scope - autonomous vehicles, ADAS driver assistance systems. The developed model can be used to improve road safety and vehicle automation.

Research methods - used methods of analyzing the principles of technical vision, evaluating modern object recognition technologies, and developing and testing a model based on deep learning using data augmentation.

Keywords: *dataset, autopilot, model, system, accuracy, transportation, technical vision, data processing*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

Python – це високорівнева об'єктно-орієнтована мова програмування з інтерпретатором, яка характеризується суворою та динамічною типізацією. Цю мову програмування можна використовувати в різних галузях, включаючи web-розробку, наукові дослідження, машинне навчання, робототехніку та кібербезпеку.

Arduino IDE (Integrated Development Environment) – це середовище розробки, призначене для програмування плат Arduino та інших пристроїв. Середовище використовує мову C/C++, завдяки чому є доступним для широкого кола розробників.

HTTP (HyperText Transfer Protocol) – це протокол для передачі тексту гіперпосиланням. Основан на обміні даними в мережі Інтернет та є частиною серії протоколів клієнт-серверної передачі.

WI-FI (Wireless Fidelity) – технологія, яка була розроблена об’єднанням WECA на базі стандарту широкосмугової мережі IEEE 802.11.

CNN (Convolutional Neural Networks) – це клас який відноситься до глибоких штучних нейронних мереж прямого поширення, ефективно використовується для аналізу зображень.

HSV (Hue, Saturation, Value) - це колірна модель, заснована на трьох характеристиках кольору. Вона є не лінійним перетворенням моделі RGB.

API (Application Programming Interface) - це інтерфейс для взаємодії. В ньому містяться правила та інструменти для зв’язку з сервісом або програмою.

ПДР (Правила Дорожнього Руху) - це нормативний документ, який забезпечує безпеку на дорозі та регулюється виконавчою владою України.

ВСТУП

Актуальність роботи. У нашому світі точне й швидке розпізнавання дорожніх знаків стає головним для безпеки транспортних засобів, які використовують автопілот. Населені пункти перенасичені автомобілями, це підвищує ризики аварійності, зокрема через несвоєчасне реагування системи керування. Пошкоджені, забруднені або закритті перешкодами знаки, а також їх різноманіття в різних регіонах, ускладнюють розпізнавання, що призводить до порушень **ПДР**, зіткнень та інших небезпечних ситуацій.

Розробка надійних моделей технічного зору, здатних ідентифікувати та класифікувати дорожні знаки в реальному часі, набуває важливого значення для транспорту який в майбутньому буде керуватися без допомоги людини. Такі системи мають забезпечувати не тільки базове розпізнавання, а й адаптацію до змінних умов, таких як нічний час, погодні умови, а також динамічні перешкоди. Ці нюанси стають основою для прийняття рішень у системах керування транспортом, впливаючи на вибір швидкості руху, траєкторії та дотримання правил.

Крім того, впровадження моделей розпізнавання знаків може суттєво підвищити ефективність транспорту, а саме зменшити затори під час періоду максимального трафіку шляхом оптимізації маршрутів, уникнути несподіваних зупинок через обмеження швидкості та уникнення штрафних санкцій через порушення правил.

Існуючі технології, такі як глибоке навчання, відкривають нові можливості для обробки зображень у реальному часі. Але їх обробка залежить від різноманітності тренувальних даних, які мають враховувати можливі сценарії розвитку подій, від стандартних знаків до рідкісних або тимчасових.

Розвиток технічного зору не лише покращує функціонування транспортних засобів, але й відкриває шлях до розумних систем, де автомобілі, застосовуючи автопілот, взаємодіють з інфраструктурою, зменшуючи кількість дорожніх пригод та підвищуючи ефективність всієї транспортної системи.

Мета дослідження - створення та аналіз моделі технічного зору для безпілотного транспортного засобу, яка здатна в режимі реального часу розпізнавати дорожні знаки та інтегрувати отриману інформацію в систему автономного керування транспортним засобом.

Робота передбачає вивчення методів покращення якості класифікації знаків, включаючи аугментацію даних для тренування моделі. Одним з важливих аспектів є оптимізація швидкості обробки зображень, щоб забезпечити миттєву реакцію на зміну дорожньої обстановки.

Крім того, робота оцінює практичне застосування розробленої моделі у реальних транспортних системах. Аналіз охоплює адаптацію до інфраструктури, де дані з знаків інтегруються у глобальну систему управління потоком транспорту.

Результати дослідження мають стати основою для подальшого вдосконалення автопілоту, зокрема для використання в різних регіонах з урахуванням локальних особливостей дорожніх знаків.

Об’єкт дослідження - це процес розпізнавання дорожніх знаків за допомогою моделі технічного зору, яка інтегрує камеру, алгоритми нейронні мережі та механізми аналізу навколишнього середовища для забезпечення функціонування безпілотного автомобіля.

Предмет дослідження - принципи побудови та функціонування моделі технічного зору в аспекті розпізнавання дорожніх знаків, зокрема технології обробки зображень, алгоритми класифікації в реальному часі та засоби інтеграції системи в архітектуру автопілоту.

У процесі дослідження розв’язувалися наступні завдання.

1. Аналіз принципів роботи технічного зору у системах безпілотних транспортних засобів;
2. Оцінити існуючі технології та методи розпізнавання об'єктів у дорожньому середовищі та їх застосування;
3. Вивчити алгоритми машинного навчання та технічного зору, що використовуються для розпізнавання знаків;

4. Розробити модель технічного зору для системи автопілота, орієнтуючись на ефективність розпізнавання отриманих даних;
5. Провести тестування та оцінку розробленої моделі в різних умовах;
6. Оцінити вплив інтеграції системи технічного зору на продуктивність безпілотного автомобіля;
7. Визначити основні проблеми та виклики в розпізнаванні дорожніх знаків та описати можливості для подальшого вдосконалення;

РОЗДІЛ 1.
АНАЛІЗ ТА КОНЦЕПЦІЯ ТЕХНІЧНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

1.1. Поняття та принципи роботи технічного зору в автомобільних системах

Людство здобуває інформацію про оточення за допомогою свого зору і далі обробляє її у людському мозку. Беручи до уваги це явище, у 50-х роках XX столітті у науковців виникли роздуми про здатність до машинного створення цього процесу. Як зазначається у статті: «У історії розвитку машинного зору можна виділити наступні важливі роки: 1958 г - вчений Френк Розенблатт з університету Корнеля створив комп'ютерну реалізацію персептрона. Цей пристрій моделював схему розпізнавання образів людським мозком. Апаратний варіант Mark I Perceptron був побудований в 1960 р. і призначався для розпізнавання зорових образів» [10]. Ілюстрація Mark I Perceptron, зображено на рис. 1.1.

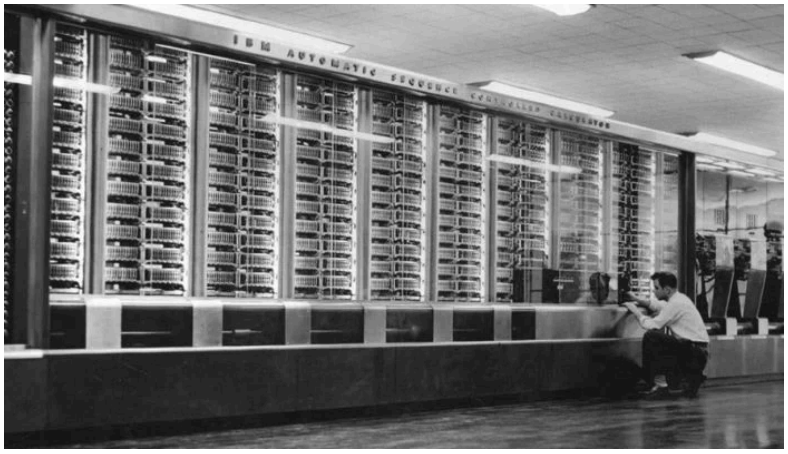


Рисунок 1.1 – Обчислювальна машина Mark I Perceptron.

Технічний зір є ключовим чинником технологій нашого часу. Він являвся головним з методів автоматизації дій із застосуванням робототехніці і комп'ютерних технологій. Системи технічного зору зосереджені на перетворенні зображення, що

поступає з отримуючих пристроїв, з виконанням операцій на основі отриманих даних.

Найбільш потребується у робототехніці, автомобільній промисловості, військовій галузі та науці. Це можна пов'язати з тим, що в цих сферах вже з’явилися спеціалізовані задачі, рішенням якими питаннями займаються дослідники в галузі глибокого навчання. Завдяки ускладненню вирішуваних задач, аналіз візуальних даних і її програмна обробка стають усе більш затребуваними.

Необхідність цієї технології виникає в ситуаціях, коли це можна віднести до ризику життя, також буває пов’язана з особливостями людини як істоти, якій властиво помилятись, отримувати обмежену кількість інформації, та обробляти дані з повільної швидкістью. В ідеальних умовах вчені прагнуть про створення універсальної моделі, яка розвивається за аналогією зростання людей. Опираюсь на ці побажання, розробники в області технічного зору сьогодні вирішують нові та важкі задачі для розвитку цієї галузі.

В автомобільних системах технічний зір об'єднує комплекс програмних та апаратних засобів, які допомагають автомобілю бачити середовище яке оточує його та проводити аналіз отриманого зображення. Після чого приймати певні дії з дотриманням правил дорожнього руху.

Отримуючи дані з різних ракурсів, транспортного засобу, система виконує обробку зображень за такими принципами:

- Збільшення контрастності;
- Зменшення шумів;
- Підвищення різкості;

Ці дії дозволяють більш детально упізнавати потрібні елементи для технічного зору, такі як: автомобілі, знаки дорожнього руху та дорожня розмітка. Влаштовані алгоритми, аналізують вхідне зображення для класифікації, виявлення та позиціювання певних об’єктів, утворюючи координати з отриманого зображення у координати простору, це дозволяє визначити відстань від об’єктів, напрямок руху та інше.

Система дозволяє працювати в режимі реального часу і своєчасно повідомляти водія про небезпеку при керуванні авто або автоматично втручатись в цей процес. До прикладу, можна привести активацію гальмування чи корекції траєкторії руху. Також автомобільні системи об’єднують інформацією технічного зору з декількох пристроїв, що надає надійніше та точніше сприйняття навколишнього середовища.

1.2. Методи розпізнавання об’єктів

Існуючі методи розпізнавання об'єктів засновані на глибокому навчанні. Можна виділити три основні методи, а саме:

- Метод оптичного потоку;
- Метод використання фонової моделі;
- Метод використання траєкторій;

Методи знайшли застосування у розпізнаванні та відстеженні рухомих об'єктів при отриманні графічної інформації з пристроїв. За основу практичного виконання методу взяли архітектуру нейронних мереж. Щоб розпочати взаємодію з зображенням, треба провести обробку та видалити непотрібну інформації для покращення аналізу. Наступним кроком потрібно побудувати модель, яка буде розпізнавати та відстежувати рухаючий об’єкт. Після того як модель прийшла навчання її можна використовувати за призначенням. Модель працює за принципом детектора, у разі виявлення певної ділянки інформації йде її відстеження.

Метод оптичного потоку входить до основних методів відстеження у реальному часі. Для аналізу відео в реальному часі використовуються методи оптичного потоку, які базуються на змінах яскравості пікселів з плином часу [2]. Головна ідея полягає в тому, якщо об'єкт рухається відносно камери, то його пікселі будуть змінюватись в часі. Метод використовується для визначення напрямку та швидкості руху об'єкта на зображенні. За допомогою цього функціоналу, можна відстежувати напрям руху окремих пікселів. Для відстеження об'єкта за допомогою методу оптичного потоку обрати область для обчислення, щоб обчислити потік кожного з пікселів в даній область оптичного потоку. Потім можна обчислити середнє значення векторів руху для всіх пікселів у вибраній області, щоб отримати

вектор руху для всього об'єкта. Цей вектор можна використовувати для відстеження руху об'єкта на зображенні, як на рис. 1.2.

Проблеми методу оптичного потоку: чутливість до освітлення та тіней, зміна розміру об'єкту, форми або швидкості руху на вхідному зображенні. Щоб досягти кращу точність можна об'єднувати метод оптичного потоку з іншими методами, наприклад, з методом використання фонової моделі.

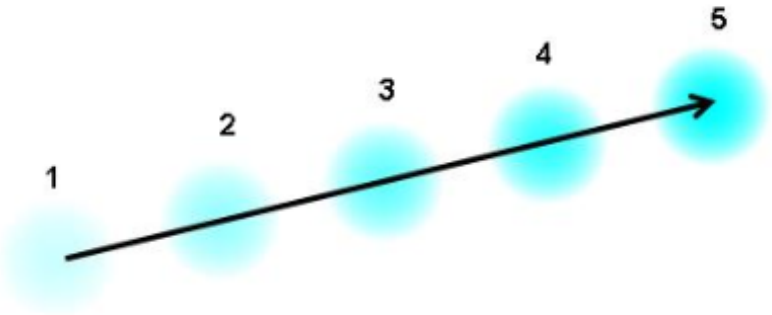


Рисунок 1.2 – Вектор оптичного потоку руху об'єкта в послідовності відео.

Метод використання фонової моделі є одним із найбільш розповсюджених способів для визначення та спостереження за рухомими об'єктами у відео. Метод ґрунтується на аналізі початкових відеоматеріал та визначає фонове зображення, а саме зображення без рухів об'єкту. Потім фонове зображення порівнюється з іншими кадрами відео.

Коли в кадрі з'являється рух, він виділяється на фоновому зображенні, завдяки чому відбувається розпізнавання об'єкта і відстеження його шляху. Цього можна досягти, віднімаючи фон із нового кадру, або порівнюючи кожен піксель нового кадру з відповідним пікселем фонового зображення, як показано як показано на рис. 1.3.

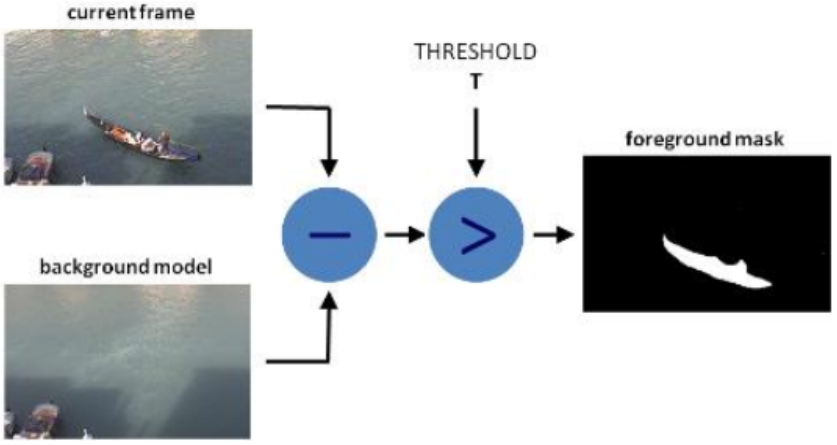


Рисунок 1.3 – Видалення фону з нового кадру або порівняння кожного пікселя нового кадру.

Основним плюсом цього методу є висока швидкість розпізнавання та мінімальний рівень відхилень у розрахунках. Але також є недоліки цього методу: чутливість до освітлення зображення або якщо є зайві деталі на фоновому тлі .

Метод використання траєкторії є методом відстеження об'єктів , де об'єкти знаходиться у взаємодії між собою, в одному середовищі. Ідея цього методу ґрунтується на розрахунку траєкторії руху об'єкту, тобто його позиція в умовному просторі буде залежати від часу. Після чого аналізується траєкторія об’єкту з метою виявлення певних закономірностей у його поведінці, наприклад: зупинка, зміна руху в іншу сторону та збільшення або зменшення швидкості пересування у середовищі.

Метод застосовується коли треба дізнатись ймовірні дій знаючи тільки траєкторію, до цих дій можуть входити небезпечні маневри, зупинки у певній точці, тощо, що відображено на рис. 1.4. Щоб отримати траєкторію руху об'єкта використовуються технології технічного зору та метод відстеження певної точки, розпізнавання розміру, форми та її положення у просторі порівнюючи різні кадри відеозображення. Щоб проаналізувати траєкторію руху використовуються методи машинного навчання, такі як:

- Класифікація;
- Кластеризація;

Наведенні методи дозволяють виявляти закономірності та особливості поведінки об'єкту.

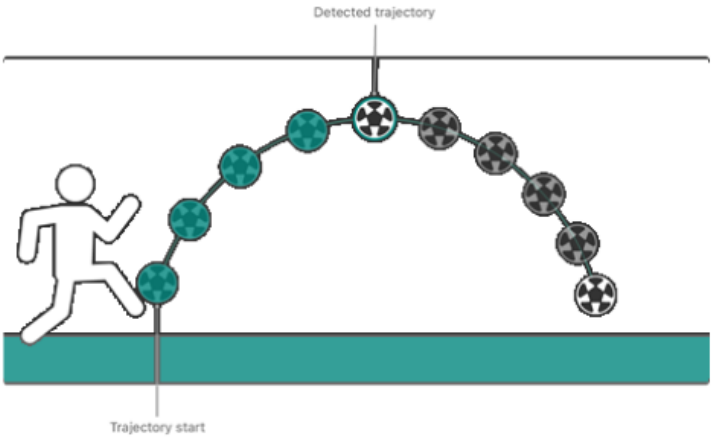


Рисунок 1.4 – Виявлення підозрілих дій та маневрів об'єктів на основі їх траєкторій.

1.3. Огляд існуючих систем розпізнавання

Існуючі системи розпізнавання дорожніх знаків розробляються як інтегровані програми у блоки управління транспортним засобом. В цих системах об’єднується алгоритми обробки зображення з декількох камер та методи вивчення інформації машинного навчання.

Сучасні системи розпізнавання засновуються на різних фундаментальних підходах, які еволюціонували від класних до методів глибокого навчання. Вже класичні алгоритми **Haar-каскади** та **Histogram of Oriented Gradient** автоматично можуть визначати основні ознаки об’єктів. Такі моделі як **R-CNN** та **Fast R-CNN** вже покращують точність розпізнавання, але все ще потребують велику ресурсомісткість через генерацію регіонів та класифікацій.

Одноетапні детектори на кшталт **You Only Look Once** та **Single Shot MultiBox Detector** зробили значний крок до покращення галузі розпізнавання, забезпечивши високу швидкість обробки за допомогою прогнозування координатів об’єкта на зображенні, не виходячи за рамки однієї мережі. Дослідження які були проведені на основі **YOLO** та **EfficientDet** продемонстрували, що моделі вже досягли балансу точності та швидкості. В дослідженнях також було сказано, що середня точність

досягає 65% при 40 кадрах в секунду, спираючись на датасет **COCO** в секунду, коли **EffecientDet** досягає точність в 74% за рахунок ускладнення архітектури [8].

На даний час для розпізнавання дорожніх знаків самою ефективно моделлю є легковажна модель. Самою відомою легковажною моделью представляється **MobileNet** у комбінації з **SSD**. Такі типи моделей оптимізовані для вбудованих систем з необмеженою кількістю ресурсів.

Окрему увагу слід приділити гібридним підходам в яких використовуються **CNN** з поєднанням трансформерів, щоб покращити контекстний аналіз. Такі системи краще розпізнати знаки у складних ситуаціях, розпізнаючи знаки які частково закриті або знаходяться не під стандартизованим кутом. Використання мереж урахування просторово-часових залежностей, на основі **LSTM** або 3D-згорток, сприяють аналізу послідовні кадрів, де критично важливо передбачати рух людей на ділянці дороги.

Однак все ще актуальні залишаються виклики помилкового спрацювання, при великому рівні артефактів на зображенні, так як вони обмежені здатністю при зміні географічних особливостей знаків та енергоефективності при тривалому використанні. Популярні тези вказують на зростання ролі в цій галузі напів самокерованого навчання та синтетичних датасетів, що дозволить зменшити прив’язаність до певного регіону.

1.4. Аналіз алгоритмів машинного навчання та технічного зору для розпізнавання знаків.

Алгоритми машинного навчання та технічного зору є частиною комплексного підходу, що об'єднує методи обробки даних та комп’ютерний аналіз. В цій галузі існують три основні парадигми:

- Контрольовані алгоритми
- Неконтрольовані алгоритми
- Навчання з підкріпленням

Контрольовані алгоритми - це алгоритми які згорткові у нейронній мережі. Вони більш використовуються в задачах з класифікацією, аналізуючи розміщені дані

шляхом автоматичного виділення ознак [4]. Основу машинного навчання в цій галузі становлять три ключові парадигми: контрольоване, неконтрольоване навчання та навчання з підкріпленням. **CNN** використовує шари фільтрів щоб визначити локальні патерни, що робить цей алгоритм більш ефективним для роботи з великим об’ємом. Цей алгоритм представлений у рис. 1.5.

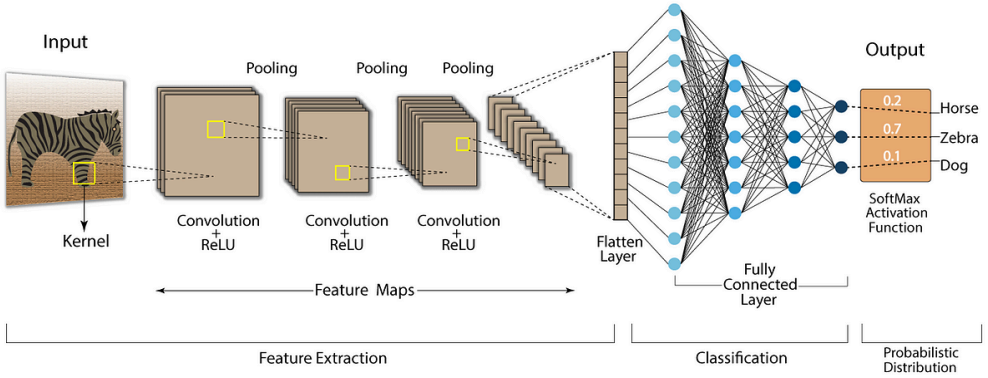


Рисунок 1.5 – Схема роботи алгоритму CNN.

Метод опорних векторів робить бінарне розділення даних, відрізняючи дорожні знаки від інших об’єктів. Неконтрольовані алгоритми k-середні, об’єднують знаки за схожістю, а саме за формою та кольором. Цей метод буде гарно застосовуватись для виявлення невідомих типів дорожніх знаків та аномалій. Навчання підкріплене до систем автономного руху транспортного засобу, де модель отримує зворотній зв’язок через “винагороду” за коректне розпізнавання дорожнього знаку.

Технічний зір виконує попередню обробку зображення та його аналіз. За допомогою фільтрів Гаусса при етапі нормалізації вирівнюється контрастність, яскравість та видаляються всі шуми. Також використовують попрогову обробку, що розділяє пікселі за яскравістю або використовуються детекцію країв за алгоритмом Canny. Після чого виконується виділення ознак, таких як:

1. Розпізнається форма та співвідношення сторін;
2. **HSV** проводить ідентифікацію чорних та синіх знаків;
3. Текстурні характеристики;

Для класифікації використовуються, як традиційні підходи зіставлення з іншим кадром, так і глибоке навчання нейронної мережі. Одним з важливих етапів є постобробка. Вона відповідає за корекцію помилок через контексту інформацію або інтеграцію з системою глобального позиціювання для більшої точності розпізнавання типів знаків, на певній ділянці дороги. Більшість систем які існують, поєднують в собі технічний зір з іншими датчиками, наприклад, лідерами та радарами, що підвищує його надійність.

Але досі існують проблеми з нестачею якісних даних для тренування моделей, зміна погодних умов при зйомці, різні дорожні помітки та знаки в різних країнах. Перспективні напрямки включають в себе використання трансферного навчання для адаптації системи до нового навколишнього середовища, генеративність мережі **GAN** для синтезу даних для тренування та оптимізацію алгоритмів для мобільних пристроїв. Ефективність систем залежить від їх синергії між алгоритмами.

1.5. Визначення ключових викликів та проблем у розпізнаванні дорожніх знаків.

Незважаючи, що протягом XX та XXI століття був великий прогрес у сфері машинного навчання та технічного зору, але розпізнавання дорожніх знаків залишається складною задачею з деяким переліком проблем. Першою та найпоширенішою проблем є зміна умов навколишнього середовища. Освітлення може різко змінюватись протягом дня, воно залежить від погоди, що створює перешкоди для точного розпізнавання знаків. Яскравим прикладом буде відблиск сонячного сяйва від знаку або дощу, знижують контрастність об’єкту який треба знайти системі. Тінь також відіграє велику проблему в розпізнаванні та затемнити дорожні знаки. Системи засновані на алгоритмах, які чутливі до HSV-фільтрації та часто роблять помилки в умовах сутінків або штучного освітлення, де спектральні характеристики дорожніх знаків сильно відрізняються.

Ще одним з серйозних викликів є **різноманітність дизайну дорожніх знаків**. Навіть стандартизація знаків може відрізнятися розміром, формою, кольоровою гамою або символікою. Це ускладнює створення універсальної моделі для всіх країн.

Системи, навчені на даних з одного регіону, часто демонструють низьку ефективність у інших географічних зонах через відмінності у використанні мовних написів або спеціальних символів. Ця проблема посилюється відсутністю єдиної глобальної бази даних із зразками знаків, що обмежує можливості тренування нейронних мереж. Крім того, тимчасові дорожні знаки, встановлені під час ремонтних робіт, часто мають нестандартний вигляд або розміщення, що потребує додаткової адаптації алгоритмів.

Важливу роль відіграють **технічні обмеження апаратних засобів**. Камери з низькою роздільною здатністю або недостатньою частотою кадрів можуть пропускати знаки, особливо на високих швидкостях руху. Шуми на зображенні, спричинені вібрацією транспортного засобу або забрудненням об’єктива, також знижують точність розпізнавання. Попередня обробка зображень, зокрема фільтрація Гаусса або використання детекторів країв Canny, не завжди ефективно усувають ці артефакти, особливо в умовах динамічного руху.

Динамічні перешкоди, такі інші транспортні засоби, пішоходи або рекламні конструкції, часто перекривають знаки, роблячи їх непомітними для системи. Навіть сучасні методи, засновані на відстеженні траєкторій або використанні фонової моделі, не завжди здатні відокремити об’єкт інтересу від таких перешкод. Часткове перекриття знака, наприклад, гілкою дерева або снігом, може призвести до неправильної класифікації, оскільки нейронні мережі, особливо CNN, працюють із цілісними образами. Це вимагає вдосконалення алгоритмів сегментації та використання контекстуальних даних, таких як геолокація, для передбачення можливих знаків на певній ділянці дороги.

Обмеженість обчислювальних ресурсів у реальному часі є ще однією критичною проблемою. Складні архітектури нейронних мереж, такі як глибокі CNN, потребують потужного обладнання для обробки даних, що не завжди доступне в автомобільних системах. Необхідність балансу між швидкістю та точністю призводить до компромісів: спрощені моделі можуть працювати швидше, але з

меншою надійністю. Це особливо відчутно в умовах міського руху, де знаки з’являються часто та непередбачувано.

Не менш важливим викликом є **якість та обсяг даних для тренування моделей**. Хоча сучасні методи, такі як генеративні змагальні мережі GAN, дозволяють синтезувати додаткові зразки, більшість навчальних наборів все ще мають обмежену різноманітність. Зображення знаків у рідкісних погодних умовах або при екстремальних кутах огляду представлені недостатньо, що знижує узагальнюючу здатність моделей. Крім того, розробка систем, здатних адаптуватися до нових типів знаків без повного перетренування, залишається актуальною проблемою.

Інтеграція систем розпізнавання з іншими модулями автономного керування також створює складності. Система має не лише точно ідентифікувати знак, але й коректно інтерпретувати його у контексті поточного стану дороги, враховуючи інші об’єкти та правила. Це вимагає вдосконалення мультимодальних алгоритмів, здатних аналізувати дані з різних джерел синхронно.

Варто враховувати й антропогенні фактори. Вандалізм, пошкодження знаків або їхнє тимчасове приховування роблять їх не розпізнаваними для стандартних систем. У таких випадках важливим стає використання альтернативних джерел інформації, таких історичні дані з системи глобального позиціювання або карти з відмітками про знаки, що дозволяє компенсувати можливі прогалини. Подолання цих викликів потребує комплексного підходу:

1. Поєднання передових алгоритмів машинного навчання;
2. Вдосконалення апаратних компонентів та інтеграції з допоміжними системами;

Перспективними напрямками є розвиток трансферного навчання для швидкої адаптації моделей до нових умов, використання квантових обчислень для прискорення обробки даних та створення міжнародних стандартів для уніфікації дизайну дорожніх знаків. Лише за умови синергії цих складових можна досягти

високої надійності та точності систем розпізнавання, що є критичним для безпеки автономного транспорту.

Висновки за розділом 1

Системи технічного зору в автомобілях об'єднують в собі обробку зображення, нейронні мережі та пристроїв які розрізняють об’єкти навколо себе у реальному часі. Ці системи спираються на методи глибокого навчання, аналіз траєкторії та оптичного потоку, це дозволяє розпізнавати об’єкти навіть при під час руху транспорту. Також системи покращують безпеку, пристосовуючись до зміни дорожніх умов, але їх ефективність залежить від якості вхідних даних, адаптації до дизайну дорожніх знаків у різних країнах та освітлення.

Головними проблемами залишаються чутливість до змін середовища, технічні обмеження апаратури та нестача уніфікованих даних для навчання моделей. Подолання цих бар’єрів потребує розвитку трансферного навчання, синтезу даних через GAN-мережі та інтеграції з картографічними системами.

РОЗДІЛ 2.
ПРОЕКТУВАННЯ МОДЕЛІ ТЕХНІЧНОГО ЗОРУ

2.1. Архітектура системи технічного зору.

Система розпізнавання дорожніх знаків складається з кількох взаємопов'язаних елементів. Одним із ключових компонентів є модель автомобіля. Вона включає в себе такі основні частини:

L298N Motor Drive Module

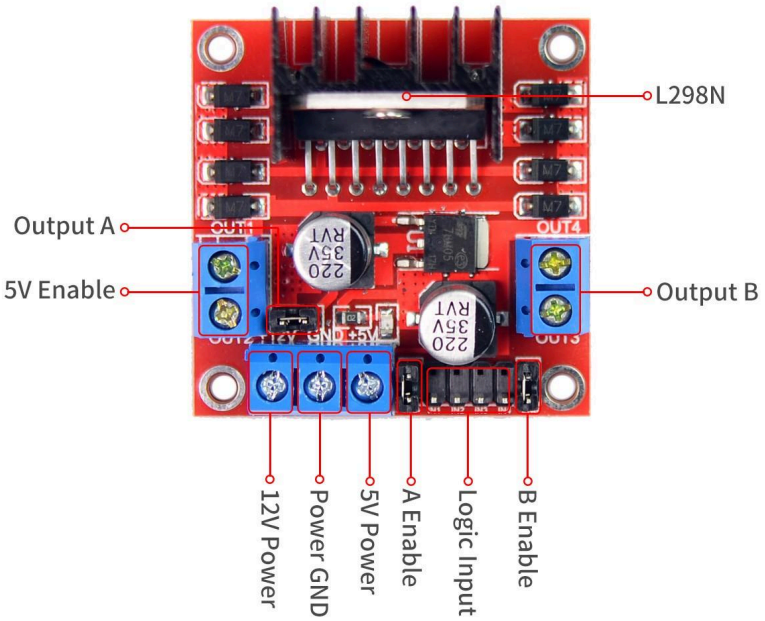


Рисунок 2.1 – Модуль L298N драйвер крокового двигуна.

Драйвер L298N використовується для управління двигунами постійного струму, двофазними крововими двигунами з 4-х або катушками реле. Він працює з напругою від 5 до 35 вольт. Модуль фіксується на поверхні гвинтами – для цього на його платі є чотири монтажні отвори.

5V DC geared motor

Рисунок 2.2 – Двигун 5V DC geared motor.

Він призначений для застосувань, де потрібне підвищене крутний момент та контроль швидкості. Він оснащений вбудованим редуктором, що робить його компактним та довговічним. Мотор працює від постійного струму з напругою 5V. Для зручної інтеграції в робототехнічні системи, автоматизовані механізми або інші пристрої на корпусі передбачені спеціальні отвори або кріплення для фіксації гвинтами.

ESP32 - WiFi Camera Module

Рисунок 2.3 – Модуль камери на основі ESP32.

Модуль ESP32 призначенний для бездротового підключення до камери та IoT проектів. Він підключає камеру до плати, яка передає зображення через **WI-FI** або Bluetooth підключення. Модуль працює в діапазоні напруг від 3.3 до 5 В [3].

Також до конструкції моделі входить:

- 1 × 18650 акумуляторна батарея з вимикачем;
- 10 × мідний стовп М3 * 15;
- 4 × Дюпон жіночий до жіночого 20 см;
- 2 × лінія Дюпона чоловіча до жіночої 20 см;
- 4 × Кронштейн для кріплення двигуна;
- 16 × М3 * 10 гвинтів з плоскою головкою;
- 8 × М3 * 12 гвинтів з плоскою головкою;
- 24 × М3 гайки;
- 8 × М3*30 гвинти з круглою головкою;
- 2 × М1.6*10 гвинти;
- 2 × М1.6 гайки;
- 4 × М3*8 гвинти з плоскою головкою;
- 1 × Синій тепловідвід для драйверів, малий (TMC2100);
- 1 × Корпус материнської плати ESP32;

Загальна структура системи:

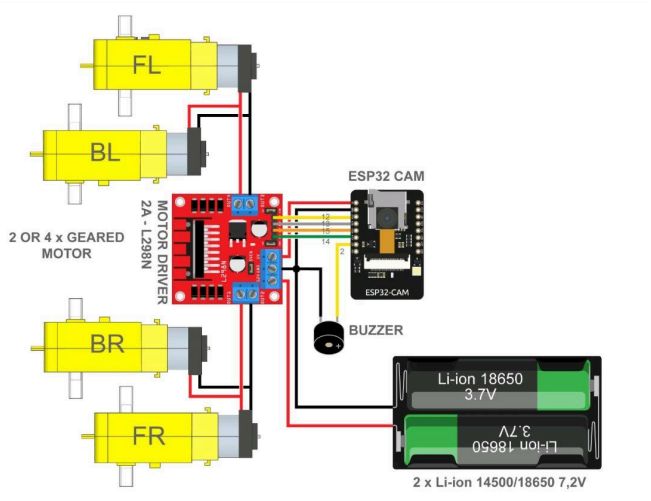


Рисунок 2.4 – Структурна схема моделі.

Компоненти які були визначені в цьому пункті об’єднуються в єдину систему та забезпечує чітке розпізнавання дорожніх знаків у реальному часі. З камери модулю камери EPS32 інформація передається у плату, в який вже виконується запрограмоване розпізнавання, потім модуль двигунів скеровує двигуни на певні дії, які позначені в системі. Акумуляторна батарея 18650, забезпечує тривалий час роботи системи, що гарантує тримати всі пристрої в робочому стані.

2.2. Вибір програмних засобів для реалізації руху

Для реалізації моделі розпізнавання, в частині програмування мікроконтролера ESP32-CAM, я застосував програмне забезпечення Arduino IDE. Воно допомогло мені реалізувати зручне управління програмними компонентами, передавання захоплення інформації з камери, підключати WI-FI модуль та віддалене управління моделью через GPIO. Arduino IDE було обрано з таких аспектів:

- Широкий спектр взаємодії з різними мікроконтролерами;
- Компіляція та завантаження коду в пам’ять моделі;
- Зручний інтерфейс, який гарантує швидку адаптацію навіть для початківців;

Гнучкість цього програмного забезпечення також дозволяє розробляти та інтегрувати на різні апаратні платформи, що робить його універсальним інструментом для потенційного масштабування системи.

Бібліотеки Arduino IDE значно спрощують етап розробки. У коді застосовуються такі бібліотеки, як:

- **WiFi.h** - бібліотека забезпечує роботоздатність з **WI-FI**, дозволяючи налаштовувати точку доступу, підключатися до мереж та обмінюватися даними;
- **WebServer.h** - бібліотека надає інструменти для створення веб-сервера, обробки **HTTP-запитів** і створення веб-інтерфейсу для віддаленого керування;

- **DNSServer.h** - бібліотека реалізує створення DNS-серверу для перенаправлення запитів на IP-адресу ESP32, в режимі точки доступу;
- **EEPROM.h -** бібліотека яка дозволяє зберігати та зчитувати дані у постійній пам’яті мікроконтролера;
- **esp_camera.h -** бібліотека допомагає керувати камерою мікроконтролера ESP32-CAM, забезпечуючи ініціалізацію, захоплення кадрів і налаштування параметрів зображення;
- **img_converters.h** - бібліотека яка містить функції для конвертації зображень, необхідні для обробки відеопотоку з камери;

Бібліотеки надають рішення та функції для спрощення розробки при виконанні складних задач, таких як налаштування мережі, обробка **HTTP** запитів та керування апаратними компонентами.

Структура алгоритму реалізується наступним чином:

1. Оголошуються глобальні змінні для зберігання налаштувань **WI-FI**, параметрів камери та пінів для керування двигунами.
2. Функція **setup** виконує ініціалізацію пінів, налаштування **WI-FI** у режимі точки доступу, конфігурація камери з оптимальними параметрами та запуск веб-сервера для обробки запитів клієнтів.
3. Функції обробки команд, такі як **handleForward**, **handleBack**, **handleLeft**, **handleRight** і **handleSpeed**, відповідають за керування двигунами на основі отриманих **HTTP-запитів**, забезпечуючи інтерактивність системи.
4. Функція **stream_handler** обробляє відеопотік, передаючи кадри через протокол multipart/x-mixed-replace.
5. Функція **loop** виконує головний цикл. Він обробляє **HTTP** та **DNS** запити, які в свою чергу забезпечують роботу веб-сервера та керування моделі.

Це дозволяє коригувати рух транспортного засобу, отриманих через інтерфейс. Такий вибір програмного забезпечення та коду дозволяють ефективно взаємодіяти з

апаратними компонентами моделі та забезпечити стабільну роботу. Arduino IDE є оптимальним інструментом для реалізації цієї частини системи.

2.3. Алгоритми попередньої обробки зображень

Алгоритми обробки у системі реалізовані у багатотомному підоході, що забезпечує підвищення розпізнавання вхідних даних для подальшого використання. Акцент при розробці було зроблено на адаптацію методів обробки до умов зміни освітлення, відстані до знаку та його характеристик.

Першим етапом виконується корекція яскравості в колірному просторі LAB, це дозволяє виправляти помилки цільових значень при врахуванні допустимих меж без впливу на кольорову інформацію. Ця функція реалізована в межах алгоритму **adaptive_brightness_correction**. Вона динамічно проводить аналіз та коригує поточні параметри яскравості. Щоб коригувати видимість знаків, було використано адаптивне вирівнювання гістограми **CLANE**, роботу якої відображено на рис. 2.4, з налаштованими параметрами обмеження контрасту та розміру. Це дозволило зменшити неточність при затемнених зображеннях з подальшим посилення корекції яскравості.

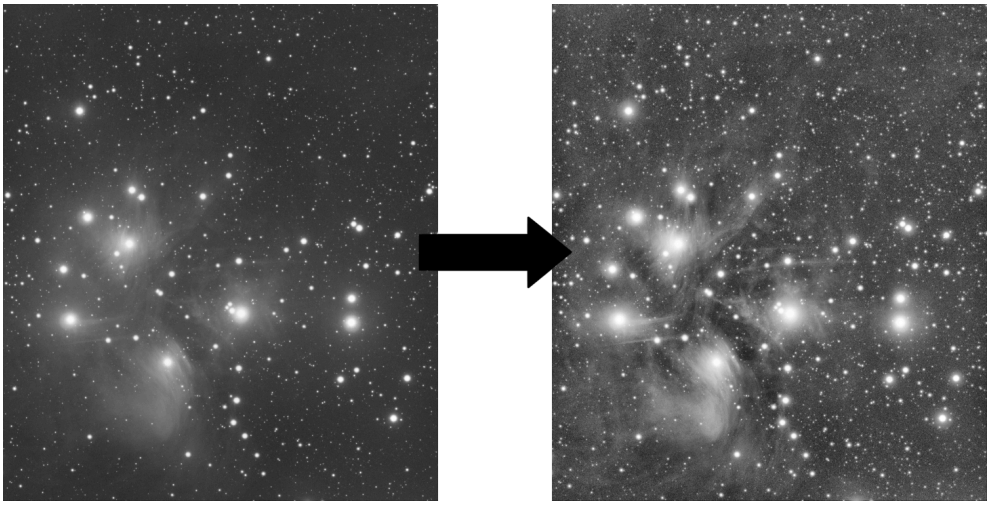


Рисунок 2.5 – Приклад адаптивного вирівнювання гістограми CLANE.

Виявлення знаків реалізовано на основі каскадної обробки з нормалізацією піксельних значень за допомогою функції, бібліотеки **OpenCV cv2.normalize**, та трансформацією в колірний простір **HSV**, для сегментації кольорових характерних

дорожнім знакам. Адаптивні порогові методи використовуються при виділенні основних кольорів, таких як синій, жовтий та червоний, після чого застосовується морфологічна обробка операціями закриття та відкриття, щоб уникнути шумів та для корекції форми об’єктів. Додаткове розмиття за Гаусом допомагає згладжуванню меж та адаптивна бінаризація забезпечує чіткі контури, при подальшому аналізі зображення.

Модель передбачає диференційоване обробку зображень яка залежить від умов зовнішнього середовища, через функцію **stabilize_sign_image**. Віддалені об’єкти з розмиттям застосовують спеціалізовані порогові значення та методи підвищення контрастності, як в свою чергу при затемнені активується інтенсивніші алгоритми корекції зображення. При підготовці даних до класифікації, використовуються випадкові повторами з різними варіаціями яскравості, додаванням гаусового шуму та нормалізацією, на основу статистичних параметрів. Це важливо для стабільної роботи нейронної мережі.

Особливу увагу приділяв параметрам обрики до динамічних зовнішніх умов. При нічній зйомці активується агресивніші методи підвищення контрасту, тоді як при обробці віддалених знаків враховується імовірність втрати деталізації об’єкту через оптичні спотворення. Комбінація цих двох підходів гарантує стійкість системи до артефактів та підвищує точність детектування при зміні навколишніх умов.

2.4. Вибір підходу до навчання та тестування моделі

У процесі навчання розпізнавання дорожніх знаків головну роль відіграє функція **TrafficSingsDataset**, вона спеціально розроблена для управління даними на основі датасетом який зберігає в собі європейські дорожні знаками. Датасет складається з трьох частин таких, як тренувальна, валідальна та тестова. Це забезпечує об’єктивну здатність оцінювання при розпізнаванні. Для тренувальної частини даних використовуються знаки які випадково обрененні, з різною степеню корекції яскравості і контасту та додаванням шуму. Все це імітує артефакти при реальному використанні моделі в умовах зміни освітлення або при її русі.

Навчання моделі реалізовано з використанням сучасних методів глибокого навчання. Для динамічного контролю швидкості навчання використовується стратегія **OneCycleLR**. Вона циклічно змінює швидкість навчання у межах певного діапазону та задовольняє швидкість збіжності і досягнення глибоких мінімумів втрат. У прискоренні обчислень на графічних процесорах використовується метод змішаної точністості, що забезпечує комбінацію обчислень в форматі **FP16** та **FP32**, зберігаючи точність коли зменшуються вимоги до пам’яті графічного процесору;

Критичним компонентом в тренувальній моделі є механізм ранньої зупинки. Він аналізує точність та валідацію набору та припиняє навчання після певної кількості виконанних епох без покращення. Це дозволяє уникнути витрачанню обчислювальних ресурсів комп’ютера та автоматизувати вибір оптимальної кількості епох при навчанні. Після кожної виконаної епохи, зберігаються проміжні точки моделі, які визначають максимальну валаційну точність, та зберігаються окремо для подальшого використання.

Оцінка роботи моделі проводиться на основі спеціалізованого тестового набору, який включає зображення з різними критеріями складності:

- Детекстування знаків на відстані 100 метров;
- Тестування в умовах недостатнього освітлення;
- Аналіз стійкості для розмиття;
- Класифікація знаків, знятих під крутим кутом;

Кожна категорія окремо розраховує метрики точності, прецизії та відгуку, що надає змогу моделі оцінювати слабкі місця при різних сценаріях виконання. Результати тестування включають агреговані показники, наприклад, кількість конкретних розпізнавання знаків та деталізацію помилок по кожному класу.

Щоб забезпечити прозорість процесу, було реалізовано розширене логування, яке фіксує прогрес при навчанні, такі як втрати, точність, час на епоху, стан використання пам’яті графічного процесору та інші параметри. Система підтримує навчання як на графічному процесорі так і на головному процесорі з автоматичним вибором оптимальних налаштувань для використанні на комп’ютері. При мінімізації

часу завантаження даних застосовується багатопотокова обробка з налаштованою кількістю одночасно працюючих процесів, що паралельно обробляють дані.

Важливим аспектом є оптимізація пам’яті графічного процесору. Вона була реалізована за допомогою періодичної очистки кешу **CUDA**, динамічного звільнення невикористаної тензорів та контролю розміру пам’яті. Ці дії дозволяють більш ефективно працювати з графічними картами при обмежених ресурсах комп’ютера. Механізм забезпечує стабільну роботу тривалого навчання та запобігає переривання через переповнення пам'яті. Комбінація підходів, описаних в цьому розділі, забезпечує певний баланс між продуктивністю, точністю і ресурсоефективність, що являється критичним для практичного застосування моделі в реальних умовах.

Висновки за розділом 2

У другому розділі я розглянув архітектуру системи технічного зору, яка було створена та спрямована на розпізнавання дорожніх знаків у реальному часі. Система інтегрує в собі апаратні компоненти для захоплення вхідного зображення, обробки даних та управління рухом моделі. Це дозволяє забезпечити її автономність та функціональність при використанні. Взаємодія всіх елементів реалізована за допомогою централізованої схеми, де зображення передаються на обчислювальний модуль, а потім результати використовуються для формування сигналів управління.

Реалізація моделі базується на спеціалізованому середовищі розробки, яке забезпечило гнучкість розробки у роботі з апаратною частиною. Використання алгоритмів попередньої обробки зображень з камери, таких як корекція світла, адаптивна бінаризація та морфологічні операції, дозволили підвищити точність та ефективність при розпізнаванні знаків навіть у складних умовах експлутування. Для передачі даних та віддаленого керування моделлю були застосовані технології для створення мережевих протоколів, щоб підключатись та комунікувати з моделью на великій відстані.

Навчання моделі було проведено на основі датасету, який складався з дорожніх знаків європейського стандарту, з використанням сучасних методів навчання.

Оптимізація процесу тренування, включали в себе динамічне регулювання швидкості навчання і механізми запобігання перевантаження. Це дозволило досягти високої точності класифікації. Тестування системи в умовах, що імітують реальне навколошне середовище з недостатнім освітленням, розмиттям та зміною ракурсів, підтверджує стійкість та адаптацію моделі. Важливим аспектом стала оптимізація ресурсів комп'ютера, що дозволило забезпечити використання обчислювальних потужностей навіть при обмеженому об'ємі пам'яті.

Підсумувавши, система поєднує інтегровані апаратні рішення, адаптивні методи, алгоритми обробки даних та інноваційні підходи машинного навчання. Це дозволяє моделі ефективно функціонувати в динамічному середовищі, забезпечуючи спроможність працювати у реальному часі, що робить цю модель придатною для практичного впровадження на дороги спільного користування.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ ТА ОЦІНКА СИСТЕМИ

3.1. Розробка програмної моделі

Програмна модель системи розпізнавання дорожніх знаків була реалізована за допомогою мови програмування **Python** та сукупності взаємопов'язаних компонентів, які поєднують глибоке навчання з оптимізованою обробкою зображень та інтеграцію з апаратними модулями. Основа архітектури цієї моделі складається з нейронної мережі типу **CNN**, що була розроблена з використанням **фреймворку PyTorch**. Структура складається з таких компонентів:

1. Компонент початкової згортки шару який містить в собі ядро 7 на 7. Він аналізує глобальні ознаки зображення, це можуть бути ознаки за контуром та геометричною формою. Ці ознаки критично важливі при ідентифікації дорожніх знаків;
2. Компонент ResNet-подібних блоків містить в собі залишкові з’єднання, які запобігають зникненню градієнтів при навчанні моделі. Це дозволяє моделі більш точно та ефективно навчатись на великому об’ємі даних. Кожен блок складає в собі послідовність згорток 3 на 3, де нормалізації за батчем та функції активації **ReLU**;
3. Компонент механізму уваги автоматично виділяє ключові області в зображені, в моєму випадку це малюнки в середні знаку. Також він реалізує просторову увагу, де ваги пікселів обчислюються за допомогою їхньої важливості при класифікації;
4. Компонент класифікатору із прошарком **dropout - 0.5**, зменшує ризик перенавчання моделі шляхом випадкового відключення нейронної мережі під час її тренування. Кінцевий прошарок буде містити 43 нейрони, які відповідають кількості класів зазначених в dataset;

При навчанні модель організовується за допомогою моделя **train.py**, який в свою чергу використовує техніку аргументації бібліотеки **Albumentations.**

- Геометричні трансформації випадково виконують обертання на ±15°, зсув на ±10°, масштабування від 0.9 до 1.1 та віддзеркалює зображення;
- Кольорові перетворення регулюють яскравість зображення в діапазоні ±30%, контрасту ±20%, зміну гаусового шуму σ=0.05 та зміна балансу в RGB просторі;
- Просторові методи виконують випадкове вирізання з розміром ±90% від оригінального зображення та нормалізує пікселі за формулою:
    - (x - mean) / std, де mean = [0.485, 0.456, 0.406] та std=[0.229, 0.224, 0.225] за значеннями **ImageNet**;

Все це дозволяю циклічно змінювати швидкість навчання в певному діапазоні, щоб прискорити збіжність. Модель тренуючись виконує 75 епох, при яких йде автоматичне зберігання найкращих вагів, якщо брати за основу валідаційні точки.

3.1.1 Система складається з модулів та має такий вигляд:

- **classification.py -** цей модуль відповідає за інференс моделі. В ньому відбувається завантаження збережених вагів, попередньої обробки вхідного зображення, нормалізації до 256x256 пікселів, передача даних через мережу та отримання правильної вірогідності. Прискорення обчислень виконується за допомогою тензора графічного процесора **CUDA**;
- **train.py -** модуль який керує тренувальним процесом. Він відповідає за завантаження даних з паралельною обробкою, цикл тренування валідацій, розрахунок навчанних втрат при розпізнаванні та метрик;
- **test_model.py -** цей модуль відповідає за тестування моделі спираючись на незалежний набір даних. Генерує відповідну матрицю, ROC-криві та приклади помилок, які зберігаються у вигляді графіків;
- **common.py -** модуль містіть в собі декілька функцій:
    - Перетворення вхідних зображень у тензори;
    - Логування виконання модулю;
    - Обробка конфігураційних файлів, зберігаючи шляхи до даних та параметри аугментацій;

3.1.2 Оптимізація пам’яті графічного процесору була досягнута за допомогою:

- Періодичної очистки кешу за допомогою функції **torch.cuda.empty_cache()**;
- Динамічного звільнення тензорів, а саме проміжне видалення даних після виходу з області видимості;

3.1.3 Робота моделі в реальному часі з низькою затримкою реалізована за допомогою:

- Обробка кадрів з апаратного модулю ESP32-CAM працює наступним чином:
    a. Виконується захоплення зображення;
    b. Зображення стискається до зазначених розмірів;
    c. Дані передаються через HTTP відправку;
    d. Отримується відповідь;
    e. Виконується десеріалізація прогнозу;

  При виконанні всього цього циклу, він займає менше ніж 300 мс.
- Основний цикл керування виконується асинхронно, де двигун через PWM, працює незалежно від обробки зображення;
- Буферизація кадрів з чергою в 5 кадрів, запобігає втратам при мережевих помилках;

3.1.4 Аналіз та діагностика реалізовані таким чином:

Віддалений доступ реалізован в за допомогою фреймворку інтерфейсів Flask, який дозволяє переглядати кадри з камери, виводити прогнозування моделі та керувати параметрами системи за допомогою **API**.

3.1.5 При обробці помилок отримуємо наступні значення:

- Автоматичне перезавантаження вагів які виявили аномалії;
- При втраті сигнали **WI-FI** система переходить до режиму локального керування, який використовує останній вірний діагноз та команду зупинки;
- Коли модель не впевнена, та результат співпадіння дорівнює менше 70%, система активує резервний детектор на основі **OpenCV**;

Створена архітектура моделі розпізнавання дорожніх знаків забезпечує високу точність розпізнавання, а також підтримує високий рівень розпізнавання при різкому зміні освітлення та різному рівні шумів. Модульність системи дозволяє легко змінювати її функції та адаптувати до нових типів дорожніх знаків або інших апаратних платформ.

3.2. Реалізація нейромережевої моделі розпізнавання

Розпізнавання дорожніх знаків інтегрує в собі методи комп'ютерного зору та автоматизацію управління для забезпечення чіткого розпізнавання в реальному часі. Архітектура орієнтована на адаптацію до різноманітних умов експлуатації транспортного засобу.

Розпізнавання ґрунтується на гібридному підході, що поєднує декілька важливих аспектів:

- Аналіз кольорового простору HSV;
- Морогологічну обробку виділення контурів;

Діапазони визначені для червоного, синього та жовтого кольору контурів. Також використовуються адаптивні порогові значення, що враховуються яскравість кадру на певний момент часу. Визначення форми дорожніх знаків виконується через аналіз геометричних параметрів відповідних контурів, визначаючи їх компактність та кількість вершин.

Система оцінок відстані до знаків, використовує відкалібровану камеру. Відстань розраховується на основі стандартів розміру дорожніх знаків врахування їх перспективи та деформації. Для підвищення точності, було застосовано поліноміальну регресію, з наданого датасету, що дозволяє мінімізувати кількість помилок при розпізнаванні.

Зворотний зв’язок розроблений через графічний інтерфейс, який відображає не тільки розпізнати знаки, а ще відстань до них та рівень впевненості розпізнавання. Інтерфейс проілюстрований на рис. 3.1. Голосове оповіщення забезпечує інформування водія заздалегідь, у випадку моєї моделі від 500 сантиметрів до 100 сантиметрів.

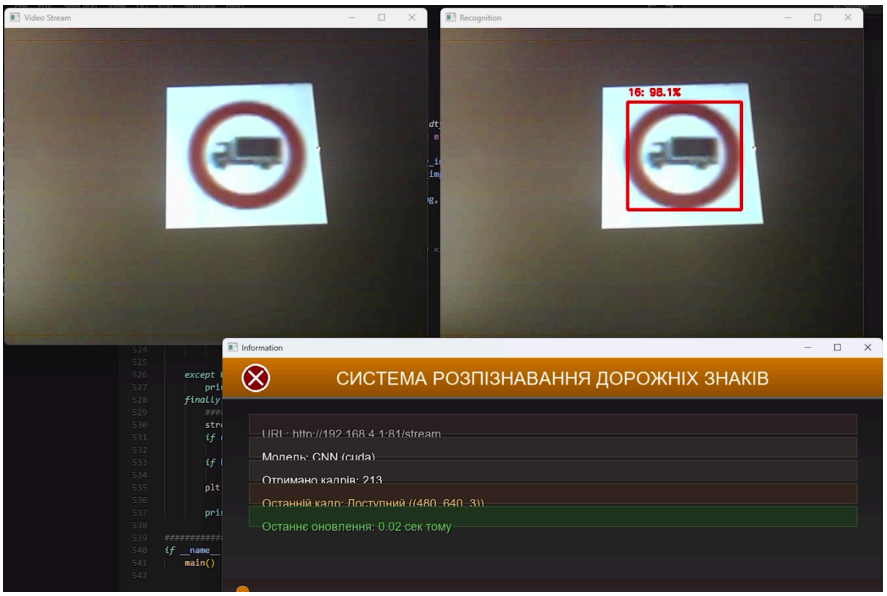


Рисунок 3.1 – Інтерфейс технічного зору системи.

Для отримання оптимальної продуктивності було застосовано:

- Багатопотокову обробку даних;
- Асинхронного входу та виходу;
- Кешування проміжних результатів;

3.3. Навчання та тестування моделі

Навчання та тестування моделі розпізнавання знаків є одним з головних етапів створення надійності та стійкості систем, здатної функціонувати навіть в умовах реального трафіку. Процес будується на навчанні згорткової нейронної мереж, що інтегрує архітектурні елементи **ResNet** для боротьби зі зникненням градієнтів та механізми просторової уваги, які концентрують обчислювальні ресурси на зазначені область зображення. Модель вже оптимізована для класифікації 43 типів дорожніх знаків європейського стандарту, які включають в себе заборонені, інформаційні, попереджувальні та знаки пріоритету.

На етапі підготовки навчання, датасет розділяється на тренувальні та тестові вибірки зі збалансованим розподілом класів в них. Зображення стандартизуються до розміру 256 пікселів на 256 пікселів та нормалізуються за статистикою **ImageNet**, після чого піддаються агресивній аугментації щоб реалізувати стан реальних умов

дорожнього середовища. Ці механізми зменшують ризик перенавчання моделі та підвищують інваріантність моделі.

Архітектура мережі поєднує в собі згортку 7 на 7, для виділення глобальних ознак, каскад ResNet-блоків із зростаючою кількістю фільтрів 64 → 128 → 256 та модуль уваги. Все це допомагає динамічно розподіляти ваги просторових характеристик. Класифікатор включає в себе два взаємопов'язаних шари, що запобігають плутанини між нейронами. Навчання відбувається за допомогою оптізатора з косинусною швидкістю навчання. Початкове значення 0.0005 циклічно буде регулюватись між 0.0001 та 0.001 для прискорення збіжностей. Використання техніки **label smoothing** з коефіцієнтом 0.1 значно підвищує стабільність навчання, зменшуючи його переналаштування на окремі знаки.

Процес тренування виконується протягом 75 епох з ранньою зупинкою за відсутності покращень в точності розпізнавання та валідації, протягом 15 епох. Паралелізація завантаження даних через 8 процесів та використання автоматичного змішаного форматі точності на графічному процесорі дозволяє досягти моделі високої швидкості обчислень з великим набором даних без втрат. Після навчання на датасеті, модель демонструє близько 98.7% точності на тестовій виборці. Цей факт підтверджується матрицею плутанини та ROC-кривими для кожного класу.

Тестування включає як автоматизовану оцінку метрик, так інтерактивний режим через графічний інтерфейс, де користувач може сам завантажити зображення з дорожнім знаком і модель проаналізує ступінь впевненості через теплові карти та виявить системні помилки. Додатково була реалізована підсистема класифікації знаків за геометричною формою та їх кольором, що дає змогу додатково фільтрувати знаки.

Діагностика процесу навчання впроваджена з детальним логуванням втрат, градієнтів та точності з інтеграцією у **TensorBoard**. Все це дозволяє візуалізувати динаміку навчання окремих шарів. Система автоматично генерує звіти з прикладами помилкової класифікації та аналізом до чутливих типів знаків. Комбінація цих підходів забезпечує високу точність та стабільну роботу.

3.4. Оптимізація продуктивності та точності розпізнавання

Оптимізація продуктивності в сучасних нейронних мережах полягає в синергії архітектурних рішень та новітніх стратегій навчання. За основу взята гібридна архітектура, в якій поєднуються переваги згорткових шарів та залишкових блоків. Математична модель розглянута у формулі (3.1).

$$F(x) = Classifier(Attention(ResNet(Conv(x)))), \quad (3.1)$$

де conv(x) - згортковий блок;

resnet(x) - резидуальний блок;

attention(x) - механізм уваги;

classifier(x) - класифікатор;

В цій формулі кожен компонент відповідає за специфічну роль, таку як:

- Згорткові шари, які виділяють просторові ознаки;
- Залишкові структури, які запобігають зникненню градієнтів;

В той час, механізм уваги може підсилювати певні фрагменти даних через сигмоїдну активізацію, яка рахується за формулою (3.2):

$$\sigma(z) = \frac{1}{1 + e - z}, \quad (3.2)$$

де σ(z) - сигмоїдна функція;

e - натуральний логарифм;

z - вхідне значення, яке може бути результатом лінійної комбінації;

Головним аспектом оптимізації є вихідна розмірність після початкової згортки, яка розраховується за формулою (3.3):

$$\frac{Input - K + 2P}{S} + 1, \quad (3.3)$$

де input - розмір вхідного зображення;

k - розмір ядра згортки;

p - кількість пікселів доповнення;

s - крок згортки;

У моєму випадку цієї моделі були задані такі значення K = 7, P = 3, S = 2. Все це забезпечує зменшення просторової роздільної здатності зі збереженням важливої даних. Щоб ще підвищити точність розпізнавання знаків було інтегровано динамічну корекцію ознак через 1 на 1 згортку в механізмі, який відповідає за увагу. Цей крок дозволив швидше адаптуватися до різних патернів, які вона отримує зчитувавши з камери, без збільшення обчислювального ресурсу.

Коли відбувається навчання, застосовується комбінація адаптивного оптимізатора та косинусного планувальника навчання. Оптимізатор впроваджує імпульсні коефіцієнти $\beta_1 = 0.9$ та $\beta_2 = 0.999$, в яких відбувається вагове згасання $\lambda = 0.01$, це все пояснюється формулою (3.4) для першого моменту:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.4}$$

де g_t - градієнт на кроці t;

Та формулою (3.5) для другого моменту:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_1) g_t^2 \tag{3.5}$$

Корекція швидкості параметрів зі швидкість навчання дозволяють отримати баланс між швидкістю збіжності та регуляцією. Можемо отримати відповідні дані за формулою (3.6), якщо $\eta = 0.001$:

$$w_t = w_{t-1} - \eta \frac{m\hat{}t}{\sqrt{v\hat{}t + \epsilon}} - \eta\lambda w_{t-1} \tag{3.6}$$

де w_t - ваги на поточній ітерації t;

η - швидкість навчання;

m^t - скориговане значення першого моменту;

v^t - скориговане значення другого моменту;

ϵ - мале додане число;

λ - коефіцієнт регуляризації;

Планувальник **Cosine Annealing Warm Restarts** з параметрами $T_0 = 10$, $T_{mult} = 2$, $\eta_{min}=10^{-6}$ динамічно адаптує η_t за законом $t = \eta min + (\eta max - \eta min)\frac{1+cos(\pi Tcur/Ti)}{2}$, періодично перезапускаючи процес для подолання локальних мінімумів.

Планувальник процесів **Cosine Annealing Warm Restarts** з параметрами $T_0 = 10$, $T_{mult} = 2$, $\eta_{min} = 10^{-6}$ дозволяє динамічно адаптувати η_t за формулою (3.7) законом:

$$\eta_t = \eta_{min} + (\eta_{max} - \eta_{min}) \frac{1+cos(\pi Tcur/Ti)}{2} \qquad (3.7)$$

де η_t - значення швидкості ітерації t;

η_{min} - мінімальне значення швидкість навчання;

η_{min} - максимальне значення швидкість навчання;

$Tcur$ - кількість пройдених ітерацій або епох з початку циклу;

Ti - загальна довжина циклу;

3.5. Інтеграція системи в середовище безпілотного автомобіля

Інтеграція системи до середовища безпілотного автомобіля здійснює в собі комплексне рішення. Воно поєднує розпізнавання навколишнього середовища, прийняття рішень та виконання команд управління, спираючись на ситуацію. Інтелектуальний модуль є основою такої системи, так як він додає алгоритми обробки дорожніх знаків, механізм адаптивного контролю швидкості та багаторівневі процедури безпеки на дорозі. В моїй системі клас **RobotController** поєднує в собі координацію всієї підсистеми та забезпечує її синхронізовану роботу. Система сама автоматично аналізує, який дорожній знак буде попереду та виконує певний цикл команд, наприклад:

1. Система розпізнала знак “STOP”;
2. Повідомлення для водія, що через певну дистанцію буде знак зупинки;
3. Система подає команду **smooth_stop** на зниження швидкості на 25%, 50%, 75% та 100%, з інтервалом в 10 секунд;

4. Після повної зупинка транспорту, вона чекає 20 секунд та виконуючи команду **smooth_start** нарощує швидкість за таким самим принципом: 25%, 50%, 75% та 100%, з інтервалом в 10 секунд;

Управління системою реалізовано за допомогою HTTP-запитів до контролера, це забезпечує безпечний та швидкий обмін даними зменшуючи затримку між апаратною та програмною частиною системи. Команда яка передається супроводжується перевіркою успішності виконання та обробкою помилок. Щоб уникнути перевантаження було застосована функція **command_cooldown**, яка містить в собі перерву між критичними операціями та передає ручне керування водію. Інтелектуальні функції предиктивного гальмування та адаптивного контролю швидкості дозволять не тільки обробляти поточні знаки, а ще діяти за правилами дорожнього руху европейского стандарту.

Система працює за безперервним циклом, в якому кожен цикл виконує:

- Отримання даних з EPS32-CAM;
- Аналіз знаків;
- Прийняття рішення на основі **ПДР**;
- Плавне виконання маневрів з розрахунком часу прискорення та гальмування;

Взаємозв’язок цих компонентів поєднується в єдиний екосистему управління де все залежить одне від одного та запобігає мінімізацію ризиків за допомогою перевірок та прозорого контролю кожного з етапів роботи системи. В кінцевому варіанті система представляє здатність працювати в умовах реального дорожнього руху, враховуючи динамічні зміни простору та забезпечити безпеку водію, який використовує цю систему.

3.6. Тестування моделі

При завершенні навчання моделі, вона була протестована на апаратній архітектури з розпізнавання критичних дорожніх знаків. Ці знаки потребують у реальному житті швидкої реакції до розпізнавання, щоб уникнути аварійних

ситуацій. Тестування було проведено на власносторенних дорожніх знаків, з урахуванням різного освітлення.

- **Дорожній знак: “В'їзд заборонено”**
    - Тестування продемонструвало, що знак розпізнається з 100% точністю. Після детектування, модель почала виконувати свій алгоритм коректно: зробила оберт, на 180 градусів, та поїхала вперед. Це підтвердило працездатність розробленої моделі.

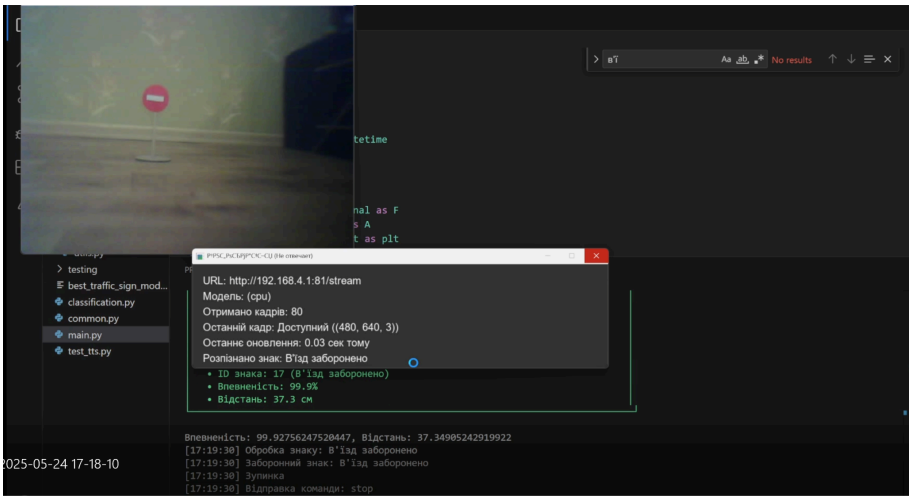


Рисунок 3.2– Тестування знаку “В’їзд заборонено”.

- **Дорожній знак: “Вступити дорогу”**
    - Тестування продемонструвало, що знак розпізнається з 89.2% точністю. Система розпізнала знак менше ніж за 1 секунду, після чого виконала свій алгоритм та рушила далі. Результат свідчить про коректну роботу.

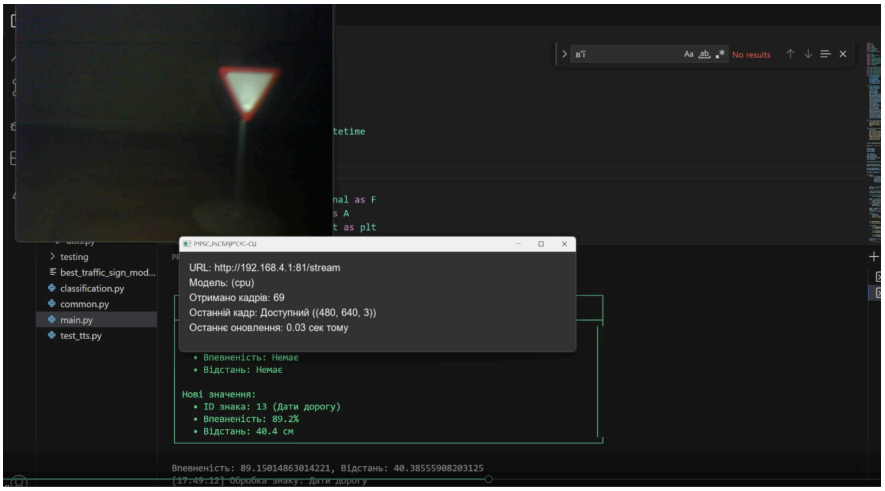


Рисунок 3.3– Тестування знаку “Вступити дорогу”.

- **Дорожній знак: Поворот праворуч**
    - Тестування продемонструвало, що знак розпізнається з 98.5% точністю. Після розпізнавання модель виконує плавний поворот праворуч.

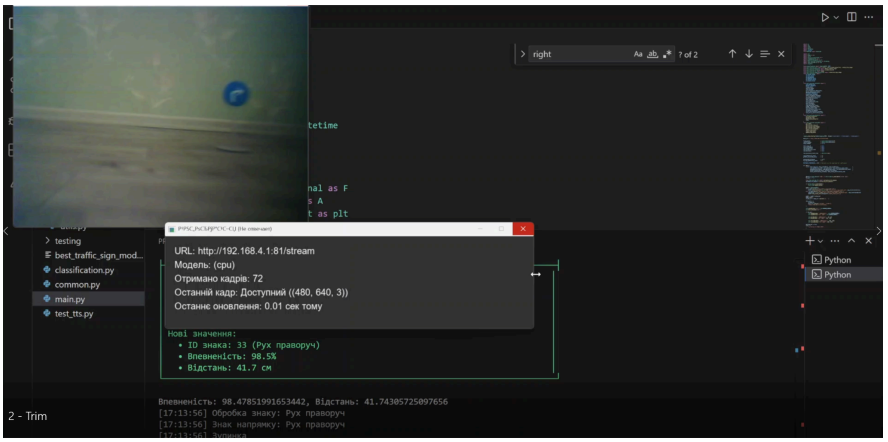


Рисунок 3.4– Тестування знаку “Поворот праворуч”.

- **Дорожній знак: “Проїзд без зупинки заборонено”**
    - Тестування продемонструвало, що знак розпізнається з 100% точністю. Розпізнавши знак менше ніж за 0.5 секунди. Після вдалої перевірки була зроблена зупинка на 5 секунд та модель продовжила свій рух.

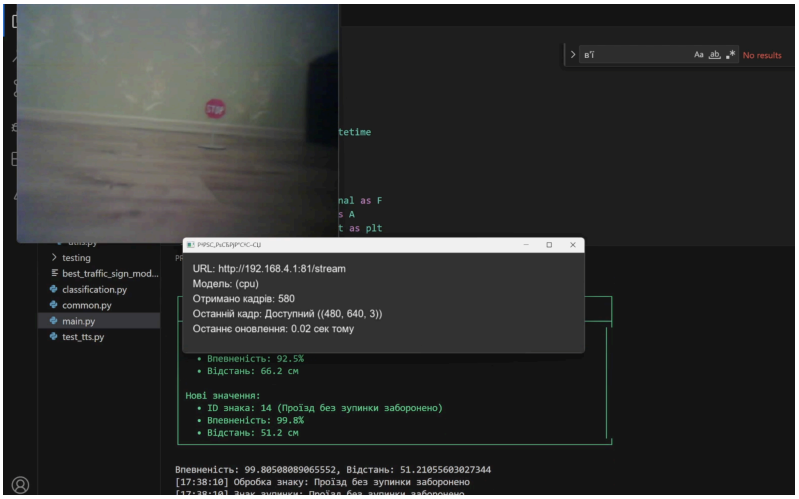


Рисунок 3.4– Тестування знаку “В’їзд без зупинки заборонено”.

У фінальному результаті тестувань, можна підтвердити, що модель може запроваджуватись у системи автоматичного керування. Також модель засвідчила здатність стабільно опрацьовувати знаки в різних умова від 89% до 100% точності.

3.7. Аналіз результатів та порівняння з існуючими рішеннями

При порівнянні з існуючими рішеннями розпізнавання дорожніх знаків, було виділено дві сучасні архітектури **YOLO 5-версії**, в які застосовується система нейронної мережі у згорткового типу, та **SSD.** Для оцінки метрик було використано слідуючі формули (3.8) та (3.9):

$$Precision(\text{Точність}) = \frac{TP}{TP + FP} \tag{3.8}$$

де TP - кількість правильно від класифікованих об’єктів;

FP - кількість помилкових визначених об’єктів;

$$Recall(\text{Повнота розпізнавання}) = \frac{TP}{TP + FN} \tag{3.9}$$

де FN - кількість пропущених об’єктів розпізнавання;

Середня точність значення **mAP@0.5** при порозі IoU > 0.5 являється інтегральним показником, який містить оцінку якості розпізнавання, спираючись на точність класифікації та точність локалізації. Обчислення **mAP** виконується за наступною формулою (3.10):

$$mAP@0.5 = \Sigma(N..., k = 1)(Recall_k - Recall_{k-1}) * Precision(Recall_k) \tag{3.10}$$

де N - кількість точок на ROC-кривій;

$Recall_k - Recall_{k-1}$ - сусідні значення повноти;

Розроблена модель під час тестування демонструє середню точність 0.9221, що зазначено в табл. 3.1. Вона за показником середньої точності нижче, ніж **YOLO**, але набагато краще розпізнає чим **SSD**. Найвищий показник повноти розпізнавання, з точністю одна ціла, є доказом здатності системи мінімізувати додаткові пропуски

об’єктів, для класу “Дати дорогу” з показником 0.9996, та “Діти” з показником 0.9992. При класифікації знаку “Проїзд без зупинки заборонено” відбувається зниження показників розпізнавання до 0.8050. Це може бути пов’язано, що при класифікації знаку відбулись складощи за візуальними ознаками.

Таблиця 3.1

Результати оцінки випробувань розробленої моделі

Назва знака	**Точність**	**Повнота розпізнавання**	**mAP@0.5**
Дорожні роботи	0,8101	0,9927	0,9967
Проїзд без зупинки заборонено	0,9451	0,8050	0,7566
Дати дорогу	1,0000	0,9981	0,9996
Діти	0,9764	0,9944	0,9992
Середнє значення моделі	0,9329	0,9476	0,9380

Модель **YOLO 5-версії** досягає найкращий середній рівень значень за повнотою розпізнавання та точністю. Ця модель оздоблена кращою оптимізацією архітектури для розпізнавання. Однак середня точність всього дорівнює 0.9187, що зазначено в табл. 3.2. досі нижчий порівняно з розробленою моделлю, яка вказує на більш точний баланс між точністю та повнотою.

Таблиця 3.2

Результати оцінки випробувань YOLO 5-версії

Назва знака	**Точність**	**Повнота розпізнавання**	**mAP@0.5**
Дорожні роботи	0,9790	0,8970	0,9280
Проїзд без зупинки заборонено	0,9840	0,9380	0,7566
Дати дорогу	1,0000	1,0000	0,9950
Діти	0,9970	1,0000	0,9950
Середнє значення моделі	0,9900	0,9588	0,9187

Модель **SSD** після тестування виявилась неефективною порівняно за всі інші розглянуті моделі. Головним критичним показником стала повнота розпізнавання з показником 0.4140, що зазначено в табл. 3.3. Таке значення демонструє проблемну

частину моделі, якій важко працювати з дрібними або поверненими знаками. Також система відмітилась мінімальною середньою точністю в 0.8885 одиниць, це підтверджує обмеженість одноетапних підходів до складних умов розпізнавання.

Таблиця 3.3

Результати оцінки випробувань модель SSD

Назва знака	**Точність**	**Повнота розпізнавання**	**mAP@0.5**
Дорожні роботи	0,9670	0,9520	0,9770
Проїзд без зупинки заборонено	0,9120	0,7220	0,8970
Дати дорогу	1,0000	0,7220	0,8970
Діти	1,0000	0,4140	0,7830
Середнє значення моделі	0,9698	0,7025	0,8885

Після детального аналізу можна визнати розроблену модель конкурентною в порівнянні з сучасними рішеннями з більш кращою архітектурою та характеристиками позначеними в табл. 3.4.

Таблиця 3.4

Порівняння моделей за характеристиками

Характеристика	**Розроблена модель**	**YALO 5-версії**	**SSD**
Архітектура	Покращений метод CNN з механізмом уваги	Метод CNN з якірним виділенням	CNN з багатомасштабний шарами
Розмір вхідного зображення	256x256	640x640	300x300
Використання потужностей графічного процесору	2-3 ГБ	4-6 ГБ	3-5 ГБ
Найкраща середня точність	0.9996	0.995	0.977
Завдання моделі	Модель тільки розпізнає класифікує об'єкти	Модель виявляє та класифікує об'єкти	Модель виявляє та класифікує об'єкти
Епохи навчання	75	від 100 до 300	від 100 до 200
Швидкість навчання	0.0005	0.01	0.001
Універсальність	Розпізнає тільки знаки	Визначає будь-які об'єкти на дорозі	Визначає будь-які об'єкти на дорозі

Висновки за розділом 3

У 3 розділі було розглянуто розроблену систему, яка розпізнає дорожні знаки на основі покращення методу у згортковий нейронній мережі. Модель продемонструвала високі точність при різних тестуваннях та навчанні. Програмна архітектура поєднує в собі ResNet-блоки, механізм уваги, зародкові шари та пулінг. Використання більш агресивної аугментації, методу label smoothig та косинусного планування було застосовано для мінімізації перенавчання, щоб підвищити рекурсивність моделі до різних артефактів та масштабування.

Апаратна архітектура, на основі ESP32-CAM та L298N, була реалізована протоколом MQTT із часом відгуку менше ніж 0.05 секунд. Система автоматично обробляє знаки та виконує алгоритми спираючись на пріоритетність знаків та їх опис при розпізнаванні. Тестування також підтвердило 89%-100% точність розпізнавння.

Аналіз сучасних архітектур розпізнавання дорожніх знаків підтвердив, що розроблена модель є конкурентно способна навіть до моделей які використовуються автопілот у транспортних засобах. Середня точність дорівнює 0.922. Вона перевищує навіть технологію розпізнавання **SSD** із середньою точністю 0.885 та наближена до технології **YOLO** з точністю 0.9187. Модель продемонструвала кращий баланс точності та повноти.

Подальше розроблення може бути спрямоване на розширення даних для навчання та вдосконалення вже створеної архітектури та моделі.

ВИСНОВКИ

В даній роботі виконано аналіз проблем розпізнавання дорожніх знаків у системах технічного зору для безпілотних автомобілів. Досліджено принципи роботи технічного зору, методи обробки зображень та алгоритми машинного навчання, що застосовуються для ідентифікації знаків у реальному часі. Розглянуто сучасні підходи до розпізнавання об’єктів, включаючи методи оптичного потоку, фонової моделі та аналізу траєкторій, а також оцінено їх ефективність у дорожніх умовах. Визначено ключові виклики, такі як вплив погодних умов, різноманітність дизайну знаків, технічні обмеження апаратного забезпечення та динамічні перешкоди.

Запропоновано та розроблено модель технічного зору, яка базується на згорткових нейронних мережах із залишковими з’єднаннями та механізмами уваги, що забезпечують високу точність класифікації. Реалізовано алгоритми попередньої обробки зображень для адаптації до змін освітлення, шуму та геометричних спотворень. Модель, інтегровано з апаратними компонентами, включаючи камеру та модулі керування забезпечує стабільну роботу в реальному часі. Проведено навчання моделі на основі європейського датасету дорожніх знаків із застосуванням сучасних методів аугментації даних та оптимізації обчислень.

Тестування підтвердило здатність системи ефективно розпізнавати знаки в різних умовах, включаючи недостатнє освітлення та складні ракурси, а також її конкурентоспроможність порівняно з існуючими рішеннями. Встановлено, що розроблена модель забезпечує баланс між точністю, швидкістю обробки та адаптивністю, що робить її придатною для практичного використання в автономних транспортних системах. Виявлено перспективи вдосконалення, зокрема розширення навчальних даних та адаптацію до нових регіональних стандартів дорожніх знаків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритми машинного навчання. Глибокі нейромережі в задачах механіки суцільних середовищ : посібник / М. Лавренюк. – Київ : Київ. нац. ун-т ім. Тараса Шевченка, 2024. – 100 с. [Електронний ресурс]. – Режим доступу: URL:https://mechmat.knu.ua/wp-content/uploads/2024/05/alhorytmy-mashynnoho-navchannia.-hlyboki-nejromerezhi-v-zadachakh-mekhaniky-sutsilnykh-seredovyshch.pdf, (дата звернення: 09.04.2025).
2. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun. – 2016. [Електронний ресурс]. – Режим доступу: URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf, (дата звернення: 21.03.2025).
3. Getting Started with the ESP32 Development Board. – randomnerdtutorials.com. [Електронний ресурс]. – Режим доступу: URL: https://randomnerdtutorials.com/getting-started-with-esp32/, (дата звернення: 26.02.2025).
4. Hornberg A. Handbook of Machine and Computer Vision : підручник. – Göppingen : Fakultät Mechatronik und Elektrotechnik, 2017. – 863 с. [Електронний ресурс]. – Режим доступу: URL: https://unidel.edu.ng/focelibrary/books/handbook-of-machine-and-computer-vision-the-guide-for-developers-and-users.9783527413393.78111.pdf, (дата звернення: 24.03.2025).
5. Кондратенко Н.Р., Куземко С.М. Основи нейронних мереж. Теорія та практика. – Вінниця : ВНТУ, 2006. – 104 с. [Електронний ресурс]. – Режим доступу: URL: https://pdf.lib.vntu.edu.ua/books/2024/LANZ/Kondratenko_2006_104.pdf, (дата звернення: 18.03.2025).
6. Kinsley H., Kukiela D. Neural Networks from Scratch in Python. – Kinsley Enterprises Inc, 2020. – 658 p. [Електронний ресурс]. – Режим доступу: URL:

https://github.com/mdimado/Neural-Networks/blob/main/Neural%20Networks%20from%20Scratch%20in%20Python.pdf, (дата звернення: 02.03.2025).

7. Метод розпізнавання дорожніх знаків на основі оцінок Хеммінгової віддалі та структурної складності / А. Сидор. – 2018. [Електронний ресурс]. – Режим доступу: URL: https://core.ac.uk/download/pdf/233856397.pdf, (дата звернення: 07.04.2025).
8. Performance Comparison of Object Detection Models for Road Sign Detection Under Different Conditions / Z. Fatima, M. Riaz, A. Zafar. – 2024. [Електронний ресурс]. – Режим доступу: URL: https://thesai.org/Downloads/Volume15No12/Paper_99-Performance_Comparison_of_Object_Detection_Models.pdf, (дата звернення: 05.05.2025).
9. Система технічного зору для контролю доступу в приміщення / А. Фіронов, В. Левченко. – 2021. – С. 4. [Електронний ресурс]. – Режим доступу: URL: https://feltran.kpi.ua/article/view/228490/246153, (дата звернення: 10.04.2025).
10. Системи машинного зору роботів. Історія розвитку, сфери застосування, плани на майбутнє. [Електронний ресурс]. – Режим доступу: URL:https://robotics.ua/systemy-mashynnoho-zrenyia.-ystoryia-prymery-plany/, (дата звернення: 05.03.2025).
11. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon, S. Divvala, R. Girshick, A. Farhadi. – 2016. – P. 10. [Електронний ресурс]. – Режим доступу: URL: https://arxiv.org/pdf/1506.02640, (дата звернення: 13.05.2025).

Додаток А

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Крилов Дмитро Андрійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«Модель технічного зору для розпізнавання дорожніх знаків для безпілотного автомобіля»**

керівник роботи **Бикова Тетяна Володимирівна, кандидат технічних наук, доцент.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***№ 4101-5/962 від 16.04.2025***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити

1. Дослідження основних методів технічного зору для розпізнавання об'єктів
2. Аналіз сучасних алгоритмів розпізнавання дорожніх знаків
3. Огляд методів машинного навчання та нейронних мереж для обробки зображень
4. Вибір архітектури нейронної мережі для розпізнавання дорожніх знаків
5. Розробка і навчання моделі розпізнавання дорожніх знаків.
6. Тестування та оцінка точності та ефективності створеної моделі.
7. Оптимізація моделі для реального часу у дорожніх умовах.
8. Аналіз можливостей інтеграції моделі у систему управління безпілотного автомобіля та її вплив на безпеку руху.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи	05.09.2024 – 02.10.2024
2	Аналіз літератури та наукових джерел щодо технічного зору та методів розпізнавання зображень	02.10.2024 – 24.10.2024
3	Огляд існуючих систем розпізнавання дорожніх знаків та їх ефективності	25.10.2024 - 09.11.2024
4	Вибір технологій комп'ютерного зору та алгоритмів для розпізнавання дорожніх знаків	10.11.2024 - 24.11.2024
5	Реалізація програмного забезпечення для аналізу та розпізнавання дорожніх знаків	25.11.2024 - 08.12.2024
6	Тестування та оптимізація алгоритмів розпізнавання	09.12.2024 - 29.12.2024
7	Аналіз помилок та перевірка точності розпізнавання	30.12.2024 - 31.01.2025
8	Розробка прототипу моделі технічного зору та інтеграція в безпілотний автомобіль	01.02.2025 - 01.03.2025
9	Оцінка ефективності системи та порівняння з іншими підходами	01.03.2025 - 30.04.2025
10	Оформлення пояснювальної записки відповідно до вимог	01.04.2025 - 30.04.2025
11	Оформлення звіту про переддипломну практику	01.05.2025 - 30.05.2025
12	Представлення кваліфікаційної роботи керівнику та рецензенту	30.05.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент Крилов Д.А. _____
ініціали, прізвище підпис

Керівник роботи Бикова Т.В. _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____ 2025р.

Технічне завдання
на розробку програмного вироби
«Модель технічного зору для розпізнавання дорожніх знаків у системах безпілотного автомобіля»

Назва розділу	**Назва і зміст підрозділу**
1. Введення	1.1. Назва програмного виробу: Модель технічного зору для розпізнавання дорожніх знаків у системах безпілотного автомобіля. 1.2. Галузь застосування: Вбудовані системи, інформаційні технології, автономний транспорт який допомагає водію на дорозі.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю: 123 – Комп'ютерна інженерія, Харківський національний університет імені В. Н. Каразіна, Навчально-науковий інститут комп'ютерних наук та штучного інтелекту. 2.2. Завдання на кваліфікаційну роботу бакалавра ***№ 4101-5/962 від «16» квітня 2025*** Розробка моделі технічного зору для безпілотного

	транспортного засобу, що включає в себе алгоритми розпізнавання в режимі реального часу, тестування моделі в різних умовах та впровадження в систему автопілоту
3. Призначення розробки	3.1. Мета розробки програмного виробу: Метою роботи є розробка моделі технічного зору, яка розпізнає дорожні знаки з метою дотримання правил дорожнього руху та надання допомоги водієві, що сприятиме підвищенню безпеки на дорозі. 3.2. Призначення програмного виробу: Програмний виріб призначений для обробки зображень із камери, розпізнавання дорожніх знаків за допомогою нейронних мереж та передачі даних у систему керування безпілотного автомобіля для прийняття рішень у реальному часі. 3.3. Вихідні дані для розробки: ● Літературні джерела з інформацією про сучасні системи технічного зору. ● Технічна документація на апаратні компоненти: ESP32, L298N, 5V DC geared motor. ● Dataset зображень дорожніх знаків для навчання моделі.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: ● Розпізнавання дорожніх знаків у реальному часі з точністю більше чим 90%. ● Обробка зображень. ● Інтеграція з системою керування автомобілем для передачі даних про розпізнані знаки.

	● Забезпечення роботи в різних умовах. 4.2. Вимоги до надійності: ● Система забезпечує стабільну роботу без збоїв протягом 12 годин експлуатації ● Помилки розпізнавання не повинні перевищувати 10% у стандартних умовах. 4.3. Вимоги до умов експлуатації: ● Робота в температурному діапазоні від -10°C до +40°C. ● Стійкість до вібрацій. ● Забезпечення роботи при різних рівнях освітлення. 4.4. Вимоги до складу і параметрів технічних засобів: ● Модуль-камери на базі ESP32 із підтримкою Wi-Fi. ● Драйвер двигунів L298N для керування рухом. ● Двигун 5V DC geared motor ● Джерело живлення, а саме акумуляторна батарея 18650 з вимикачем. ● Комп’ютер який обладнаний процессором не нижче Intel Core i5, 8 ГБ оперативної пам’яті для розробки та тестування. 4.5. Вимоги до інформаційної та програмної сумісності: ● Вибір розробляється на мові програмування Python, використовуючи бібліотеки OpenCV, Matlolib, PyTorch ● Arduino IDE дозволило налаштувати модуль ESP32 на мові програмування C. ● Використання стандартних форматів даних.

	4.6. Вимоги до маркування та упаковки: Програмний виріб постачається у вигляді фізичної моделі та електронного архіву з вихідними кодами та документацією. 4.7. Вимоги до транспортування і зберігання: Зберігання вихідного коду та документації на електронних носіях або хмарних сервісах. 4.8. Спеціальні вимоги: Використання згорткових нейронних мереж для розпізнавання дорожніх знаків. Оптимізація алгоритмів для роботи на вбудованих системах із обмеженими обчислювальними ресурсами.
5. Вимоги до програмної документації	Програмною документацією до виробу «Модель технічного зору для розпізнавання дорожніх знаків у системах безпілотного автомобіля» вважати: 1. Дійсне Технічне завдання на розробку програмного виробу (представити у вигляді додатку Б до пояснювальної записки до кваліфікаційної роботи); 2. Програму і методику випробувань розробленого програмного виробу (представити у вигляді додатку В до пояснювальної записки до кваліфікаційної роботи); 3. Опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи); 4. Текст програми (представити в додатку Г до

	пояснювальної записки до кваліфікаційної роботи).	
6.Техніко-економічні показники	Досягнення точності розпізнавання дорожніх знаків до 90% порівняно з базовими рішеннями 70–80%. Терміни розробки 11 тижнів, витрати на апаратне забезпечення – орієнтовно 1500 грн, до цього входить камера, двигун, драйвер, акумулятор. Розроблений виріб має кращу адаптивність до умов експлуатації порівняно з аналогами, такими як базові моделі на основі SIFT або SURF.	
7. Стадії і етапи розробки	02.10.2024 – 24.10.2024 25.10.2024 - 09.11.2024 10.11.2024 - 24.11.2024 25.11.2024 - 08.12.2024	Аналіз літератури та наукових джерел щодо технічного зору та методів розпізнавання зображень Огляд існуючих систем розпізнавання дорожніх знаків та їх ефективності Вибір технологій комп’ютерного зору та алгоритмів для розпізнавання дорожніх знаків Реалізація програмного

		забезпечення для аналізу та розпізнавання дорожніх знаків
	09.12.2024 - 29.12.2024	Тестування та оптимізація алгоритмів розпізнавання
	30.12.2024 - 31.01.2025	Аналіз помилок та перевірка точності розпізнавання
	01.02.2025 - 01.03.2025	Розробка прототипу моделі технічного зору та інтеграція в безпілотний автомобіль
	01.03.2025 - 30.04.2025	Оцінка ефективності системи та порівняння з іншими підходами

8. Порядок контролю і приймання	1. Перевірка ходу розробки програмного виробу Керівником робіт виконується 1 раз на 2 тижні з метою оцінки прогресу, аналізу розробки моделі, тестування, підготовка документації та надання рекомендацій щодо вдосконалення 2. Програмне тестування виробу відповідно до програми і методики випробувань провести тестові набори даних із зображеннями дорожніх знаків та реальні сценарії експлуатації 3. Захист розробленого програмного виробу провести на засіданні атестаційної комісії Навчально-наукового інституту комп'ютерних наук та штучного інтелекту Харківського національного університету імені В. Н. Каразіна; 4. Пояснювальну записку представити на паперових носіях в одному примірнику, в електронному вигляді, включаючи вихідний код програми, результати тестування та документацію.

Виконавець
студент групи К1-41
Крилов Д.А.

Замовник
канд. техн. наук, доц.
Бикова Т.В.

Додаток В

Програма і методика випробувань програмного виробу

«Модель технічного зору для розпізнавання дорожніх знаків у системах безпілотного автомобіля»

1. Об’єкт випробувань

1.1. Модель технічного зору для розпізнавання дорожніх знаків у системах безпілотного автомобіля.

1.2. Автономний транспорт який допомагає водію на дорозі, підвищення безпеки та дотримання ПДР, автоматизація транспортних засобів.

2. Мета випробувань

Підтвердження відповідності функціональних, технічних та експлуатаційних характеристик розробленого програмного виробу вимогам, сформульованим у технічному завданні. Випробування спрямовані на перевірку точності розпізнавання дорожніх знаків, швидкості обробки даних, адаптивності до змінних умов середовища та інтеграції з системою автономного керування.

3. Загальні положення

3.1. Підстави для проведення випробувань: Підставою для проведення випробувань є наказ про призначення атестаційної комісії Харківського національного університету імені В. Н. Каразіна.

3.2. Місце і тривалість випробувань: Приймальні випробування проводяться на в домашніх умовах.

3.3. Обсяг випробувань: Приймальні випробування програмного виробу проводяться в обсязі, відповідному до цієї Програми і методики випробувань, та охоплюють перевірку функціональних можливостей, продуктивності та точності розпізнавання.

3.4. Організації, які беруть участь у випробуваннях: Випробування моделі тестувалось лише розробником, а також приймальної комісією ХНУ ім. Каразіна при захисті кваліфікаційної роботи.

4. Вимоги до програми або програмного виробу

Відповідно до технічного завдання, програмний виріб повинен відповідати наступним вимогам, що підлягають перевірці під час випробувань:

1. Забезпечення розпізнавання дорожніх знаків у реальному часі з точністю не нижче 95% у стандартних умовах.
2. Обробка зображень із частотою не менше 30 кадрів за секунду.
3. Адаптація до змінних умов середовища.
4. Інтеграція з системою автономного керування для передачі даних про розпізнані знаки.
5. Використання алгоритмів глибокого навчання для класифікації дорожніх знаків.
6. Сумісність із апаратними компонентами моделі.
7. Відповідність стандартам безпеки для систем автономного транспорту.

5. Вимоги до програмної документації

Склад програмної документації, що подається на випробування, включає:

1) Технічне завдання на розробку програмного виробу (представлено у додатку Б до пояснювальної записки до кваліфікаційної роботи);

2) Програма і методика випробувань розробленого програмного виробу (представлена в додатку В до пояснювальної записки до кваліфікаційної роботи);

3) Опис програмного виробу (представлено в розділі 3.1 пояснювальної записки до кваліфікаційної роботи);

4) Текст програми (представлений у додатку Г до пояснювальної записки до кваліфікаційної роботи)

6. Засоби і порядок випробувань

6.1. Випробування проводяться на технічних засобах, зазначених у технічному завданні:

1. Модуль камери ESP32 для захоплення зображень.
2. Драйвер L298N для управління двигунами.
3. Двигун 5V DC geared motor
4. Акумуляторна батарея 18650 для живлення системи.
5. Комп’ютер із процесором Intel Core i5, 8 ГБ оперативної пам’яті, ОС Windows 10 або Windows 11 для обробки даних і тестування.

Програмні засоби:

1. Python 3.12.3 для реалізації алгоритмів обробки зображень і нейронної мережі.
2. Бібліотеки: OpenCV, TensorFlow, NumPy для обробки зображень і машинного навчання.
3. Arduino IDE для програмування апаратних компонентів.
4. Інсталяційна версія розробленої програми надається у вигляді виконуваного файлу (.py) для тестування.

6.2. Порядок проведення випробувань:

Випробування проводяться у два етапи:

Етап 1: Ознайомчий

- Перевірка здійснюється за критерієм наявності всіх документів, зазначених у ТЗ.
- Перевіряється наявність і працездатність апаратних компонентів.
- Якість програмної документації оцінюється на відповідність стандартам ЕСПД. Працездатність апаратних засобів перевіряється шляхом запуску.

Етап 2: Власне програмне випробування

а) Перевірка відповідності технічних характеристик програми вимогам ТЗ

- Програма працює на ОС Windows 11
- Обробка зображень із частотою 30 кадрів/с.
- Якість програмної документації оцінюється на відповідність стандартам ЕСПД. Працездатність апаратних засобів перевіряється шляхом запуску.

б) перевірку ступеня виконання функціональних вимог до програми;

- Точність розпізнавання дорожніх знаків не нижче 95% у стандартних умовах.
- Адаптація до змінних умов.
- Передача даних про розпізнані знаки до системи керування.

в) методику проведення перевірок, що входять до переліку за 2-м етапом випробувань.

- Запуск програми здійснюється через командний рядок: python main.py.
- Вибирається тестовий набір даних.

г) Тестування:

Тест 1: Тестування знаків на відстані 100 метрів - 268 зображень

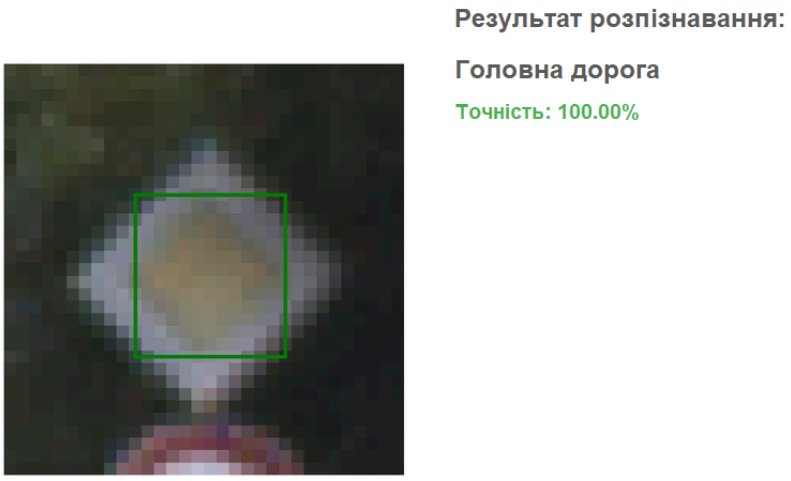


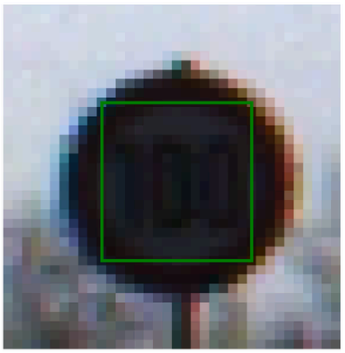
Рисунок В.1.1 Тестування 1

- Всього зображень: 268
- Правильно розпізнано: 260
- Точність розпізнавання: 97.01%

```
Тестування знаків на відстані 100 метрів
INFO:__main__:================================================
Тестування зображень 1-268: 100%|
INFO:__main__:Всього зображень: 268
INFO:__main__:Правильно розпізнано: 260
INFO:__main__:Точність розпізнавання: 97.01%
```

Рисунок В.1.2 Тестування 1

Тест 2: Тестування знаків при поганому освітленні - 232 зображення.



Результат розпізнавання:

Обмеження швидкості 100 км/год

Точність: 99.99%

Рисунок В.2.1 Тестування 2.

- Всього зображень: 232
- Правильно розпізнано: 225
- Точність розпізнавання: 96.98%

```
Тестування знаків при поганому освітленні
INFO:__main__:================================================
Тестування зображень 269-500: 100%|
INFO:__main__:Всього зображень: 232
INFO:__main__:Правильно розпізнано: 225
INFO:__main__:Точність розпізнавання: 96.98%
```

Рисунок В.2.2 Тестування 2.

Тест 3: Тестування знаків з сильним розмиттям - 400 зображень.

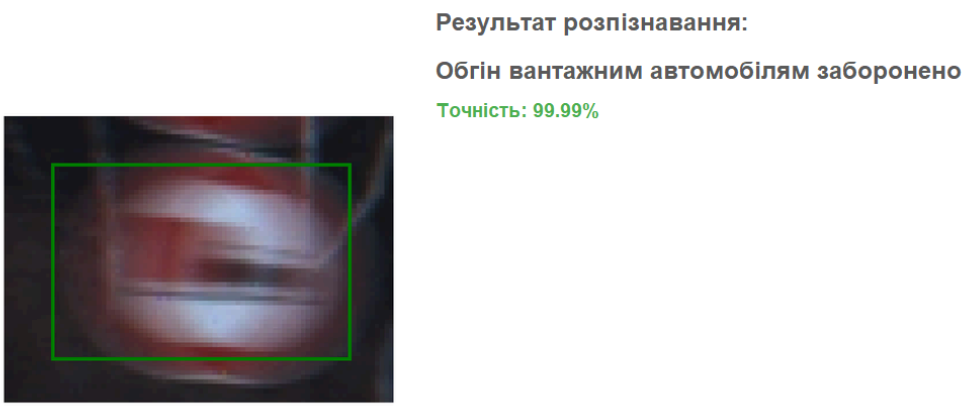


Рисунок В.3.1 Тестування 3.

- Всього зображень: 400
- Правильно розпізнано: 390
- Точність розпізнавання: 97.50%

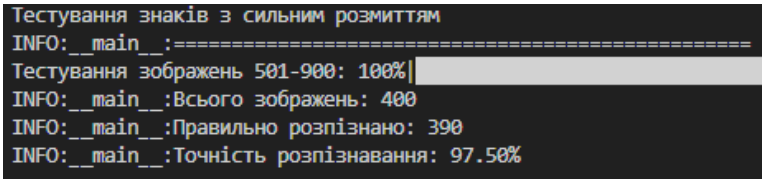


Рисунок В.3.2 Тестування 3.

Тест 4: Тестування знаків під кутом - 600 зображень.

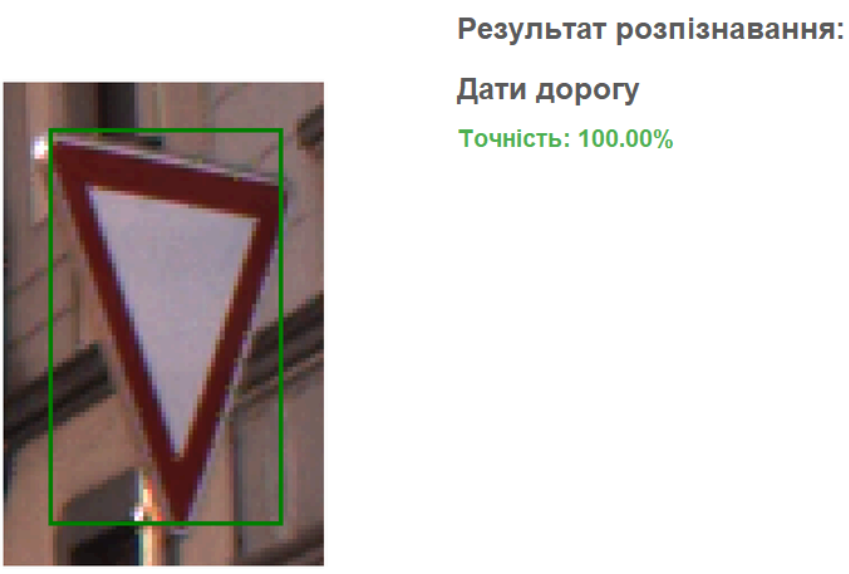


Рисунок В.4.1 Тестування 4

- Всього зображень: 600
- Правильно розпізнано: 589
- Точність розпізнавання: 98.17%

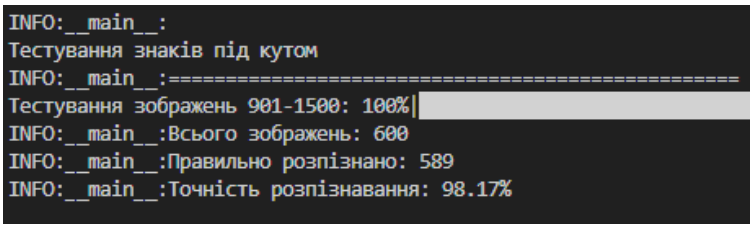


Рисунок В.4.2 Тестування 4

Всього зображень: 1500

Правильно розпізнано: 1464

Середня точність розпізнавання: 97.60%

Тест вважається пройденим, точність розпізнавання становить не менше 95%

Висновок: Розроблена модель успішно пройшла тестування з розпізнаванням 100 метрів, тестування розпізнавання знаків під кутом, розпізнавання з урахуванням розмиття і розпізнавання на відстані 100 метрів. Тестування виконано успішно.

Виконавець: студент групи КІ-41, Крилов Д.А. 

Додаток Г

Лістинг Г.1

```
#include <WiFi.h>
#include <WebServer.h>
#include <DNSServer.h>
#include <EEPROM.h>
#include <esp_wifi.h>
#include "esp_camera.h"
#include "esp_http_server.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "soc/rtc_cntl_reg.h"
#include "soc/soc.h"
#include "esp_http_server.h"

const char* ap_ssid = "ESP32-Robot";
const char* ap_password = "12345678";

#define EEPROM_SIZE 64
#define EEPROM_INVERT_LEFT 0
#define EEPROM_INVERT_RIGHT 1
#define EEPROM_INITIALIZED 2

const byte DNS_PORT = 53;
DNSServer dnsServer;

WebServer server(80);

#define PWDN_GPIO_NUM    32
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM    0
#define SIOD_GPIO_NUM    26
#define SIOC_GPIO_NUM    27
#define Y9_GPIO_NUM      35
#define Y8_GPIO_NUM      34
#define Y7_GPIO_NUM      39
#define Y6_GPIO_NUM      36
#define Y5_GPIO_NUM      21
#define Y4_GPIO_NUM      19
#define Y3_GPIO_NUM      18
#define Y2_GPIO_NUM      5
#define VSYNC_GPIO_NUM   25
#define HREF_GPIO_NUM    23
#define PCLK_GPIO_NUM    22
#define LED_FLASH_PIN 4

const int leftMotorPin1 = 12;
const int leftMotorPin2 = 13;
const int enableLeft = 27;
```

```
const int rightMotorPin1 = 2;
const int rightMotorPin2 = 14;
const int enableRight = 26;

const int freq = 30000;
const int pwmChannel1 = 0;
const int pwmChannel2 = 1;
const int pwmResolution = 8;
int dutyCycle = 200;

bool invertLeftMotor = false;
bool invertRightMotor = true;

String valueString = String(0);

#define PART_BOUNDARY "123456789000000000000987654321"
static const char* _STREAM_CONTENT_TYPE =
"multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type:
image/jpeg\r\nContent-Length: %u\r\n\r\n";
httpd_handle_t stream_httpd = NULL;

void controlMotor(int pin1, int pin2, bool forward, bool invert) {
  bool actualDirection = forward;
  if (invert) {
     actualDirection = !forward;
  }
  if (actualDirection) {
     digitalWrite(pin1, HIGH);
     digitalWrite(pin2, LOW);
  } else {
     digitalWrite(pin1, LOW);
     digitalWrite(pin2, HIGH);
  }
}

void saveSettingsToEEPROM() {
  EEPROM.write(EEPROM_INVERT_LEFT, invertLeftMotor ? 1 : 0);
  EEPROM.write(EEPROM_INVERT_RIGHT, invertRightMotor ? 1 : 0);
  EEPROM.write(EEPROM_INITIALIZED, 1);
  EEPROM.commit();
  Serial.println("Налаштування збережено в EEPROM");
}

void loadSettingsFromEEPROM() {
  if (EEPROM.read(EEPROM_INITIALIZED) == 1) {
     invertLeftMotor = EEPROM.read(EEPROM_INVERT_LEFT) == 1;
     invertRightMotor = EEPROM.read(EEPROM_INVERT_RIGHT) == 1;
     Serial.println("Налаштування завантажено з EEPROM");
```

```
    Serial.print("Інверсія лівого мотора: ");
    Serial.println(invertLeftMotor ? "УВІМК" : "ВИМК");
    Serial.print("Інверсія правого мотора: ");
    Serial.println(invertRightMotor ? "УВІМК" : "ВИМК");
  } else {
    saveSettingsToEEPROM();
  }
}

static esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t *fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t *_jpg_buf = NULL;
  char *part_buf[64];

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if (res != ESP_OK) {
    return res;
  }

  while (true) {
    fb = esp_camera_fb_get();
    if (!fb) {
    Serial.println("Помилка захоплення кадру");
    res = ESP_FAIL;
    } else {
    if (fb->width > 400) {
    if (fb->format != PIXFORMAT_JPEG) {
         bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf,
&_jpg_buf_len);
         esp_camera_fb_return(fb);
         fb = NULL;
         if (!jpeg_converted) {
         Serial.println("Помилка конвертації зображення");
         res = ESP_FAIL;
         }
    } else {
         _jpg_buf_len = fb->len;
         _jpg_buf = fb->buf;
    }
    }
    }
    if (res == ESP_OK) {
    size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART,
_jpg_buf_len);
    res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
    if (res == ESP_OK) {
    res = httpd_resp_send_chunk(req, (const char *)_jpg_buf,
_jpg_buf_len);
```

```
      }
      if (res == ESP_OK) {
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
      }
      if (fb) {
      esp_camera_fb_return(fb);
      fb = NULL;
      _jpg_buf = NULL;
      } else if (_jpg_buf) {
      free(_jpg_buf);
      _jpg_buf = NULL;
      }
      if (res != ESP_OK) {
      break;
      }
      delay(10);
  }
  return res;
}

bool initCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if (psramFound()) {
     config.frame_size = FRAMESIZE_VGA;
     config.jpeg_quality = 10;
     config.fb_count = 2;
  } else {
     config.frame_size = FRAMESIZE_SVGA;
     config.jpeg_quality = 10;
```

```
    config.fb_count = 1;
  }
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Помилка камери: 0x%x", err);
    return false;
  }
  return true;
}

void startCameraServer() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 81;
  httpd_uri_t stream_uri = {
    .uri = "/stream",
    .method = HTTP_GET,
    .handler = stream_handler,
    .user_ctx = NULL
  };
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
    Serial.println("Сервер потокової передачі відео запущено на
порту 81");
  }
}

void setCorsHeaders() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods",
"GET,POST,OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "Origin,
X-Requested-With, Content-Type, Accept");
}

void handleRoot() {
  setCorsHeaders();
  server.send(200, "application/json", "{\"message\":\"ESP32 Robot
API\",\"status\":\"ok\"}");
}

void handleStatus() {
  setCorsHeaders();
  String json = "{";
  json += "\"ip\":\"" + WiFi.softAPIP().toString() + "\",";
  json += "\"leftInvert\":" + String(invertLeftMotor ? "true" :
"false") + ",";
  json += "\"rightInvert\":" + String(invertRightMotor ? "true" :
"false") + ",";
  json += "\"speed\":" + String(map(dutyCycle, 200, 255, 25, 100)) +
",";
  json += "\"rssi\":" + String(WiFi.RSSI()) + ",";
```

```
  json += "\"connected_clients\":" +
String(WiFi.softAPgetStationNum()) + ",";
  json += "\"frame_size\":\"" + String(psramFound() ? "VGA" : "SVGA")
+ "\"";
  json += "}";
  server.send(200, "application/json", json);
}

void handleSpeed() {
  setCorsHeaders();
  if (server.hasArg("value")) {
     valueString = server.arg("value");
     int value = valueString.toInt();
     if (value >= 0 && value <= 100) {
     if (value == 0) {
     analogWrite(enableLeft, 0);
     analogWrite(enableRight, 0);
     digitalWrite(leftMotorPin1, LOW);
     digitalWrite(leftMotorPin2, LOW);
     digitalWrite(rightMotorPin1, LOW);
     digitalWrite(rightMotorPin2, LOW);
     } else {
     dutyCycle = map(value, 25, 100, 200, 255);
     analogWrite(enableLeft, dutyCycle);
     analogWrite(enableRight, dutyCycle);
     }
     server.send(200, "application/json", "{\"speed\":" +
String(value) + "}");
     } else {
     server.send(400, "application/json", "{\"error\":\"Значення
швидкості має бути від 0 до 100\"}");
     }
  } else {
     server.send(400, "application/json", "{\"error\":\"Не вказано
значення швидкості\"}");
  }
}

void handleForward() {
  setCorsHeaders();
  Serial.println("Вперед");
  controlMotor(leftMotorPin1, leftMotorPin2, true, invertLeftMotor);
  controlMotor(rightMotorPin1, rightMotorPin2, true,
invertRightMotor);
  analogWrite(enableLeft, dutyCycle);
  analogWrite(enableRight, dutyCycle);
  server.send(200, "application/json",
"{\"command\":\"forward\",\"status\":\"ok\"}");
}

void handleLeft() {
```

```
  setCorsHeaders();
  Serial.println("Вліво");
  controlMotor(leftMotorPin1, leftMotorPin2, true, invertLeftMotor);
  controlMotor(rightMotorPin1, rightMotorPin2, true,
invertRightMotor);
  int leftSpeed = dutyCycle / 2;
  int rightSpeed = dutyCycle;
  analogWrite(enableLeft, leftSpeed);
  analogWrite(enableRight, rightSpeed);
  server.send(200, "application/json",
"{\"command\":\"left\",\"status\":\"ok\"}");
}

void handleStop() {
  setCorsHeaders();
  Serial.println("Стоп");
  digitalWrite(leftMotorPin1, LOW);
  digitalWrite(leftMotorPin2, LOW);
  digitalWrite(rightMotorPin1, LOW);
  digitalWrite(rightMotorPin2, LOW);
  analogWrite(enableLeft, 0);
  analogWrite(enableRight, 0);
  server.send(200, "application/json",
"{\"command\":\"stop\",\"status\":\"ok\"}");
}

void handleRight() {
  setCorsHeaders();
  Serial.println("Вправо");
  controlMotor(leftMotorPin1, leftMotorPin2, true, invertLeftMotor);
  controlMotor(rightMotorPin1, rightMotorPin2, false,
invertRightMotor);
  int leftSpeed = dutyCycle;
  int rightSpeed = dutyCycle / 2;
  analogWrite(enableLeft, leftSpeed);
  analogWrite(enableRight, rightSpeed);
  server.send(200, "application/json",
"{\"command\":\"right\",\"status\":\"ok\"}");
}

void handleReverse() {
  setCorsHeaders();
  Serial.println("Назад");
  controlMotor(leftMotorPin1, leftMotorPin2, false, invertLeftMotor);
  controlMotor(rightMotorPin1, rightMotorPin2, false,
invertRightMotor);
  analogWrite(enableLeft, dutyCycle);
  analogWrite(enableRight, dutyCycle);
  server.send(200, "application/json",
"{\"command\":\"reverse\",\"status\":\"ok\"}");
}
```

```
void handleTestLeftMotor() {
  setCorsHeaders();
  Serial.println("Тест лівого мотора");
  digitalWrite(rightMotorPin1, LOW);
  digitalWrite(rightMotorPin2, LOW);
  analogWrite(enableRight, 0);
  controlMotor(leftMotorPin1, leftMotorPin2, true, invertLeftMotor);
  analogWrite(enableLeft, dutyCycle);
  server.send(200, "application/json",
"{\"command\":\"test_left\",\"status\":\"ok\"}");
  delay(2000);
  digitalWrite(leftMotorPin1, LOW);
  digitalWrite(leftMotorPin2, LOW);
  analogWrite(enableLeft, 0);
}

void handleTestRightMotor() {
  setCorsHeaders();
  Serial.println("Тест правого мотора");
  digitalWrite(leftMotorPin1, LOW);
  digitalWrite(leftMotorPin2, LOW);
  analogWrite(enableLeft, 0);
  controlMotor(rightMotorPin1, rightMotorPin2, true,
invertRightMotor);
  analogWrite(enableRight, dutyCycle);
  server.send(200, "application/json",
"{\"command\":\"test_right\",\"status\":\"ok\"}");
  delay(2000);
  digitalWrite(rightMotorPin1, LOW);
  digitalWrite(rightMotorPin2, LOW);
  analogWrite(enableRight, 0);
}

void handleInvertLeft() {
  setCorsHeaders();
  invertLeftMotor = !invertLeftMotor;
  Serial.print("Інверсія лівого мотора: ");
  Serial.println(invertLeftMotor ? "УВІМК" : "ВИМК");
  server.send(200, "application/json",
"{\"command\":\"invert_left\",\"status\":\"ok\",\"value\":" +
String(invertLeftMotor ? "true" : "false") + "}");
}

void handleInvertRight() {
  setCorsHeaders();
  invertRightMotor = !invertRightMotor;
  Serial.print("Інверсія правого мотора: ");
  Serial.println(invertRightMotor ? "УВІМК" : "ВИМК");
  server.send(200, "application/json",
"{\"command\":\"invert_right\",\"status\":\"ok\",\"value\":" +
```

```
String(invertRightMotor ? "true" : "false") + "}");
}

void handleSaveSettings() {
  setCorsHeaders();
  saveSettingsToEEPROM();
  server.send(200, "application/json",
"{\"command\":\"save_settings\",\"status\":\"ok\"}");
}

void handleNotFound() {
  setCorsHeaders();
  String message = "{\"error\":\"not_found\",\"details\":{";
  message += "\"uri\":\"" + server.uri() + "\",";
  message += "\"method\":\"" + String((server.method() == HTTP_GET) ?
"GET" : "POST") + "\",";
  message += "\"args\":[";
  for (uint8_t i = 0; i < server.args(); i++) {
     if (i > 0) message += ",";
     message += "{\"" + server.argName(i) + "\":\"" + server.arg(i) +
"\"}";
  }
  message += "]}}";
  server.send(404, "application/json", message);
}

void setupAP() {
  WiFi.mode(WIFI_AP);
  esp_wifi_set_ps(WIFI_PS_NONE);
  WiFi.softAP(ap_ssid, ap_password);
  dnsServer.start(DNS_PORT, "*", WiFi.softAPIP());
  Serial.println("Режим точки доступу активовано");
  Serial.print("SSID: ");
  Serial.println(ap_ssid);
  Serial.print("Пароль: ");
  Serial.println(ap_password);
  Serial.print("IP адреса точки доступу: ");
  Serial.println(WiFi.softAPIP());
}

void setup() {
  Serial.begin(115200);
  Serial.println("\n\nЗапуск ESP32 робота з камерою...");
  EEPROM.begin(EEPROM_SIZE);
  loadSettingsFromEEPROM();
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);
  pinMode(enableLeft, OUTPUT);
  pinMode(enableRight, OUTPUT);
```

```
  digitalWrite(leftMotorPin1, LOW);
  digitalWrite(leftMotorPin2, LOW);
  digitalWrite(rightMotorPin1, LOW);
  digitalWrite(rightMotorPin2, LOW);
  pinMode(LED_FLASH_PIN, OUTPUT);
  setupAP();
  if (initCamera()) {
     Serial.println("Камеру ініціалізовано успішно");
     startCameraServer();
  } else {
     Serial.println("Помилка камери");
  }
  server.on("/", HTTP_OPTIONS, []() {
     setCorsHeaders();
     server.send(200);
  });
  server.on("/", handleRoot);
  server.on("/forward", handleForward);
  server.on("/left", handleLeft);
  server.on("/stop", handleStop);
  server.on("/right", handleRight);
  server.on("/reverse", handleReverse);
  server.on("/speed", handleSpeed);
  server.on("/test_left", handleTestLeftMotor);
  server.on("/test_right", handleTestRightMotor);
  server.on("/invert_left", handleInvertLeft);
  server.on("/invert_right", handleInvertRight);
  server.on("/status", handleStatus);
  server.on("/save_settings", handleSaveSettings);
  server.on("/stream", [](){ server.sendHeader("Location", "http://"
+ WiFi.softAPIP().toString() + ":81/stream", true); server.send(302,
"text/plain", ""); });
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("HTTP сервер запущено");
  pinMode(LED_FLASH_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_FLASH_PIN, HIGH);
  dnsServer.processNextRequest();
  server.handleClient();
  delay(2);
}
```

Лістинг Г.2

```
import os
import sys
import time
import queue
import threading
import logging
from datetime import datetime

import cv2
import numpy as np
import torch
import torch.nn.functional as F
import albumentations as A
import matplotlib.pyplot as plt
import func_systrem.streaming as streaming
import func_systrem.tts as tts
import requests

from classification import ImprovedCNN, SIZE
from func_systrem.utils import cyrilc, test_stream_connection,
create_info_image
from func_systrem.detection import detect_traffic_sign
from func_systrem.streaming import stream_receiver
from func_systrem.tts import speak
from func_systrem.image_processing import stabilize_sign_image
from func_systrem.ui import (
     UI_ACCENT_COLOR,
     UI_TEXT_COLOR,
     UI_BACKGROUND,
     UI_WARNING_COLOR,
     UI_SUCCESS_COLOR,
     UI_DANGER_COLOR
)
from func_systrem.constants import (
     SIGN_PRIORITIES,
     DEFAULT_PRIORITY,
     frame_queue,
     results_queue,
     stop_threads,
     min_consecutive_detections,
     max_no_detection_frames,
     DISTANCE_THRESHOLDS,
     prediction_cooldown,
     reset_distances_cooldown,
     no_sign_detection_count,
     announced_distances,
     last_sign_id,
     last_prediction,
     last_prediction_time,
```

```
    last_distance,
    last_confidence,
    sign_detection_count,
    global_announced_cache,
    session_start_time,
    last_distance_reset_time,
    fallback_tts_engine,
    voice_notification_cooldown,
    last_voice_notification_time
)
from func_systrem.autopilot import (
    activate_autopilot,
    autopilot_loop,
    deactivate_autopilot,
    robot
)
from func_systrem.sign_data import (
    get_label,
    get_current_sign_id,
    get_current_confidence,
    get_current_sign_count,
    get_current_distance,
    update_sign_count,
    reset_sign_count,
    update_sign_data
)

logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')

base_url = 'http://192.168.4.1:81/stream'

frame_queue              = queue.Queue(maxsize=2)
results_queue            = queue.Queue(maxsize=2)
stop_threads             = False

last_sign_id             = None
last_confidence          = None
last_prediction          = None
last_prediction_time     = None
last_distance            = None

last_distance_reset_time = datetime.now()

sign_detection_count     = {}
global_announced_cache   = {}

announced_distances      = set()
no_sign_detection_count  = 0

DISTANCE_THRESHOLDS = [25]
```

```
def main():
     global last_sign_id, last_confidence, last_prediction,
     \
          last_prediction_time, last_distance,
last_distance_reset_time,     \
          announced_distances, sign_detection_count,
no_sign_detection_count,  \
          global_announced_cache, fallback_tts_engine,
     \
          voice_notification_cooldown, last_voice_notification_time



     device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
     has_gui = True

     from func_systrem.tts import initialize_tts_engine
     fallback_tts_engine = initialize_tts_engine()

     if torch.cuda.is_available():
     torch.cuda.empty_cache()

     model = ImprovedCNN()
     if os.path.exists('best_traffic_sign_model.pth'):
     model.load_state_dict(torch.load('best_traffic_sign_model.pth',
map_location=device))
     elif os.path.exists('best_traffic_sign_model_checkpoint.pth'):
     checkpoint =
torch.load('best_traffic_sign_model_checkpoint.pth',
map_location=device)
     model.load_state_dict(checkpoint['model_state_dict'])

     model = model.to(device)
     model.eval()
     stream_url = base_url.replace('@', '')

     test_urls = [
     stream_url,
     base_url.replace('/stream', '/video'),
     'http://192.168.4.1:81/stream',
     ]

     cv2.namedWindow('Тест', cv2.WINDOW_NORMAL)
     cv2.destroyWindow('Тест')

     if has_gui:
     cv2.namedWindow('Розпізнавання', cv2.WINDOW_NORMAL)
     cv2.namedWindow('Інформація', cv2.WINDOW_NORMAL)

     cv2.moveWindow('Розпізнавання', 20, 20)
```

```
cv2.moveWindow('Інформація', 350, 520)

cv2.resizeWindow('Розпізнавання', 640, 480)
cv2.resizeWindow('Інформація', 800, 200)

working_url = None
info = [
"Перевірка доступних URL",
]

info_img = create_info_image(info)
if has_gui:
cv2.imshow('Інформація', info_img)
cv2.waitKey(1)
else:
plt.imshow(cv2.cvtColor(info_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

for url in test_urls:
connection_ok, error_msg = test_stream_connection(url)
if connection_ok:
     working_url = url
     break
else:
     info = [
          f"Тестування URL: {url}",
          f"Причина: {error_msg}",
     ]
     info_img = create_info_image(info)

     if has_gui:
          cv2.imshow('Інформація', info_img)
          cv2.waitKey(1)
     else:
          plt.imshow(cv2.cvtColor(info_img, cv2.COLOR_BGR2RGB))
          plt.axis('off')
          plt.show()

if working_url is None:
info = [
     "Натисніть 'q' для виходу"
]

info_img = create_info_image(info)

if has_gui:
     cv2.imshow('Інформація', info_img)
     while True:
          key = cv2.waitKey(100) & 0xFF
          if key == ord('q'):
```

```
                cv2.destroyAllWindows()
                return
    else:
        plt.imshow(cv2.cvtColor(info_img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()
        return
    else:
    stream_url = working_url

    info = [
    f"URL: {stream_url}",
    f"Модель: ({device})",
    ]

    info_img = create_info_image(info)

    if has_gui:
    cv2.imshow('Інформація', info_img)
    cv2.waitKey(1)
    else:
    plt.imshow(cv2.cvtColor(info_img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

    streaming.is_receiving = True
    receiver_thread = threading.Thread(target=stream_receiver,
args=(stream_url,))
    receiver_thread.daemon = True
    receiver_thread.start()
    time.sleep(2)
    activate_autopilot()
    print("Автопілот активовано успішно")

    last_prediction          = None
    last_prediction_time     = None
    last_distance            = None
    last_confidence          = None
    last_sign_id             = None

    last_distance_reset_time = datetime.now()
    announced_distances      = set()

    global_announced_cache   = {}
    sign_detection_count     = {}
    no_sign_detection_count  = 0

    try:
    while True:
        current_time = datetime.now()
        elapsed_since_reset = (current_time -
```

```
last_distance_reset_time).total_seconds()
          if elapsed_since_reset > reset_distances_cooldown:
               if last_sign_id is not None:
                    global_announced_cache[last_sign_id] = set()
               announced_distances = set()
               last_distance_reset_time = current_time

          autopilot_loop()

          info = [
               f"URL: {stream_url}",
               f"Модель: ({device})",
               f"Отримано кадрів: {streaming.frame_count}"
          ]

          if streaming.last_frame is not None:
               info.append(f"Останній кадр: Доступний
({streaming.last_frame.shape})")

               if streaming.last_frame_time:
                    elapsed = (datetime.now() -
streaming.last_frame_time).total_seconds()
                    info.append(f"Останнє оновлення: {elapsed:.2f}
сек тому")
          else:
               info.append("Останній кадр: Немає")

          if last_prediction is not None:
               info.append(f"Розпізнано знак:
{get_label(last_prediction)}")
               if last_confidence is not None:
                    info.append(f"Впевненість:
{last_confidence:.1f}%")
               if last_distance is not None:
                    info.append(f"Відстань: {last_distance:.1f} см")

          debug_img = create_info_image(info)
          if has_gui:
               cv2.imshow('Інформація', debug_img)
          else:
               plt.imshow(cv2.cvtColor(debug_img, cv2.COLOR_BGR2RGB))
               plt.axis('off')
               plt.show()

          if streaming.last_frame is not None:
               frame = streaming.last_frame.copy()
               display_frame = frame.copy()

               current_time = datetime.now()
               should_predict = (last_prediction_time is None or
                                 (current_time -
```

```
last_prediction_time).total_seconds() > prediction_cooldown)

                if should_predict:
                     found, signs = detect_traffic_sign(frame)

                     if found and signs:
                          recognized_signs = []

                          for sign_info in signs:
                          sign = sign_info['sign']
                          try:
                               sign = stabilize_sign_image(sign,
target_size=(SIZE, SIZE))

                               distance = sign_info['distance']
                               confidence_threshold = 97.0
                               diff_threshold = 30.0

                               if distance > 70:
     continue                                   elif distance < 15:
     continue                                   elif distance > 30 or
distance < 20:
confidence_threshold = 85.0
                                    diff_threshold = 15.0
                               else:
                                    confidence_threshold = 97.0
                                    diff_threshold = 30.0

                               if len(sign.shape) == 2:
                                    sign_rgb = cv2.cvtColor(sign,
cv2.COLOR_GRAY2RGB)
                               else:
                                    sign_rgb = cv2.cvtColor(sign,
cv2.COLOR_BGR2RGB)

                               if distance > 150:
                                    lab = cv2.cvtColor(sign_rgb,
cv2.COLOR_RGB2LAB)
                                    l, a, b = cv2.split(lab)
                                    clahe =
cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
                                    cl = clahe.apply(l)
                                    enhanced_lab = cv2.merge((cl, a,
b))
                                    sign_rgb =
cv2.cvtColor(enhanced_lab, cv2.COLOR_LAB2RGB)

                               transform = A.Compose([
                                    A.Normalize(mean=[0.485, 0.456,
0.406], std=[0.229, 0.224, 0.225]),
                                    A.GaussianBlur(blur_limit=(3, 3),
```

```
p=0.5),

A.RandomBrightnessContrast(brightness_limit=0.1, contrast_limit=0.1,
p=0.5),
                                ])

                                transformed = transform(image=sign_rgb)
                                sign_tensor =
torch.FloatTensor(transformed['image']).permute(2, 0, 1).unsqueeze(0)
                                sign_tensor = sign_tensor.to(device)

                                with torch.no_grad():
                                     outputs = model(sign_tensor)

                                probabilities = F.softmax(outputs,
dim=1)[0]
                                top_p, top_class =
probabilities.topk(2)

                                prediction = top_class[0].item()
                                confidence = top_p[0].item() * 100

                                confidence_diff = (top_p[0] -
top_p[1]).item() * 100

                                if confidence > confidence_threshold
and confidence_diff > diff_threshold:
                                     recognized_signs.append({
                                          'class_id': prediction,
                                          'confidence': confidence,
                                          'confidence_diff':
confidence_diff,
                                          'distance': distance,
                                          'bbox': sign_info['bbox'],
                                          'sign': sign,
                                          'priority':
SIGN_PRIORITIES.get(prediction, DEFAULT_PRIORITY)
                                     })
                           except Exception as e:
                                cv2.putText(display_frame, f"Помилка:
{str(e)}", (10, 30),
                                            cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)
                                no_sign_detection_count += 1

                           if recognized_signs:
                           recognized_signs.sort(key=lambda x:
(x['priority'], -x['confidence']))
                           best_sign = recognized_signs[0]
                           current_sign_id = best_sign['class_id']
```

```
                        for i, sign_data in
enumerate(recognized_signs):
                            x, y, w, h = sign_data['bbox']
                            color = (0, 0, 255) if i == 0 else (0,
255, 0)
                            cv2.rectangle(display_frame, (x, y), (x
+ w, y + h), color, 3)
                            label = f"{sign_data['class_id']}:
{sign_data['confidence']:.1f}%"
                            cv2.putText(display_frame, label, (x,
y-10),
                                        cv2.FONT_HERSHEY_SIMPLEX,
0.5, color, 2)

                        if current_sign_id in sign_detection_count:
                            sign_detection_count[current_sign_id]
+= 1
                        else:
                            old_counts =
sign_detection_count.copy()
                            sign_detection_count =
{current_sign_id: 1}
                            for sign_id, count in
old_counts.items():
                                if sign_id != current_sign_id and
count >= min_consecutive_detections:
                                    sign_detection_count[sign_id]
= count
                            if current_sign_id not in
global_announced_cache:

global_announced_cache[current_sign_id] = set()

                        no_sign_detection_count = 0

                        if sign_detection_count[current_sign_id] >=
min_consecutive_detections:
                            if current_sign_id not in
global_announced_cache:

global_announced_cache[current_sign_id] = set()

                            if last_sign_id != current_sign_id:
                                announced_distances = set()

                            update_sign_data(
                                sign_id=current_sign_id,

confidence=best_sign['confidence'],
                                distance=best_sign['distance']
                            )
```

```
                                last_prediction = current_sign_id
                                last_prediction_time = current_time

                                current_distance =
best_sign['distance']
                                if (20 <= current_distance <= 30 and
     best_sign['confidence'] >= 97.0 and
                                     current_sign_id not in
global_announced_cache):
                                     info_text = f"Знак
{get_label(current_sign_id)}"
                                     priority_level =
best_sign['priority']
                                     if priority_level <= 2:
                                          info_text += " (високий
пріоритет)"
                                     elif priority_level <= 4:
                                          info_text += " (середній
пріоритет)"

                                     speech_text = f"{info_text} на
відстані {int(current_distance)} сантиметрів"
                                     speak(speech_text)


global_announced_cache[current_sign_id] = {current_distance}
                                     print(f"Озвучено знак
{get_label(current_sign_id)} на відстані {int(current_distance)}
см.")

                                print(f"Впевненість:
{best_sign['confidence']}, Відстань: {best_sign['distance']}")
                           else:
                                update_sign_data(
                                     sign_id=current_sign_id,

confidence=best_sign['confidence'],
                                     distance=best_sign['distance']
                                )
                                print(f"Впевненість:
{best_sign['confidence']}, Відстань: {best_sign['distance']}")
                           autopilot_loop()

                           info_text = f"Знак
{get_label(current_sign_id)}"
                           priority_level = best_sign['priority']
                           priority_text = ""
                           if priority_level <= 2:
                                priority_text = " (високий пріоритет)"
                           elif priority_level <= 4:
```

```
                              priority_text = " (середній пріоритет)"
                         info_text += priority_text

                         for threshold in DISTANCE_THRESHOLDS:
                              is_in_local_set = threshold in
announced_distances
                              is_in_global_cache = (current_sign_id
in global_announced_cache and
                                                    threshold in
global_announced_cache[current_sign_id])

                                                                if
(last_distance is not None and                                      20
<= last_distance <= 30 and
                                        not is_in_local_set and
                                        not is_in_global_cache and
                                        best_sign['confidence'] >= 97.0):

                                        speech_text = f"{info_text} на
відстані {int(last_distance)} сантиметрів"
                                        speak(speech_text)
                                        announced_distances.add(threshold)
                                        if current_sign_id not in
global_announced_cache:

global_announced_cache[current_sign_id] = set()

global_announced_cache[current_sign_id].add(threshold)
                                        print(f"Озвучено знак
{get_label(current_sign_id)} на відстані {int(last_distance)} см.")
                                        break

                         if len(recognized_signs) > 1:
                              print(f"Виявлено
{len(recognized_signs)} знаків:\n {get_label(current_sign_id)}")
                              for i, sign_data in
enumerate(recognized_signs):
                                        print(f"  {i+1}.
{get_label(sign_data['class_id'])} - "
                                              f"Пріоритет:
{sign_data['priority']}, "
                                              f"Впевненість:
{sign_data['confidence']:.1f}%, "
                                              f"Відстань:
{sign_data['distance']:.1f} см")
                         else:
                              no_sign_detection_count += 1
                    else:
                         no_sign_detection_count += 1

               if no_sign_detection_count >= max_no_detection_frames:
```

```
                    reset_sign_count()
                    last_prediction = None
                    last_distance = None
                    last_confidence = None
                    no_sign_detection_count = 0

               if last_prediction is not None:
                    display_frame = cv2.cvtColor(display_frame,
cv2.COLOR_BGR2RGB)
                    display_frame = cv2.cvtColor(display_frame,
cv2.COLOR_RGB2BGR)

               if has_gui:
                    cv2.imshow('Розпізнавання', display_frame)
               else:
                    plt.figure(figsize=(12, 6))
                    plt.subplot(121)
                    plt.imshow(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
                    plt.title('Відеопотік')
                    plt.axis('off')
                    plt.subplot(122)
                    plt.imshow(cv2.cvtColor(display_frame,
cv2.COLOR_BGR2RGB))
                    plt.title('Розпізнавання')
                    plt.axis('off')
                    plt.show()
          else:
               error_img = np.zeros((480, 640, 3), dtype=np.uint8)
               error_img = cyrilc(error_img, "Немає відеопотоку",
(180, 240), font_size=32)
               if has_gui:
                    cv2.imshow('Розпізнавання', error_img)
               else:
                    plt.imshow(cv2.cvtColor(error_img,
cv2.COLOR_BGR2RGB))
                    plt.axis('off')
                    plt.show()

          if has_gui:
               key = cv2.waitKey(1) & 0xFF
               if key == ord('q'):
                    break
               elif key == ord('a'):
                    if robot.active:
                         deactivate_autopilot()
                         print("Автопілот деактивовано")
                    else:
                         activate_autopilot()
                         print("Автопілот активовано")
          else:
```

```
                plt.pause(0.03)

            time.sleep(0.03)

        except KeyboardInterrupt:
        print("\nПрограма зупинена користувачем")
        finally:
        stop_url = "http://192.168.4.1/stop"
        response = requests.get(stop_url, timeout=1)
        if response.status_code == 200:
            print("Модель зупинена")
        else:
            print(f"Помилка {response.status_code}")

        streaming.is_receiving = False
        if receiver_thread.is_alive():
            receiver_thread.join(timeout=1)
        if has_gui:
            cv2.destroyAllWindows()
        plt.close('all')

if __name__ == "__main__":
    main()
```

Лістинг Г.3

```
from __future__ import print_function
import sys
import numpy as np
import cv2
from func_systrem.constants import image_extensions
import os
import itertools as it
from contextlib import contextmanager

PY3 = sys.version_info[0] == 3
if PY3:
     from functools import reduce

class Bunch(object):
     def __init__(self, **kw):
     self.__dict__.update(kw)
     def __str__(self):
     return str(self.__dict__)

class Sketcher:
     def __init__(self, windowname, dests, colors_func):
     self.prev_pt = None
     self.windowname = windowname
     self.dests = dests
     self.colors_func = colors_func
     self.dirty = False
     self.show()
     cv2.setMouseCallback(self.windowname, self.on_mouse)
     def show(self):
     cv2.imshow(self.windowname, self.dests[0])
     def on_mouse(self, event, x, y, flags, param):
     pt = (x, y)
     if event == cv2.EVENT_LBUTTONDOWN:
          self.prev_pt = pt
     elif event == cv2.EVENT_LBUTTONUP:
          self.prev_pt = None
     if self.prev_pt and flags & cv2.EVENT_FLAG_LBUTTON:
          for dst, color in zip(self.dests, self.colors_func()):
               cv2.line(dst, self.prev_pt, pt, color, 5)
          self.dirty = True
          self.prev_pt = pt
          self.show()

class StatValue:
     def __init__(self, smooth_coef = 0.5):
     self.value = None
     self.smooth_coef = smooth_coef
     def update(self, v):
     if self.value is None:
```

```
            self.value = v
        else:
            c = self.smooth_coef
            self.value = c * self.value + (1.0-c) * v

class RectSelector:
    def __init__(self, win, callback):
    self.win = win
    self.callback = callback
    cv2.setMouseCallback(win, self.onmouse)
    self.drag_start = None
    self.drag_rect = None
    def onmouse(self, event, x, y, flags, param):
    x, y = np.int16([x, y])
    if event == cv2.EVENT_LBUTTONDOWN:
        self.drag_start = (x, y)
        return
    if self.drag_start:
        if flags & cv2.EVENT_FLAG_LBUTTON:
            xo, yo = self.drag_start
            x0, y0 = np.minimum([xo, yo], [x, y])
            x1, y1 = np.maximum([xo, yo], [x, y])
            self.drag_rect = None
            if x1-x0 > 0 and y1-y0 > 0:
                self.drag_rect = (x0, y0, x1, y1)
        else:
            rect = self.drag_rect
            self.drag_start = None
            self.drag_rect = None
            if rect:
                self.callback(rect)
    def draw(self, vis):
    if not self.drag_rect:
        return False
    x0, y0, x1, y1 = self.drag_rect
    cv2.rectangle(vis, (x0, y0), (x1, y1), (0, 255, 0), 2)
    return True
    @property
    def dragging(self):
    return self.drag_rect is not None



def splitfn(fn):
    path, fn = os.path.split(fn)
    name, ext = os.path.splitext(fn)
    return path, name, ext

def anorm2(a):
    return (a*a).sum(-1)
```

```
def anorm(a):
     return np.sqrt(anorm2(a))

def homotrans(H, x, y):
     xs = H[0, 0]*x + H[0, 1]*y + H[0, 2]
     ys = H[1, 0]*x + H[1, 1]*y + H[1, 2]
     s  = H[2, 0]*x + H[2, 1]*y + H[2, 2]
     return xs/s, ys/s

def to_rect(a):
     a = np.ravel(a)
     if len(a) == 2:
     a = (0, 0, a[0], a[1])
     return np.array(a, np.float64).reshape(2, 2)

def rect2rect_mtx(src, dst):
     src, dst = to_rect(src), to_rect(dst)
     cx, cy = (dst[1] - dst[0]) / (src[1] - src[0])
     tx, ty = dst[0] - src[0] * (cx, cy)
     M = np.float64([[ cx,  0, tx],
                     [  0, cy, ty],
                     [  0,  0,  1]])
     return M

def lookat(eye, target, up = (0, 0, 1)):
     fwd = np.asarray(target, np.float64) - eye
     fwd /= anorm(fwd)
     right = np.cross(fwd, up)
     right /= anorm(right)
     down = np.cross(fwd, right)
     R = np.float64([right, down, fwd])
     tvec = -np.dot(R, eye)
     return R, tvec

def mtx2rvec(R):
     w, u, vt = cv2.SVDecomp(R - np.eye(3))
     p = vt[0] + u[:,0]*w[0]
     c = np.dot(vt[0], p)
     s = np.dot(vt[1], p)
     axis = np.cross(vt[0], vt[1])
     return axis * np.arctan2(s, c)

def draw_str(dst, target, s):
     x, y = target
     cv2.putText(dst, s, (x+1, y+1), cv2.FONT_HERSHEY_PLAIN, 1.0, (0,
0, 0), thickness = 2, lineType=cv2.LINE_AA)
     cv2.putText(dst, s, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.0, (255,
255, 255), lineType=cv2.LINE_AA)

def nothing(*arg, **kw):
     pass
```

```
def clock():
     return cv2.getTickCount() / cv2.getTickFrequency()

@contextmanager
def Timer(msg):
     print(msg, '...',)
     start = clock()
     try:
     yield
     finally:
     print("%.2f ms" % ((clock()-start)*1000))

def grouper(n, iterable, fillvalue=None):
     args = [iter(iterable)] * n
     if PY3:
     output = it.zip_longest(fillvalue=fillvalue, *args)
     else:
     output = it.izip_longest(fillvalue=fillvalue, *args)
     return output

def mosaic(w, imgs):
     imgs = iter(imgs)
     if PY3:
     img0 = next(imgs)
     else:
     img0 = imgs.next()
     pad = np.zeros_like(img0)
     imgs = it.chain([img0], imgs)
     rows = grouper(w, imgs, pad)
     return np.vstack(map(np.hstack, rows))

def getsize(img):
     h, w = img.shape[:2]
     return w, h

def mdot(*args):
     return reduce(np.dot, args)

def draw_keypoints(vis, keypoints, color = (0, 255, 255)):
     for kp in keypoints:
     x, y = kp.pt
     cv2.circle(vis, (int(x), int(y)), 2, color)


_jet_data = {
     'red':   ((0., 0, 0), (0.35, 0, 0), (0.66, 1, 1), (0.89,1, 1),
(1, 0.5, 0.5)),
     'green': ((0., 0, 0), (0.125,0, 0), (0.375,1, 1), (0.64,1, 1),
(0.91,0,0), (1, 0, 0)),
     'blue':  ((0., 0.5, 0.5), (0.11, 1, 1), (0.34, 1, 1), (0.65,0,
```

```
0), (1, 0, 0))
}
cmap_data = { 'jet' : _jet_data }

def make_cmap(name, n=256):
     data = cmap_data[name]
     xs = np.linspace(0.0, 1.0, n)
     channels = []
     eps = 1e-6
     for ch_name in ['blue', 'green', 'red']:
     ch_data = data[ch_name]
     xp, yp = [], []
     for x, y1, y2 in ch_data:
          xp += [x, x+eps]
          yp += [y1, y2]
     ch = np.interp(xs, xp, yp)
     channels.append(ch)
     return np.uint8(np.array(channels).T*255)
```

Лістинг Г.4

```
import albumentations as A
import csv
import cv2
import logging
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from pathlib import Path
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm
from typing import Optional, Tuple, Dict
from func_systrem.constants import (
     SIZE, CLASS_NUMBER, BATCH_SIZE, EPOCHS,
     LEARNING_RATE, NUM_WORKERS, SHAPE_CLASSES,
     COLOR_CLASSES, SIGN_NAMES
)

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

if torch.cuda.is_available():
     logger.info(f"CUDA доступний. Виявлено
```

```
{torch.cuda.device_count()} GPU")
     for i in range(torch.cuda.device_count()):
     logger.info(f"GPU {i}: {torch.cuda.get_device_name(i)}")
else:
     logger.warning("CUDA недоступний! Буде використовуватися CPU")

SIZE = 256
CLASS_NUMBER = 43
BATCH_SIZE = 16
EPOCHS = 75
LEARNING_RATE = 0.0005
NUM_WORKERS = 8

SHAPE_CLASSES = {
     0: "круглий",
     1: "трикутний",
     2: "квадратний",
     3: "восьмикутний"
}

COLOR_CLASSES = {
     0: "червоний",
     1: "синій",
     2: "жовтий",
     3: "білий"
}

def get_sign_name(class_id: int) -> str:
     return SIGN_NAMES.get(class_id, f"Невідомий знак (Клас
{class_id})")

class TrafficSignsDataset(Dataset):
     def __init__(self, csv_file: str, root_dir: str, transform=None,
is_test: bool = False):
     self.annotations = pd.read_csv(csv_file)
     self.root_dir = Path(root_dir)
     self.is_test = is_test
     self.basic_transform = A.Compose([
          A.Resize(SIZE, SIZE),
          A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
     ])
     self.train_transform = A.Compose([
          A.RandomRotate90(p=0.2),
          A.RandomBrightnessContrast(p=0.2),
          A.GaussNoise(p=0.1),
          A.Resize(SIZE, SIZE),
          A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
     ])
```

```
     def __len__(self) -> int:
     return len(self.annotations)

     def __getitem__(self, idx: int) -> tuple[torch.Tensor,
torch.Tensor]:
     img_path = self.annotations.iloc[idx]['Path']
     class_id = self.annotations.iloc[idx]['ClassId']
     full_path = str(self.root_dir.parent / img_path)
     image = cv2.imread(full_path)
     if image is None:
          raise RuntimeError(f"Не вдалося завантажити зображення:
{full_path}")
     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
     if self.is_test:
          transformed = self.basic_transform(image=image)
     else:
          transformed = self.train_transform(image=image)
     image = transformed['image']
     image = torch.FloatTensor(image).permute(2, 0, 1)
     return image, torch.tensor(class_id, dtype=torch.long)

class ImprovedCNN(nn.Module):
     def __init__(self):
     super(ImprovedCNN, self).__init__()
     self.initial = nn.Sequential(
          nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
          nn.BatchNorm2d(64),
          nn.ReLU(inplace=True),
          nn.MaxPool2d(3, stride=2, padding=1)
     )
     self.layer1 = self._make_layer(64, 64, 2)
     self.layer2 = self._make_layer(64, 128, 2, stride=2)
     self.layer3 = self._make_layer(128, 256, 2, stride=2)
     self.attention = nn.Sequential(
          nn.Conv2d(256, 1, kernel_size=1),
          nn.Sigmoid()
     )
     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
     self.classifier = nn.Sequential(
          nn.Linear(256, 512),
          nn.ReLU(inplace=True),
          nn.Dropout(0.5),
          nn.Linear(512, CLASS_NUMBER)
     )
     for m in self.modules():
          if isinstance(m, nn.Conv2d):
               nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
          elif isinstance(m, nn.BatchNorm2d):
               nn.init.constant_(m.weight, 1)
               nn.init.constant_(m.bias, 0)
```

```
     def _make_layer(self, in_channels, out_channels, blocks,
stride=1):
     layers = []
     layers.append(nn.Sequential(
          nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1),
          nn.BatchNorm2d(out_channels),
          nn.ReLU(inplace=True),
          nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1),
          nn.BatchNorm2d(out_channels)
     ))
     if stride != 1 or in_channels != out_channels:
          layers.append(nn.Sequential(
               nn.Conv2d(in_channels, out_channels, kernel_size=1,
stride=stride),
               nn.BatchNorm2d(out_channels)
          ))
     for _ in range(1, blocks):
          layers.append(nn.Sequential(
               nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1),
               nn.BatchNorm2d(out_channels),
               nn.ReLU(inplace=True),
               nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1),
               nn.BatchNorm2d(out_channels)
          ))
     return nn.ModuleList(layers)

     def forward(self, x):
     x = self.initial(x)
     for layer in [self.layer1, self.layer2, self.layer3]:
          identity = x
          for i, block in enumerate(layer):
               if i == 1 and len(layer) > 1:
                    identity = block(identity)
               elif i == 0:
                    x = block(x)
               else:
                    x = block(x)
          x = F.relu(x + identity)
     att = self.attention(x)
     x = x * att
     x = self.avgpool(x)
     x = x.view(x.size(0), -1)
     x = self.classifier(x)
     return x

def load_model(model: nn.Module, model_path: str, device:
```

```
torch.device) -> bool:
     try:
     checkpoint = torch.load(model_path, map_location=device)
     if isinstance(checkpoint, dict):
          if 'model_state_dict' in checkpoint:
               state_dict = checkpoint['model_state_dict']
          else:
               state_dict = checkpoint
     else:
          state_dict = checkpoint
     model.load_state_dict(state_dict)
     logger.info("Модель успішно завантажена")
     return True
     except Exception as e:
     logger.error(f"Помилка: {e}")
     return False

def save_model(model: nn.Module, optimizer: optim.Optimizer,
scheduler, epoch: int,
               val_acc: float, filename: str) -> None:
     torch.save(model.state_dict(), filename)
     full_state = {
     'epoch': epoch,
     'model_state_dict': model.state_dict(),
     'optimizer_state_dict': optimizer.state_dict() if optimizer else
None,
     'scheduler_state_dict': scheduler.state_dict() if scheduler else
None,
     'val_acc': val_acc,
     }
     checkpoint_path = filename.replace('.pth', '_checkpoint.pth')
     torch.save(full_state, checkpoint_path)

def train_model(model: nn.Module, train_loader: DataLoader,
val_loader: DataLoader,
               device: torch.device, num_epochs: int = EPOCHS) ->
None:
     optimizer = optim.AdamW(model.parameters(), lr=LEARNING_RATE,
weight_decay=0.01)
     scheduler = optim.lr_scheduler.OneCycleLR(
     optimizer,
     max_lr=LEARNING_RATE * 10,
     epochs=num_epochs,
     steps_per_epoch=len(train_loader),
     pct_start=0.3,
     div_factor=10.0,
     final_div_factor=100.0
     )
     criterion = nn.CrossEntropyLoss()
     scaler = torch.amp.GradScaler('cuda') if device.type == 'cuda'
else None
```

```
     best_val_acc = 0.0
     patience = 0
     max_patience = 5
     for epoch in range(num_epochs):
     model.train()
     running_loss = 0.0
     correct = 0
     total = 0
     progress_bar = tqdm(train_loader, desc=f'Εποχα
{epoch+1}/{num_epochs}')
     for inputs, labels in progress_bar:
          inputs = inputs.to(device)
          labels = labels.to(device)
          optimizer.zero_grad()
          if device.type == 'cuda':
               with torch.amp.autocast('cuda'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
               scaler.scale(loss).backward()
               scaler.step(optimizer)
               scaler.update()
          else:
               outputs = model(inputs)
               loss = criterion(outputs, labels)
               loss.backward()
               optimizer.step()
          scheduler.step()
          running_loss += loss.item()
          _, predicted = outputs.max(1)
          total += labels.size(0)
          correct += predicted.eq(labels).sum().item()
          progress_bar.set_postfix({
               'loss': f"{running_loss/(progress_bar.n+1):.3f}",
               'acc': f"{100.*correct/total:.1f}%"
          })
          if device.type == 'cuda' and progress_bar.n % 10 == 0:
               torch.cuda.empty_cache()
     model.eval()
     val_loss = 0.0
     val_correct = 0
     val_total = 0
     with torch.no_grad():
          for inputs, labels in val_loader:
               inputs = inputs.to(device)
               labels = labels.to(device)
               if device.type == 'cuda':
                    with torch.amp.autocast('cuda'):
                         outputs = model(inputs)
                         loss = criterion(outputs, labels)
               else:
                    outputs = model(inputs)
```

```
                    loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = outputs.max(1)
                val_total += labels.size(0)
                val_correct += predicted.eq(labels).sum().item()
     val_acc = 100. * val_correct / val_total
     logger.info(f'Епоха {epoch+1}: Вал. точність = {val_acc:.2f}%')
     if val_acc > best_val_acc:
          best_val_acc = val_acc
          torch.save(model.state_dict(),
'best_traffic_sign_model.pth')
          logger.info(f'Збережена нова найкраща модель з точністю
{val_acc:.2f}%')
          patience = 0
     else:
          patience += 1
     if patience >= max_patience:
          logger.info(f'Рання зупинка на епосі {epoch+1}')
          break
     if device.type == 'cuda':
          torch.cuda.empty_cache()

def predict(model: nn.Module, image: np.ndarray, device:
torch.device) -> Tuple[int, float, str]:
     model.eval()
     if len(image.shape) == 2:
     image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
     elif len(image.shape) == 3 and image.shape[2] == 3:
     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
     transform = A.Compose([
     A.Resize(SIZE, SIZE),
     A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
     ])
     transformed = transform(image=image)
     image = transformed['image']
     image = torch.FloatTensor(image).permute(2, 0, 1).unsqueeze(0)
     image = image.to(device)
     with torch.no_grad():
     if device.type == 'cuda':
          with torch.amp.autocast('cuda'):
               outputs = model(image)
     else:
          outputs = model(image)
     probabilities = F.softmax(outputs, dim=1)
     confidence, predicted = probabilities.max(1)
     predicted_class = predicted.item()
     confidence_percent = confidence.item() * 100
     sign_name = get_sign_name(predicted_class)
     if device.type == 'cuda' and torch.cuda.is_available():
     del image, outputs, probabilities, confidence, predicted
```

```
     torch.cuda.empty_cache()
     return predicted_class, confidence_percent, sign_name

def training(batch_size=None, num_workers=None, pin_memory=True) ->
Optional[ImprovedCNN]:
     try:
     logger.info('Ініціалізація датасетів...')
     actual_batch_size = batch_size or BATCH_SIZE
     actual_num_workers = num_workers or NUM_WORKERS
     try:
          train_dataset = TrafficSignsDataset(
               csv_file='dataset/Train.csv',
               root_dir='dataset/Train',
               is_test=False
          )
          test_dataset = TrafficSignsDataset(
               csv_file='dataset/Test.csv',
               root_dir='dataset/Test',
               is_test=True
          )
          train_size = int(0.85 * len(train_dataset))
          val_size = len(train_dataset) - train_size
          train_dataset, val_dataset = torch.utils.data.random_split(
               train_dataset,
               [train_size, val_size]
          )
     except Exception as e:
          logger.error(f"Помилка: {e}")
          return None
     logger.info(f'Розмір тренувального датасету:
{len(train_dataset)} зображень')
     logger.info(f'Розмір валідаційного датасету: {len(val_dataset)}
зображень')
     logger.info(f'Розмір тестового датасету: {len(test_dataset)}
зображень')
     try:
          effective_num_workers = actual_num_workers
          if os.name == 'nt' and effective_num_workers > 0:
               logger.warning("На Windows багатопотокове завантаження
іноді викликає проблеми. Використовуємо 0 workers.")
               effective_num_workers = 0
          logger.info(f"Використовується workers:
{effective_num_workers}, pin_memory: {pin_memory}")
          train_loader = DataLoader(
               train_dataset,
               batch_size=actual_batch_size,
               shuffle=True,
               num_workers=effective_num_workers,
               pin_memory=pin_memory and torch.cuda.is_available(),
               persistent_workers=effective_num_workers > 0,
               drop_last=True
```

```
            )
            val_loader = DataLoader(
                  val_dataset,
                  batch_size=actual_batch_size,
                  shuffle=False,
                  num_workers=effective_num_workers,
                  pin_memory=pin_memory and torch.cuda.is_available(),
                  persistent_workers=effective_num_workers > 0
            )
            test_loader = DataLoader(
                  test_dataset,
                  batch_size=actual_batch_size,
                  shuffle=False,
                  num_workers=effective_num_workers,
                  pin_memory=pin_memory and torch.cuda.is_available(),
                  persistent_workers=effective_num_workers > 0
            )
      except Exception as e:
            logger.error(f"Помилка: {e}")
            return None
      if torch.cuda.is_available():
            device = torch.device('cuda')
            torch.backends.cudnn.benchmark = True
            torch.backends.cudnn.deterministic = False
            torch.cuda.empty_cache()
            logger.info(f'Використовується GPU:
{torch.cuda.get_device_name(0)}')
      else:
            device = torch.device('cpu')
            logger.warning('GPU недоступний, використовується CPU
(навчання буде повільним)')
      try:
            model = ImprovedCNN()
            model = model.to(device)
            total_params = sum(p.numel() for p in model.parameters())
            logger.info(f'Всього параметрів у моделі:
{total_params:,}')
            if device.type == 'cuda':
                  logger.info(f'Зайнято пам\'яті GPU:
{torch.cuda.memory_allocated(0) / 1024 / 1024:.1f} МБ')
                  logger.info(f'Зарезервовано пам\'яті GPU:
{torch.cuda.memory_reserved(0) / 1024 / 1024:.1f} МБ')
      except Exception as e:
            logger.error(f"Помилка: {e}")
            return None
      logger.info('Початок навчання...')
      try:
            train_model(model, train_loader, val_loader, device)
      except Exception as e:
            logger.error(f"Помилка: {e}")
            return None
```

```
        return model
        except Exception as e:
        logger.error(f"Помилка: {e}")
        return None

def show_image(image, title='Зображення'):
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis('off')
        plt.show()

def main():
        try:
        model = ImprovedCNN()
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        model = model.to(device)
        if not load_model(model, 'best_traffic_sign_model.pth', device):
                logger.error("Не вдалося завантажити модель.")
                return
        model.eval()
        except Exception as e:
        logger.error(f"Помилка: {e}")
        return

if __name__ == '__main__':
        main()
```

Лістинг Г.5

Повний код проекту доступний у репозиторії за адресою: https://github.com/dmytrokrylovua/QualificationWork_esp32/tree/master