КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

Нейромережний застосунок розпізнавання дорожніх знаків та об‘єктів на дорозі

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп’ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав:
студент 2 курсу магістратури, групи ТШІ-21
Генташ Ю.А.
(ПІБ)

Науковий керівник: Білан С.М.
(ПІБ)
кандидат технічних наук, доцент
(науковий ступінь, вчене звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент

підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол №12 від «11» травня 2023 р.

Зав. кафедри _____ доц. Іларіонов О.Є.
підпис

Київ 2023

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаної літератури із 21 джерела та додатків. Загальний обсяг роботи 85 сторінок. Робота містить 1 таблицю та 17 рисунків.

Актуальність теми. Останнім часом швидкість розвитку технологій зростає з кожним днем, тому виникає все більше можливостей для впровадження інноваційних рішень у різних сферах діяльності. Однією з таких областей є дорожній рух, де вже застосовуються різноманітні технології для забезпечення безпеки та комфорту учасників дорожнього руху. Одним зі способів підвищення безпеки дорожнього руху є автоматизація процесу розпізнавання дорожніх знаків та об'єктів на дорозі. Це дозволить автоматично визначати швидкість руху, відстань до інших транспортних засобів та передбачати небезпечні ситуації на дорозі.

Мета роботи: покращити надійність функціонування транспортних магістралей, а також підвищити їх безпеку на основі використання нового нейромережевого застосунку для розпізнавання дорожніх знаків та об'єктів на дорозі. Для досягнення поставленої мети в роботі розроблено та реалізувано застосунок на основі згорткової ШНМ. В застосунку використані найсучасніші методи машинного навчання, такі як глибоке навчання та, які дозволяють отримати високу точність в розпізнаванні дорожніх знаків та об'єктів на дорозі.

Об’єкт дослідження – процес виявлення об’єктів на зобрженнях за допомогою нейромережевих технологій.

Предмет дослідження – система виявлення дорожніх знаків та об’єктів на дорозі засобами нейромережевих технологій.

Результати роботи. Програмно реалізована інтелектуальна система для надання інформації про ситуацію на дорозі.

Апробація роботи. Система пройшла тестування

Ключові слова. Нейронна мережа, дорожні знаки, детектування, виявлення, безпека дорожнього руху, асистент водія.

ABSTRACT

The qualification work consists of an introduction, 3 chapters, conclusions, a list of references with 21 sources, and appendices. The total volume of the work is 85 pages. The paper includes 1 table and 17 figures.

Theme`s relevance. Recently, the speed of technology development is growing every day, creating more opportunities for implementing innovative solutions in various fields of activity. One such area is road traffic, where various technologies are already being used to ensure the safety and comfort of road users. One way to improve road safety is to automate the process of recognizing road signs and objects on the road. This will allow for automatic determination of speed, distance to other vehicles, and prediction of dangerous situations on the road.

The purpose of the work: to improve the reliability of transport highways and increase their safety, a new neural network application for recognizing road signs and objects on the road has been developed and implemented in this work. To achieve this goal, a convolutional neural network-based application was developed and implemented. The application uses the most advanced machine learning methods such as deep learning, which allows for high accuracy in recognizing road signs and objects on the road.

The object of the research is the process of object detection in images using neural network technologies.

The subject of the research is the system for detecting road signs and objects on the road using neural network technologies.

Work results. A software-based intelligent system for providing information about the situation on the road.

Approbation of work. The system has been tested

Keywords: neural network, road signs, detection, identification, road safety, driver assistant.

ЗМІСТ

ВСТУП

Останнім часом швидкість розвитку технологій зростає з кожним днем, тому виникає все більше можливостей для впровадження інноваційних рішень у різних сферах діяльності. Однією з таких областей є дорожній рух, де вже застосовуються різноманітні технології для забезпечення безпеки та комфорту учасників дорожнього руху.

Одним зі способів підвищення безпеки дорожнього руху є автоматизація процесу розпізнавання дорожніх знаків та об'єктів на дорозі. Це дозволить автоматично визначати швидкість руху, відстань до інших транспортних засобів та передбачати небезпечні ситуації на дорозі.

У даній дипломній роботі магістра розглянуто застосування нейромереж для розпізнавання дорожніх знаків та об'єктів на дорозі. Штучні нейронні мережі (ШНМ) - це моделі, які наближено реалізують функції мозку нервової системи людини і здатні навчатися та виконувати складні функції, що робить їх ідеальними для розпізнавання образів на дорозі.

Основна мета даної дипломної роботи – покращити надійність функціонування транспортних магістралей, а також підвищити їх безпеку на основі використання нового нейромережевого застосунку для розпізнавання дорожніх знаків та об'єктів на дорозі. Для досягнення поставленої мети в роботі розроблено та реалізувано застосунок на основі згорткової ШНМ. В застосунку використані найсучасніші методи машинного навчання, такі як глибоке навчання та, які дозволяють отримати високу точність в розпізнаванні дорожніх знаків та об'єктів на дорозі.

Для досягнення цієї мети, у роботі будуть розглянуті основні етапи розробки нейромережного застосунку, включаючи збір та підготовку даних, вибір та налаштування найбільш оптимальних моделей нейромереж, тренування

та тестування моделей, впровадження розробленого застосунку для розпізнавання дорожніх знаків та об'єктів на дорозі.

Для вибору та реалізації необхідної ШНМ в роботі розглянуті переваги та недоліки застосування нейромереж для розпізнавання дорожніх знаків та об'єктів на дорозі та можливості для подальшого вдосконалення розробленого застосунку.

Результати цієї дипломної роботи можуть бути корисні для розробки нових систем безпеки дорожнього руху та для покращення існуючих систем, що використовуються на дорогах. Крім того, ця робота може стати важливим внеском у розвиток машинного навчання та нейромереж, що використовуються в інших сферах діяльності.

І АНАЛІЗ ОСОБЛИВОСТЕЙ ЗАДАЧІ РОЗПІЗНАВАННЯ ОБРАЗІВ У КОНТЕКСТІ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

1.1 Огляд існуючих алгоритмів для розпізнавання дорожніх знаків

Сучасні інтелектуальні системи стають все більш популярними завдяки своїм широким можливостям та високій ефективності у багатьох галузях. Вони допомагають людям приймати рішення та є надійними помічниками. Інтелектуальні системи зазвичай працюють на основі аналізу великих обсягів даних і можуть приймати рішення значно швидше, ніж люди. В автомобільній індустрії інтелектуальні системи відіграють вагому роль, оскільки допомагають водіям приймати рішення в режимі реального часу. Згідно з даними Американської національної адміністрації безпеки автомобільних доріг, людські помилки є головною причиною аварій на дорогах. Найчастішими та критичними є помилки прийняття рішень в аварійних ситуаціях, повільна реакція в стресових ситуаціях та проблеми з розпізнаванням елементів дорожнього руху. З цього можна зробити висновок, що необхідні технології, які допомагатимуть уникати аварій або ж надаватимуть водіям достатньо інформації для їх попередження.

Процес розпізнавання дорожніх знаків складається з двох етапів. Перший етап включає виявлення дорожнього знаку на вхідній картинці, зокрема визначення ймовірного розташування знаку та його розмірів. Другий етап полягає в розпізнаванні самого дорожнього знаку, для чого проводиться класифікація. Для розпізнавання дорожніх знаків зазвичай використовують властивості їх форми та кольору, наприклад, інформаційні знаки часто мають синій колір. На другому етапі застосовують нейронні мережі або метод опорних векторів.

В даний час складні математичні підходи до розпізнавання об'єктів на зображеннях знаходять широке застосування в областях комп'ютерного зору, обробки та аналізу зображень, біометрії, систем безпеки та відео-контролю. Сучасні методи розпізнавання об'єктів успішно застосовуються для вирішення

різноманітних завдань, таких як розпізнавання осіб, відбитків пальців, сітківки ока, друкованих символів, автомобільних номерних знаків та маркування на поверхнях різних об'єктів.

Зараз вже досягнуто значних успіхів у розпізнаванні об'єктів та символів на зображеннях, але існують різні труднощі, які можуть значно знижувати надійність сучасних методів:

- низька якість зображення, що може призводити до поганої видимості ознак об'єктів;
- наявність складного фону на зображеннях, де можуть бути інші об'єкти з подібними візуальними ознаками;
- спотворення, що виникають під час реєстрації зображення, наприклад, під час поганих погодних умов або різних кутів та умов освітлення, що можуть створювати шумові перешкоди та спотворення;
- невідома кількість шуканих об'єктів на зображенні заздалегідь.

Тому є потреба у відшуканні методів і алгоритмів реалізації, що дозволяють знизити вплив перерахованих вище складнощів на процес розпізнавання.

Для розпізнавання дорожніх знаків на зображеннях потрібно пройти два основні етапи: спочатку потрібно визначити область, де розміщений дорожній знак, а потім розпізнати його. Кожен з цих етапів вимагає використання своїх методів і алгоритмів.

Далі розглянемо існуючі алгоритми розпізнавання знаків.

1.1.1. Порівняння із заданим шаблоном

Основним принципом цього методу є порівняння кожної області зображення із заданим шаблоном для визначення взаємної кореляції. Шаблони можуть бути задані вручну або визначатися функціями, і можуть відповідати окремим складовим об'єктів або цілому об'єкту. Якщо коефіцієнт кореляції перевищує заданий поріг, то область зображення визначається як така, що містить зображення дорожнього знаку. Деякі вдосконалення цього методу

використовують кілька шаблонів, які відповідають окремим компонентам об'єктів, які можуть бути представлені у вигляді лінійних областей. Алгоритми складаються з кількох етапів, на кожному з яких визначається достовірність знайдених областей. Якщо область зображення відповідає одному з шаблонів, то вона позначається як область, яка представляє інтерес для подальшого аналізу.

Таким чином, з використанням цих методів досягається більш висока подібність між областями зображення та заданим шаблоном. Потім виявлені області порівнюються з іншими шаблонами, що дозволяє визначати наявність шуканого об'єкта. У минулому ці методи використовувалися для класифікації та розпізнавання об'єктів на зображеннях, і їх вважали першими спробами створити ознаки об'єктів на зображеннях [2].

Сьогодні модифікації шаблонних методів демонструють високу точність розпізнавання. Недоліки цих методів полягають у повільній швидкості роботи та високій чутливості до спотворень об'єктів на зображеннях.

1.1.2. Дескриптор локальних особливостей

Для аналізу об’єктів на зображеннях, часто використовують дескриптори локальних особливостей. Дескриптор - це група параметрів, що описують характеристики зображення, такі як колір, текстура та інші. Першим кроком виявлення об'єктів на зображенні за допомогою дескрипторів є визначення характерних точок. Це точки, які мають високу локальну інформативність і параметри, що залишаються незмінними під час фотометричних і геометричних перетворень зображення. Після знаходження безлічі характерних точок на зображенні, для них обчислюють дескриптори [3].

SIFT (Scale Invariant Feature Transform) є одним з найбільш популярних алгоритмів, які включають дескриптор і детектор характерних точок зображення [4]. Цей алгоритм є локальною гістограмою напрямків градієнтів зображення. Принцип роботи SIFT полягає в обчисленні згортки вихідного зображення з ядром Гауса при змінюваному параметрі згладжування. Далі зображення перетворюються до одного розміру, і їх різниця обчислюється. Потім

порівнюється кожен піксель на зображенні з вісьмома сусідніми пікселями при тих же параметрах і масштабі, з дев'ятьма сусідніми пікселями в більшому масштабі і з дев'ятьма в меншому масштабі. Пікселі, в яких локальні екстремуми перевищують заданий поріг, вибираються як характерні точки. Для кожної обраної точки обчислюється певний локальний дескриптор, який характеризує напрямок градієнтів в даній околиці пікселів.

У 2005 році дослідники N. Dalal і B. Triggs представили алгоритм HOG (Histogram of Oriented Gradients) [5]. Принцип роботи цього алгоритму полягає в тому, що зображення подається у вигляді щільної сітки рівномірно розподілених осередків. Для пікселів всередині кожної комірки обчислюються гістограми напрямків градієнтів. На основі отриманих параметрів виконується побудова дескриптора. Для підвищення точності в цьому алгоритмі застосовується нормалізація перекриття локального контрасту гістограм. Нормалізовані дескриптори мають підвищену стійкість до зміни інтенсивності освітлення.

У 2006 році T. Tuytelaars, H. Bay, L. Van Gool представили алгоритм SURF (Speeded Up Robust Features) [6], який включає в себе детектор і дескриптор характерних точок зображення. Цей алгоритм використовує цілочисельні прямокутні фільтри різного масштабу для обчислення характерних точок зображення замість гістограм зважених градієнтів. Це дозволяє забезпечити стійкість до повороту об'єкта та зміни масштабу. Характерні точки вибираються на основі локальних екстремумів пікселів, які перевищують заданий поріг, і обчислюються локальні дескриптори в цих точках. Для обчислення дескрипторів навколо точки будується квадратна область, яка ділиться на кілька підгалузей, в кожній з яких обчислюються відгуки на два типи вейвлетів (горизонтальні та вертикальні). Отримані відгуки зважуються Гауссіаном та підсумовуються. Ці дескриптори широко використовуються для детектування різних об'єктів, включаючи автомобільні номерні знаки, і забезпечують високий рівень інваріантності до геометричних перетворень та зміни масштабу зображення.

Однак недоліком застосування цих дескрипторів є їх низька стійкість до умов освітлення, відображення на поверхнях та різних кутів реєстрації об'єктів.

1.1.3. Гістограмний аналіз зображення

Ці методи базуються на припущенні, що область на зображенні, яка містить символьні образи, має вищу інтенсивність пікселів і відрізняється від інших областей. Щоб підвищити частоту обробки та зменшити шум, на початковому етапі виконують поліпшення контрасту, бінаризацію або виділення кордонів. Це робить межі об'єктів більш контрастними, а фон затемнюється. Зазвичай, в задачах розпізнавання маркувань технічних об'єктів та автомобільних номерних знаків, шуканий об'єкт складається з символів темного кольору, розташованих на світлому тлі. Ідея полягає в скануванні зображення та обчисленні середнього значення яскравості пікселів в кожному рядку зображення. В тому місці, де знаходиться шуканий об'єкт, середня інтенсивність пікселів буде відрізнятися від інших областей зображення, і максимальне значення отриманої проекції може вказувати на розташування об'єкта.

Дослідники застосовували дані методи з різними модифікаціями. У деяких роботах метод використовувався з різними фільтрами для придушення шумів і виділення області автомобільного номерного знаку. В іншій роботі метод використовувався разом з перетворенням Хафа для детектування номерного знаку. Ці алгоритми можуть давати гарні результати, якщо розмір зображення технічного об'єкта можна порівняти з розмірами кадру.

Перевага цих методів полягає в їх простій реалізації та високій швидкості роботи. Однак, недоліком є чутливість до будь-якої області зображення, що має параметри інтенсивності пікселів, схожі з дорожнім знаком.

1.1.4. Адаптивне поліпшення

Алгоритм AdaBoost був придуманий Йоавом Фройндом та Робертом Шапіре в 1999 році як адаптивний метод машинного навчання, який використовується для класифікації об'єктів на зображеннях та відео послідовностях. AdaBoost має каскадну структуру зі слабкими класифікаторами, кожен з яких навчається на помилках попереднього та орієнтований на набір

характеристик, видаючи «вірно» або «брехня» відповіді [7]. Кожен каскад порівнює суму значень слабких класифікаторів з заданим порогом та прибирає з розгляду області зображення, де відсутні ознаки шуканого об'єкта. В результаті залишаються лише області з найбільшою ймовірністю наявності шуканого об'єкта. У 2001 році Паул Віола і Майкл Джонс вдосконалили алгоритм, використовуючи ознаки Хаара для детектування об'єктів на зображеннях. Ознаки Хаара є прямокутними областями, що складаються з кількох суміжних частин, і різні класи об'єктів мають індивідуальні ознаки, які можна виразити у вигляді розподілу ознак Хаара. У кожній області зображення обчислюється кілька тисяч варіантів розташування ознак, залежно від їхнього економічного становища та масштабу, після чого обчислюється різниця між сумами інтенсивностей пікселів в чорних та білих областях ознак Хаара [8].

Алгоритм Віоли-Джонса найчастіше використовується для виявлення осіб, але також може бути використаний для виявлення номерних знаків автомобілів. Хоча цей алгоритм має досить високу точність та швидкість роботи, його недоліком є те, що він відносно неінваріантний до афінних і проекційних спотворень об'єктів на зображеннях, а також до інтенсивності освітлення [1].

1.2 Існуючі системи розпізнавання знаків дорожнього руху

На даний момент на ринку присутні комерційні системи, що забезпечують розпізнавання знаків дорожнього руху та постачаються з автомобілем. Однак, ці системи є закритими і не можуть бути модифіковані. Дослідження предметної області показало, що наявні системи не досягають повного успіху в реальних умовах, таких як різний рівень розмитості, шум, погане освітлення та спотворення.

Загальна структура таких систем, зазвичай складається з камери для зйомки зображення, апаратного модуля захоплення, модуля виявлення, модуля класифікації та бази даних. На вхід системи надходить зображення з камери, яке обробляється за допомогою алгоритму, щоб визначити положення дорожнього

знаку. Потім програма розпізнавання розпізнає знак. Залежно від завдань, які стоять перед системою, база даних може мати різну схему.

Наведемо короткий опис існуючих систем.

1.1.5. Opel Eye

Opel став першим виробником, який адаптував систему розпізнавання знаків дорожнього руху для країн СНД. Їхній інноваційний комплекс розпізнавання дорожніх знаків, Opel Eye доступний і для українських автомобілістів. З 2011 року кожен покупець автомобілів Opel може замовити цю систему в комплектації свого автомобіля. Принцип роботи системи полягає в тому, що відеокамера з широким кутом огляду та високим розширенням, розташована в корпусі під салонним дзеркалом заднього виду, безперервно зчитує інформацію про дорожню обстановку перед автомобілем та передає до 30 кадрів в секунду на обробку двом процесорам. Якщо зображення відповідає типу дорожнього знаку, збереженому в пам'яті системи, на панелі приладів відображається попереджувальний символ - або знак з допустимою швидкістю руху, або знак з забороною обгону.

Рисунок 1 – Opel Eye

Opel Eye - це додаткове «око», що допомагає водієві залишатись уважним на дорозі. Навіть якщо діюче обмеження швидкості не потрапило в поле зору водія, система відображає актуальну інформацію на панель приладів та відразу прибереже водія від проблем, як тільки обмеження буде зняте відповідним знаком.

Система працює в режимі реального часу, тому водій отримує інформацію про поточні умови руху. Ця технологія діє принципово відрізняється від навігаційних систем, які містять інформацію про обмеження швидкості. Крім того, система Opel Eye доповнена функціями попередження про фронтальні зіткнення та зміну полоси руху. Якщо з'являється небезпека зіткнення з

автомобілем, що рухається попереду, або камера фіксує перетин полоси руху, в салоні автомобіля звучить звуковий сигнал, а на приладовій панелі з'являється відповідне повідомлення, що допомагає втомленому водію сконцентруватися та уникнути аварії [9].

Якщо знак дорожнього руху нахилений, частково перекритий або забруднений, то точність розпізнавання суттєво знижується. У цьому випадку загальна точність становить лише близько 75%.

1.1.6. Mercedes Speed Limit Assist

Speed Limit Assist (асистент обмеження швидкості) - це система, що допомагає водієві слідкувати за обмеженнями швидкості на дорогах. Вона працює за допомогою камери, розташованої в області дзеркал заднього виду, яка сканує дорожні знаки та інші об'єкти, що містять інформацію про швидкість.

Коли система розпізнає знак дорожнього руху з інформацією про обмеження швидкості, вона відображає його на панелі приладів автомобіля та починає слідкувати за швидкістю автомобіля. Якщо швидкість автомобіля перевищує обмеження, система попереджає водія звуковим та візуальним сигналами.

Рисунок 2 – Speed Limit Assist

Крім того, якщо система розпізнає, що водій не звертає уваги на обмеження швидкості та перевищує її на постійній основі, вона може надіслати попередження на екран системи мультимедіа в автомобілі. Деякі моделі Mercedes-Benz також мають функцію автоматичного обмеження швидкості, що дозволяє системі самостійно зменшувати швидкість автомобіля до встановленого обмеження на дорозі.

Камера комп'ютера сканує зображення лише для круглих поверхонь та виділяє їх. Далі система алгоритмів фільтрує всі об'єкти, які мають форму кола, але не схожі на дорожні знаки. На останньому етапі, система порівнює зі збереженими шаблонами та відкидає всі об'єкти, крім тих, які програмно запрограмовані для виявлення системою.

Точність розпізнавання значно знижується, якщо знак дорожнього руху нахилений, частково перекритий або забруднений. Загальна точність становить близько 70%.

1.1.7. BMW Speed Assistant

Технологія Speed Assistant від BMW налаштовується під ваші потреби. Система використовує дорожні знаки з обмеженням швидкості, щоб визначити змінні швидкості під час активного круїз-контролю. Якщо ви вибрали відповідні налаштування, ваш BMW автоматично підлаштується під обмеження швидкості під час руху. Водій можете налаштувати цю функцію Speed Assistant на ручний або автоматичний режим.

Рисунок 3 – Speed Assistant

Функція також відображає поточне обмеження швидкості та очікуване обмеження швидкості, якщо воно вибрано. Водій можете налаштувати цю функцію на свої потреби [10].

1.1.8. Volvo Road Sign Information

Road Sign Information є системою, яка допомагає водіям визначати швидкість та інші обмеження на дорозі. Вона використовує камеру, яка знаходиться ззаду дзеркала заднього огляду для сканування дорожніх знаків та інших інформаційних елементів на дорозі.

Після сканування камера передає знайдену інформацію до електронної системи керування автомобілем, яка обробляє дані та відображає їх на приладовій панелі. Інформація може бути відображена у вигляді піктограм або цифр, що вказують на обмеження швидкості, зони обмеженого обгону, зони зупинки та інші важливі дорожні знаки.

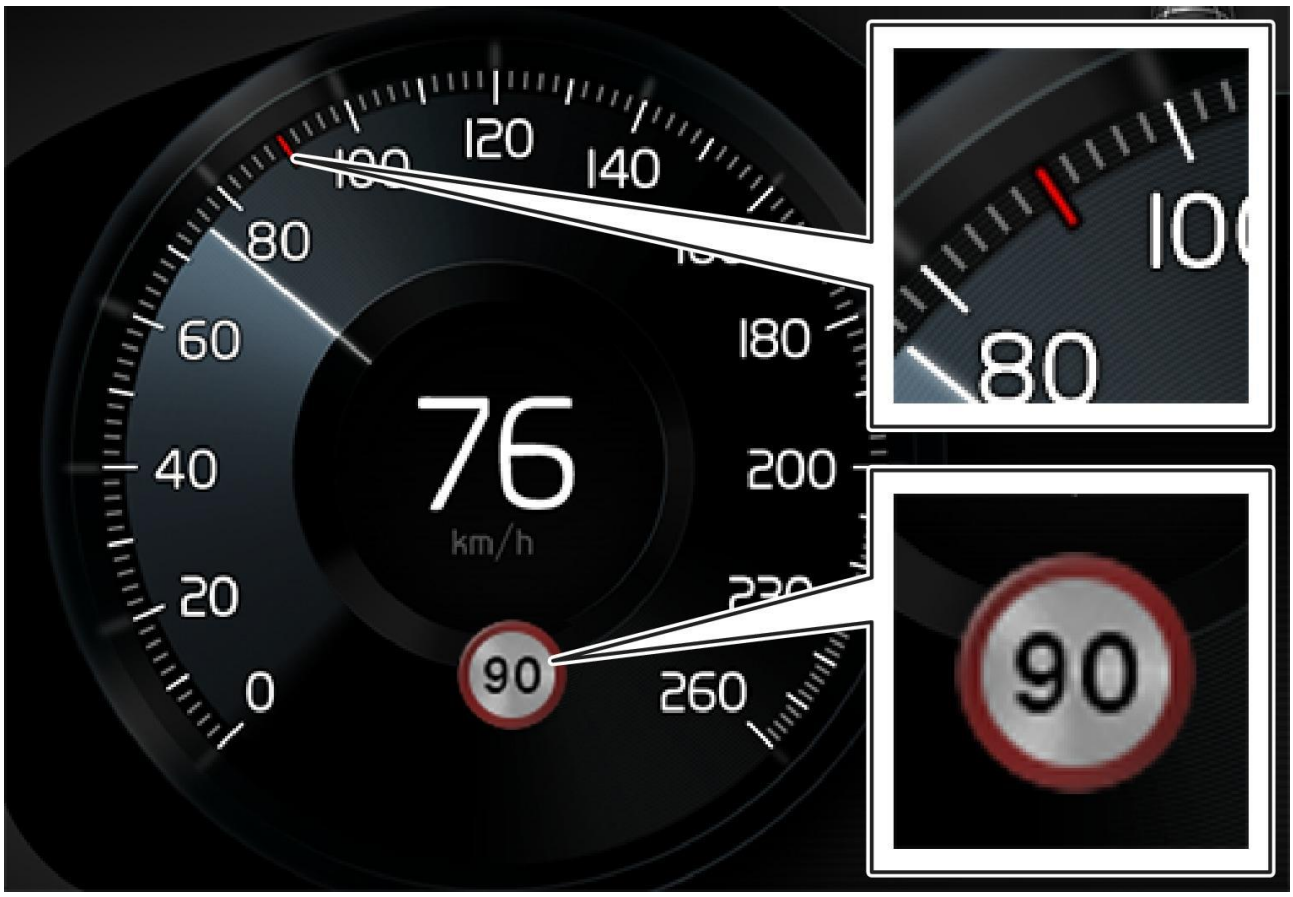

Рисунок 4 – Road Sign Information

Важливо зазначити, що Road Sign Information не лише допомагає водіям визначати обмеження швидкості, але також може попереджати про зони, де потрібно зберігати особливу обережність, наприклад, зони шкіл чи ділянки дороги, де зустрічний рух заборонений.

Ця технологія може бути особливо корисною для водіїв, які не знайомі з дорогами та знаками в нових місцях, або для водіїв, які подорожують в інші країни з різними правилами дорожнього руху. Завдяки Volvo Road Sign Information водії можуть бути впевнені в тому, що дотримуються обмежень та правил дорожнього руху на будь-якій дорозі [11].

1.3 Згорткові нейронні мережі в задачі розпізнавання дорожніх знаків та об’єктів на дорозі

Безумовний лідер на теренах вирішення задач візуальної класифікації – це глибинні штучні нейронні мережі.

Термін “штучні нейронні мережі” сформувався до середини 50-х років XX століття. Основні результати в цій області пов’язані з іменами У.Маккалоха, Д.Хеба, Ф.Розенблатта, М.Мінського, Дж.Хопфілда. Під штучними нейронними мережами розуміють обчислювальні структури, що моделюють прості біологічні процеси, зазвичай асоційовані з процесами людського мозку. Вони являють собою розподілені і паралельні системи, здатні до адаптивного навчання шляхом аналізу позитивних і негативних впливів. Штучні нейронні мережі будуються по принципах організації і функціонування їхніх біологічних аналогів. Розпізнавання образів, ідентифікація, прогнозування, оптимізація, керування складними об’єктами - ШНМ здатні вирішувати широке коло задач. У даній роботі розглянуто задачу розпізнавання об’єктів, заданих за допомогою зображення.

Дослідження у обраній та у інших сферах розпізнавання експериментально довели, що одним з найкращих та найпростіших підходів до розпізнавання є використання згорткових штучних нейронних мереж (Convolutional Neural Networks). Подальші дослідження даної роботи будуть розглядати лише такі ШНМ.

У 1998 році Y. LeCun, L. Bottou, Y. Bengio і P. Haffner запропонували згорткові ШНМ, які базуються на принципах зорової системи людини. ЗШНМ є спеціальним класом багатошарових персептронів з двовимірною структурою, які ефективно обробляють зображення з високим рівнем інваріантності до зсувів, поворотів, масштабування та інших спотворень вхідних даних.

Структура згорткові штучної нейронні мережі складається з послідовності двох типів шарів: згорткових і підвибіркових. Кожен шар містить набір площин, які складаються з штучних нейронів. Кожен згортковий штучний нейрон має зв'язок з невеликою групою нейронів попереднього шару (локальне рецептивне поле), при цьому локальні рецептивні поля нейронів згорткового шару перекриваються частково одне одним у вигляді черепиці. Значення нейронів

локального рецептивного поля множаться на матрицю синаптичних коефіцієнтів і записуються у відповідний нейрон згорткового шару [1].

На рисунку 1.2 можна побачити архітектуру згорткової штучної нейронної мережі.

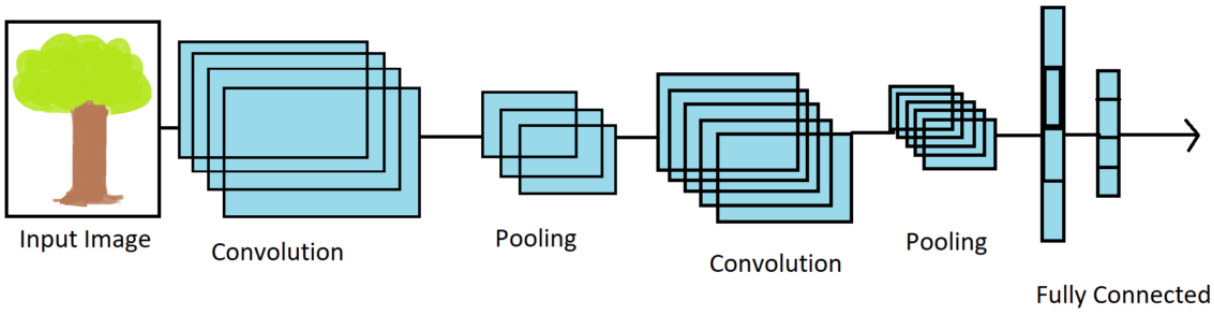

Рисунок 5 – Згорткова штучна нейронна мережа

Архітектура згорткової ШНМ (Convolutional Neural Network, CNN) є потужним інструментом для обробки зображень, що зазвичай складається з кількох шарів: вхідного шару, згорткового шару, шару підвибірки та повнозв'язаного шару. Кожен шар виконує певну функцію, що допомагає нейронній мережі виконувати задачі з обробки зображень.

Вхідний шар є першим етапом обробки зображення в CNN і приймає на вхід зображення. Потім зображення передається до згорткового шару. Згортковий шар містить фільтри, які здійснюють згортку зображення, щоб виділити візуальні ознаки зображення. Фільтри зазвичай мають різні розміри та ваги, що дозволяє їм відображати різні ознаки зображення, такі як грані, кути, текстури тощо.

Після згорткового шару зображення передається до шару підвибірки. Шар підвибірки застосовує метод зменшення розміру зображення, що допомагає скоротити кількість параметрів у моделі та зменшити обчислювальну складність. Зазвичай у шарі підвибірки використовують методи, такі як максимальне

підвибіркове зведення, що дозволяють зберегти найважливіші візуальні ознаки зображення.

Остаточним етапом обробки зображення є повнозв'язаний шар. У повнозв'язаному шарі виконується остаточна класифікація зображення на основі вектора ознак, отриманого з попередніх шарів. Повнозв'язаний шар містить нейрони, які пов'язані з усіма нейронами попереднього шару, що дозволяє здійснювати вибіркове зв'язування між вхідним та вихідним шарами.

Після того, як CNN була навчена на тренувальному наборі даних, її можна використовувати для класифікації нових зображень. Коли зображення подається на вхід CNN, воно проходить через всі шари нейронної мережі та генерується вихідний вектор, що містить ймовірності класів, до яких належить зображення.

CNN застосовується у багатьох задачах обробки зображень, таких як розпізнавання обличчя, класифікація зображень, детекція об'єктів, сегментація зображень та багато інших. Вона дозволяє досягти високої точності та швидкості обробки зображень, що робить її корисним інструментом для багатьох додатків у галузі комп'ютерного зору та штучного інтелекту.

Для вирішення задачі розпізнавання дорожніх знаків на відеопотоці одним з можливих рішень є використання мережі YoloV5.

YoloV5 (You Only Look Once version 5) - це архітектура глибокої нейронної мережі для об'єктного відслідковування в режимі реального часу. YoloV5 розроблена компанією Ultralytics на базі попередніх версій YoloV1-V4.

YoloV5 використовує методи згорткових нейронних мереж, що дозволяють швидко та ефективно розпізнавати об'єкти на зображенні. Особливістю YoloV5 є використання більш легкої та швидкої архітектури з меншою кількістю шарів, що забезпечує покращення швидкодії та точності в порівнянні з попередніми версіями.

У відношенні до задачі розпізнавання дорожніх знаків та об'єктів на дорозі, YoloV5 може розпізнавати дорожні знаки різних форм та кольорів, автомобілі, велосипеди, пішоходів та інші об'єкти, що з'являються на зображенні. YoloV5

може працювати як на відеопотоці в режимі реального часу, так і на статичних зображеннях.

1.4 Набори даних

Наведемо найвідоміші набори даних з дорожніми знаками різних країн світу.

Таблиця 1. Перелік наборів даних із зображеннями дорожніх знаків

Набір даних	Кількість категорій	Кількість зображень
Traffic Sign Detection	4	877
Chinese Traffic Sign	58	6164
DFG Traffic Sign	200	7000
Swedish Traffic Sign	17	3113
Belgian Traffic Sign	30	>10000
Mapillary Traffic Sign	>300	100000
German Traffic Sign Detection Benchmark (GTSDB)	43	600

У таблиці 1 наведено попередні набори даних для класифікації дорожніх знаків. Кожен набір даних є унікальним і має свої особливості:

1) Traffic Sign Detection – невеликий набір даних з 4 класами (світлофор, стоп, обмеження швидкості, пішохідний перехід). В основному використовується для швидкого створення прототипу програми розпізнавання. Також використовується в навчальних цілях, так як займає невелику кількість пам’яті й навчання триває досить короткий період часу.

2) Chinese Traffic Signs, DFG Traffic Sign, Swedish Traffic Sign, Belgian Traffic sign – китайський, словенський, шведський та бельгійський набори даних з дорожніми знаками відповідно.
3) Mapillary Traffic Sign – найбільший з наявних розмічених наборів даних дорожніх знаків. З-поміж інших виділяється не тільки своєю величиною, але й тим, що він містить зображення з усіх куточків світу, а не лише обмежений однією країною.
4) German Traffic Sign Recognition Benchmark (GTSRB) – один з найпоширеніших наборів даних розпізнавання дорожніх знаків. Був представлений в 2011 році на Міжнародній об’єднаній конференції з нейронних мереж (ICJNN). Містить розмічені зображення німецьких дорожніх знаків у різних погодних умовах та з різним освітленням.

1.5 Постановка задачі

Необхідно розробити інтелектуальну систему, яка буде здатна розпізнавати елементи дорожнього руху на відео шляхом використання програмного додатка, який класифікує дорожні знаки з прийнятною точністю. Основне завдання інтелектуальної мережі полягає у розпізнаванні дорожніх знаків на відео.

Розробка системи розпізнавання знаків дорожнього руху є важким завданням, яке може бути ускладнене багатьма факторами, що призводять до низької точності та повноти розпізнавання знаків дорожнього руху. Проблеми, які виникають у будь-якій системі розпізнавання об'єктів, можна розділити на кілька груп, такі як: недостатня освітленість об'єкту, пошкоджені або деформовані знаки, оптичні ефекти, які можуть впливати на точність та повноту алгоритму розпізнавання, а також розмитість та освітленість, які можуть виникати в поганих погодних умовах.

ІІ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАСТОСУНКУ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

2.1 Розробка функціональної структури застосунку

Автоматизована система розпізнавання знаків дорожнього руху буде складатись з декількох підсистем і повинна виконувати усі описані далі функції.

Рисунок 6 – Функціональна структура системи пошуку знаків дорожнього руху

1. Передобробка вхідних даних – виконання операцій, необхідних для приведення вхідного зображення в необхідний формат, підготовка зображення для розпізнавання.

a. Зчитування зображення – отримання знімку з фотокамери та передача його для обробки.

b. Трансформації та обробка зображення – масштабування та стискання зображення.

c. Збереження зображення – збереження обробленого зображення для подальшої передачі його на розпізнавання.

2. Пошук знаків – знаходження та виділення знаків на зображенні, запис отриманої інформації.

a. Детектування знаків – визначення bounding-box’ів (ймовірних ділянок з наявними на них знаками) усіх знаків на зображенні.

b. Розпізнавання знаків – визначення типу дорожнього знака на основі виділених bounding box’ів.

c. Запис результату пошуку знаків – збереження отриманого виходу від мережі у текстовий файл для кожного кадру.

3. Представлення інформації пошуку знаків – виведення у графічний або інший інтерфейс результатів пошуку дорожніх знаків.

a. Вичитування текстового файлу розпізнавання в режимі реального часу – збір інформації про знайдені знаки на кожному кадрі в режимі реального часу.

b. Вивід попереджень – генерація текстових, звукових або візуальних попереджень про знайдені знаки дорожнього руху.

Програмний застосунок буде складатись з декількох частин-модулів.

2.2 Архітектура застосунку

Основними вимогами до розроблюваної системи надання можливості користувачу отримувати в реальному часі попередження про ситуацію на дорозі. При цьому, система повинна бути максимально простою для користувачів. Тобто користувач повинен мати інтуїтивний інтерфейс взаємодії з системою. Прикладом цього може бути відображення знайдених системою знаків на бортовому комп’ютері, подавання звукових сигналів з попередженнями про знаки та інші. Тому одним з елементів системи є інтерфейс користувача.

Другим невід’ємним елементом є засоби захоплення зображення – відеокамера. Як показав досвід світових брендів, якість та швидкість отримання зображення відіграє важливу роль у продуктивності детектування знаків. Задача відеокамери у системі безперервно отримувати вхідний відео-потік та передавати його на останній, найважливіший, елемент системи – модуль детектування знаків.

Модуль детектування знаків повинен співпрацювати з двома іншими елементами системи – відеокамерою та інтерфейсом користувача. При цьому, модуль отримує на вхід дані від відеокамери, а на виході передає дані інтерфейсу для подальшої обробки і відображення користувачу. Вхідні дані являтимуть собою безперервний потік зображень від камери, які будуть оброблятись. Вихідними даними є повний опис детектованих об’єктів на кожному з вхідних зображень.

Оскільки напряму передавати дані детектованих об’єктів на інтерфейс користувача не є хорошим рішенням, додається ще один проміжний елемент системи для обробки виходу модуля розпізнавання та приведення до потрібного формату для інтерфейсу користувача.

Узагальнену схему можна побачити на наступному рисунку.

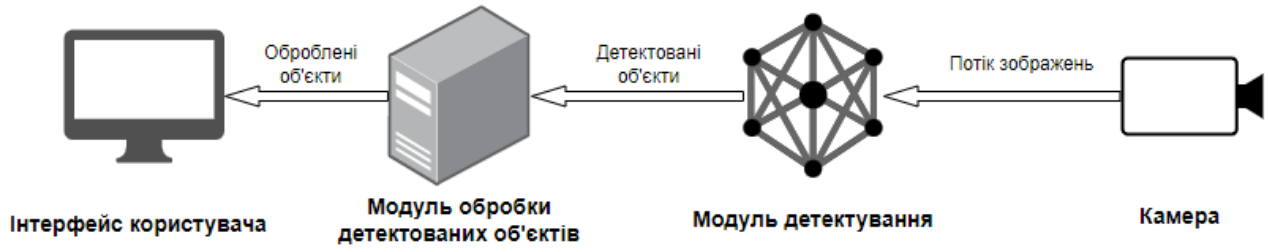

Рисунок 7 – Архітектура системи

Ситуація на дорозі може змінюватись дуже швидко й долі секунд можуть бути вирішальними для уникнення аварійних ситуацій. Це обмеження предметної області виявляє пріоритетну характеристику розроблюваної системи – швидкодію. Система повинна максимально швидко обробляти інформацію і надавати попередження про небезпеку настільки швидко, наскільки цього можна досягти.

У ході роботи було досліджено декілька підходів до вирішення задачі детектування та розпізнавання дорожніх знаків в реальному часі, а саме: Faster R-CNN, YOLO.

2.2.1 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Network) - це популярний алгоритм виявлення об'єктів, який грунтується на попередніх моделях R-CNN та Fast R-CNN. Це двоетапний метод виявлення, який спочатку пропонує регіони об'єктів на зображенні за допомогою Region Proposal Network (RPN), а потім класифікує запропоновані регіони за допомогою мережі Fast R-CNN.

На першому етапі RPN генерує набір пропозицій об'єктів, здійснюючи ковзний шар мережі над згортковою картою ознак вхідного зображення. Мережа передбачає ймовірність наявності об'єкта в кожному положенні вікна ковзання і також генерує набір координат обмежувальних рамок для кожного положення.

На другому етапі мережа Fast R-CNN бере запропоновані регіони від RPN і класифікує їх на різні категорії об'єктів. Мережа Fast R-CNN використовує слой зведення області інтересу (RoI pooling), щоб видобути вектори ознак фіксованої довжини з запропонованих регіонів, які потім надсилаються до серії повнозв'язаних шарів для класифікації та регресії bounding box’ів.

Історія створення алгоритму Faster R-CNN починалась з R-CNN і продовжувалась в Fast R-CNN. Архітектура R-CNN (Region-Based Convolutional Neural Network) була розроблена у 2014 році Россом Гіршиком та іншими [12]. Основою цього методу є наступний алгоритм:

Знаходження потенційних об'єктів на зображенні та розбиття їх на регіони за допомогою методу selective search [13]. Вилучення ознак кожного отриманого регіону за допомогою згорткових нейронних мереж. Класифікація оброблених ознак за допомогою методу опорних векторів (SVM, Support Vector Machine) та уточнення границь регіонів за допомогою лінійної регресії. В результаті отримуємо окремі регіони з об'єктами та їх класами. У центрі стоять згорткові нейронні мережі, які показують хорошу точність на прикладі зображень. Але така архітектура має недоліки:

- Енерговитратна - потребує багато часу на навчання. У методі selective search зображення спочатку сегментується на 2000 регіонів, які потім в ході ітерацій з допомогою жадного алгоритму об'єднуються в більші

регіони. Крім того, самі згорткові мережі також потребують обчислювальних потужностей.

- Не може бути використаний для відео. Знову ж таки, це випливає з недоліку вище, оскільки всі проміжні методи є енерговитратними, тому кадри просто не встигають оброблятися. Selective search не є алгоритмом машинного навчання, тому можуть виникнути проблеми з виявленням потенційних об'єктів на різних зображеннях.

На даний момент модель R-CNN застаріла і не використовується.

Недоліки R-CNN привели авторів у 2015 році до поліпшення моделі. Вони назвали її Fast R-CNN [14]. Ця архітектура ґрунтується на наступному алгоритмі:

Зображення надходить на вхід згорткової нейронної мережі та обробляється selective search. На виході ми маємо карту ознак та регіони потенційних об'єктів. Координати регіонів потенційних об'єктів перетворюються на координати на карті ознак. Отримана карта ознак з регіонами передається до шару RoI (Region of Interest) pooling layer. Тут на кожен регіон застосовується сітка розміром HxW. Потім застосовується MaxPooling для зменшення розмірності. Так, всі регіони потенційних об'єктів мають однакову фіксовану розмірність. Отримані ознаки надходять на вхід повнозв'язного шару, який передається до двох інших повнозв'язних шарів. Перший з функцією активації softmax визначає ймовірність належності класу, другий - границі (зсув) регіону потенційного об'єкта.

Fast R-CNN демонструє трохи вищу точність та значне збільшення швидкості обробки в порівнянні з R-CNN, оскільки не потрібно передавати всі регіони на згортковий шар. Проте, цей метод все ще використовує витратний Selective Search. Тому автори прийшли до Faster R-CNN.

Автори продовжили покращення Fast R-CNN та у 2016 році запропонували Faster R-CNN. Вони розробили власний метод локалізації об'єкту на заміну

Selective Search - RPN (Region Proporsal Networks) [14]. В основі RPN лежить система якорів. Архітектура Faster R-CNN формується наступним чином.

Зображення надходить на вхід згорткової нейронної мережі. Таким чином, формується карта ознак. Карта ознак обробляється шаром RPN. Тут ковзаюче вікно проходить по карті ознак. Центр ковзаючого вікна пов'язаний з центром якорів. Якорі - це області, що мають різні співвідношення сторін та різні розміри. Автори використовують 3 співвідношення сторін та 3 розміри. На основі метрики IoU (Intersection-over-Union), ступеня перетину якорів та правильних прямокутників, вирішується, чи є об'єкт у поточному регіоні. Далі використовується алгоритм Fast R-CNN: карта ознак прикмет з отриманими об'єктами передається шару RoI з наступною обробкою повнозв'язних шарів та класифікацією, а також визначенням зсуву регіонів потенційних об'єктів.

Модель Faster R-CNN справляється трішки гірше з локалізацією об’єктів на зображенні, але працює швидше ніж Fast R-CNN. [15]

2.2.2 YOLO

You Only Look Once (YOLO) є передовим алгоритмом виявлення об'єктів в реальному часі, який був представлений у 2015 році Джозефом Редмоном, Сантошем Діввалою, Россом Гіршіком і Алі Фархаді в їхній знаменитій науковій статті "You Only Look Once: Unified, Real-Time Object Detection".

Автори формулюють проблему виявлення об'єктів як проблему регресії замість класифікаційної задачі, розділяючи просторові рамки та пов'язуючи імовірності кожного виявленого зображення за допомогою однієї згорткової нейронної мережі.

Серед основних переваг YOLO можна виділити:

1- Швидкість

YOLO надзвичайно швидка, оскільки не має складних процесів. Мережа може обробляти зображення зі швидкістю 45 кадрів на секунду (FPS). Крім того, YOLO досягає більше ніж удвічі більшої середньої точності (mAP) порівняно з

іншими системами в режимі реального часу, що робить його відмінним варіантом для обробки в режимі реального часу. З графіку нижче видно, що YOLO далеко перевершує інші виявлення об'єктів з 91 FPS.

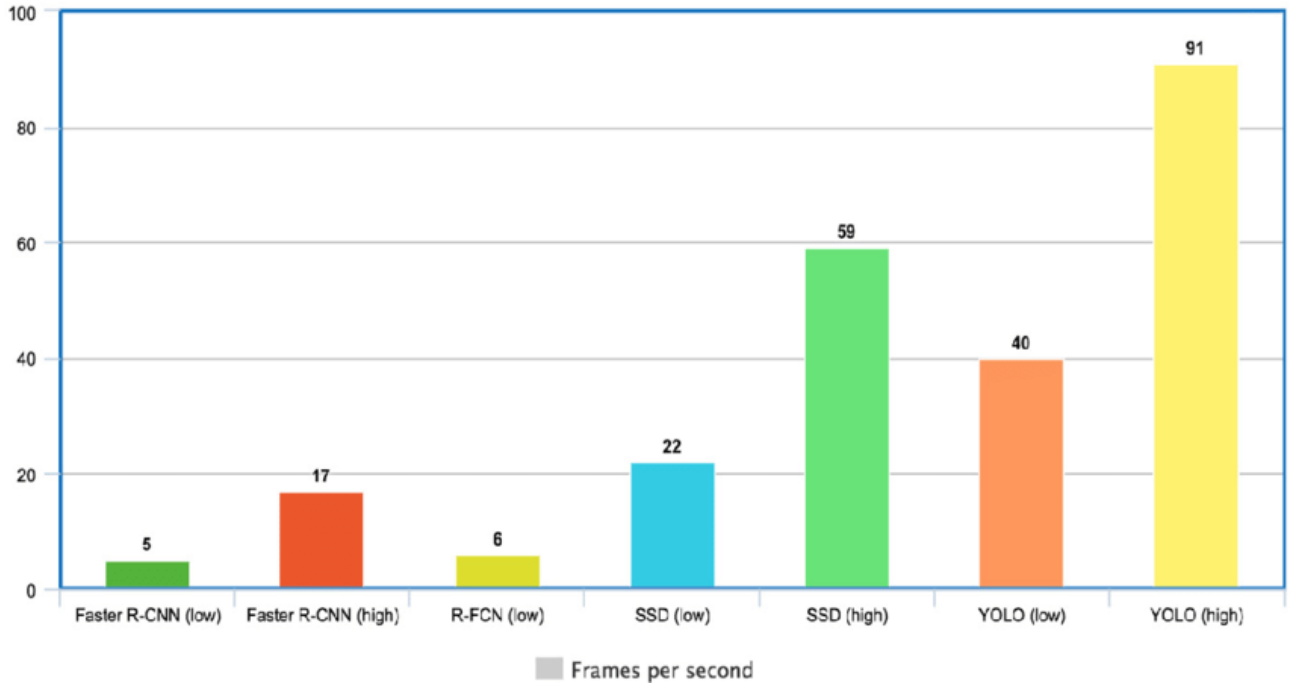

Рисунок 8 – Порівняння швидкості обробки кадрів різними мережами [16]

2- Висока точність виявлення

YOLO, незважаючи на надзвичайну швидкість роботи, не відстає, а у багатьох випадках і випереджає інші сучасні моделі за точністю з дуже малою кількістю помилок на фоні.

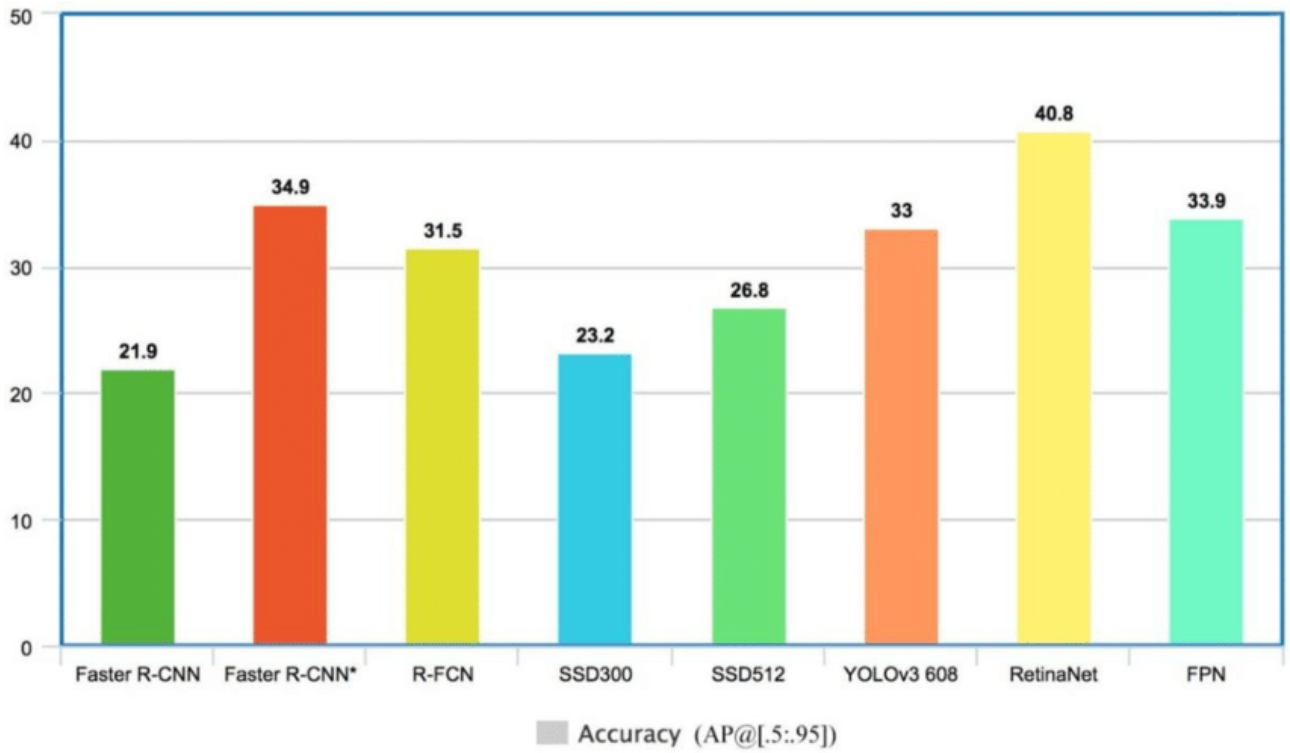

Рисунок 9 – Порівняння точності обробки кадрів різними мережами [16]

3- Краща узагальненість

Це особливо стосується нових версій YOLO. Завдяки цим поліпшенням YOLO здатна забезпечити кращу узагальненість для нових областей, що робить її ідеальною для застосувань, які потребують швидкого та надійного виявлення об’єктів.

Наприклад, у статті про автоматичне виявлення меланоми за допомогою глибоких згорткових нейронних мереж YOLO [17] показано, що перша версія YOLOv1 має найнижчу середню точність виявлення меланоми за автоматичним методом, порівняно з YOLOv2 та YOLOv3.

4 - Відкритий код

Те, що YOLO має доступ з відкритим кодом, дозволило спільноті постійно покращувати модель. Це одна з причин, чому YOLO отримала так багато покращень за досить короткий час.

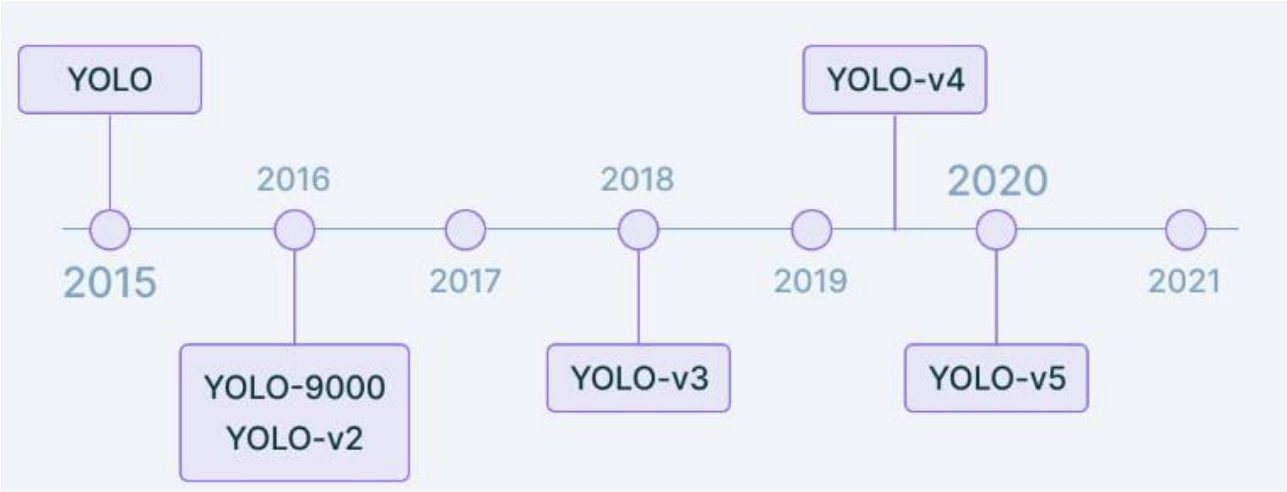

Рисунок 10 – Розвиток мережі YOLO

Найбільш стабільною версією YOLO на даний момент є YOLOv5.

YOLOv5 - це новіша версія мережі YOLO. Порівняно з старішими версіями, YOLOv5 не має опублікованої наукової роботи і вперше реалізована на платформі Pytorch, а не Darknet.

YOLOv5 було випущено Гленном Джочером в червні 2020 року. Архітектура YOLOv5 використовує CSPDarknet53 як основу своєї архітектури, подібно до YOLOv4. Випуск включає п'ять різних розмірів моделі: YOLOv5s (найменший), YOLOv5m, YOLOv5l і YOLOv5x (найбільший).

Одним з головних покращень в архітектурі YOLOv5 є інтеграція шару Focus, який представляє собою один шар, створений шляхом заміни перших трьох шарів YOLOv3. Це дозволило зменшити кількість шарів і параметрів, а також збільшити швидкість як вперед, так і назад, без значного впливу на mAP [18].

Нижче наведено порівняння часу навчання між YOLOv4 та YOLOv5s.

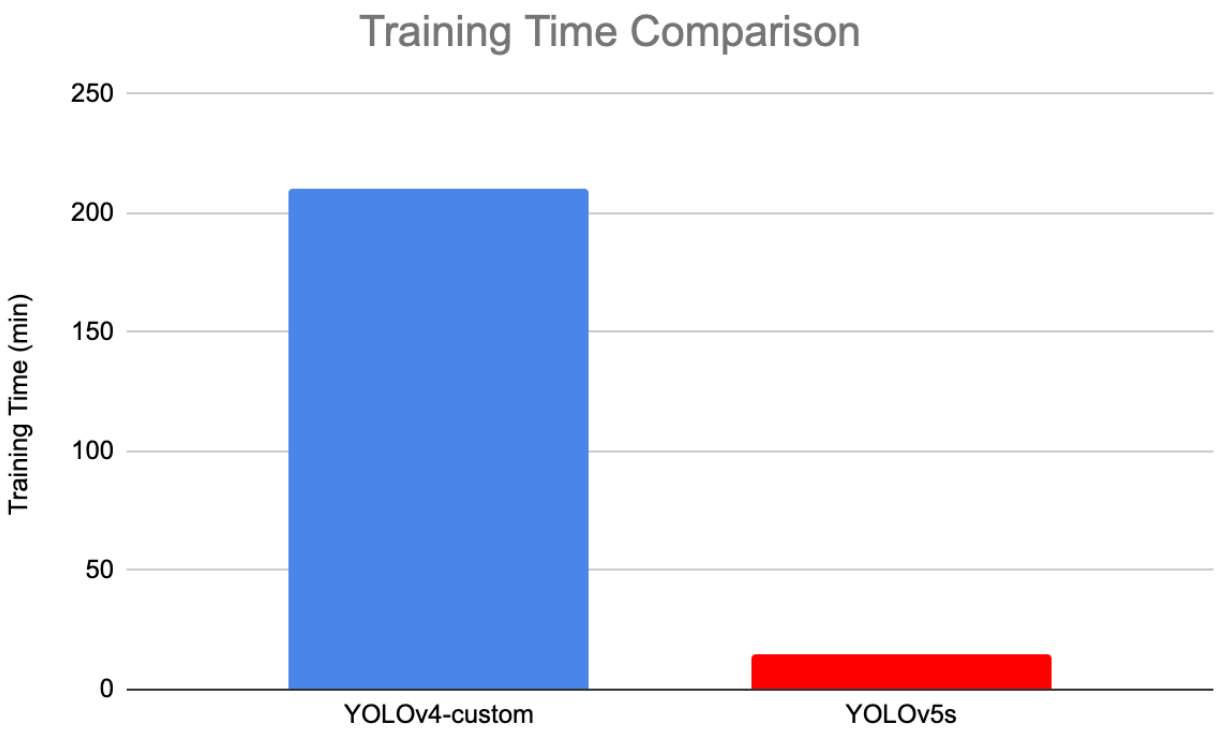

Рисунок 11 – Порівняння часу навчання YOLOv5 та YOLOv4 [19]

2.3 Проектування системи

Робота системи починається від першого елемента – камери. Функція цього елемента очевидна – отримати зображення. Зняте зображення в необробленому вигляді передається на модуль детектування.

Задача модуля детектування полягає в визначенні наявності знаків на вхідному зображенні від камери. Для виконання цієї задачі потрібно виконати 2 задачі – зробити передобробку вхідного зображення та виконати детектування об’єктів. Після передобробки зображення в потрібному нейронній мережі форматі передається для виконання задачі детектування. Задача детектування, в свою чергу, для кожного отриманого вхідного зображення повертатиме інформацію у наступному вигляді:

$[x_1\ y_1, x_2\ y_2, \ldots, x_i\ y_i, \ldots, x_n\ y_n]$

, де

x_i – кількість знайдених знаків y_i,

y_i – тип знайденого знаку (назва),

n – кількість знайдених унікальних знаків.

Тобто для кожного вхідного зображення на виході отримуємо масив з даними про типи знаків та їх кількості на зображенні.

Слід зауважити, що таке представлення є мінімально необхідним для отримання корисної інформації від системи. Фактично, це представлення виходу мережі можна налаштувати довільним чином. Наприклад, окрім наведеної інформації, для кожного детектованого дорожнього знаку можна виводити також координати центру цього знаку. Така інформація може бути дуже корисною для написання певних правил для знаків які можуть бути розташованими лише з однієї сторони дороги, наприклад знаки пріоритету 2.5 «Перевага зустрічного руху» та 2.6 «Перевага перед зустрічним рухом», знайдені у лівій частині дороги (зображення) не є інформативними і, скоріше за все, свідчать про те, що ці знаки були фізично розвернуті й є призначеними для зустрічної смуги руху.

Використання даних масиву достатньо для визначення які знаки було знайдено, але також, в подальших дослідженнях, було б корисно передавати також bounding box’и знайдених знаків. Тобто після знаходження bounding box'ів мережею, можна вирізати з вхідного зображення за координатами знаки для передачі їх на інтерфейс користувача. В даній роботі це не буде реалізовано, але, фактично, це дуже хороше покращення системи, адже у випадку якщо мережа не зможе з високою впевненістю віднести знак до одного з класів, водій сам зможе побачити пропущений його очима знак на дисплеї.

Отримавши вищенаведений масив від модуля детектування, модуль обробки детектованих об’єктів починає свою роботу. Робота даного модуля полягає в перетворенні вхідного масиву розпізнаних знаків на більш корисні дані, відсіюванні неправильних детектувань, виконанні операцій групування та застосування накладання правил декількох знаків однієї категорії.

В залежності від швидкості детектування (кількості кадрів у секунду - FPS), на модуль обробки відправлятиметься ковзне вікно певного розміру результату детектування кількох кадрів. Ця операція виконується не просто так, а через те, що наведені у попередніх розділах мережі хоч і виконують

детектування досить якісно, але в режимі реального часу система може в декількох кадрах виділити bounding box’и з відсутніми у них знаками дорожнього руху. Якщо не фільтрувати ці false positive детектування, то до водія дійде неправильна інформація, яка, в найгіршому випадку, може призвести до аварійної ситуації. Тому таке фільтрування є необхідним. При цьому, розмір ковзного вікна напряму залежить від кількості кадрів, яку обробляє модуль детектування за одиницю часу. Що швидше обробляє мережа зображення, то більше ковзне вікно повинно бути. Далі відбувається перевірка, в рамках якої модуль обробки перевіряє чи на протязі усього ковзного вікна, тобто на усіх його зображеннях було виявлено певний дорожній знак. Якщо виявлено на усіх або на більшості, то можна з високою ймовірністю стверджувати, що знак дійсно виявлено правильно. Якщо ж певний знак був знайдений лише на одному зображенні з ковзного вікна або, наприклад, лише на 10% зображень з нього, то можна вважати що виявлення відбулось випадково і знаку насправді не було на зображенні.

Після відсіювання неправильно виявлених знаків можна виконати певні перетворення даних. Спочатку модуль обробки скорочуватиме кількість даних стиранням дубльованих знаків так, щоб кількість кожного унікального знаку була рівна одиниці. Далі важливим кроком є розбиття знаків за категоріями або групування. В рамках групування знаки будуть розділені за категоріями: попереджувальні, пріоритету, заборонні, наказові.

Далі до кожної з груп можна застосовувати правила які перекривають один одного. Наприклад, якщо раніше був виявлений заборонний знак 3.29 «Обмеження максимальної швидкості Х», а на новому ковзному вікні знайдено знак 3.30 «Кінець обмеження максимальної швидкості Х», то попереднє значення обмеження можна замінити новим знаком – обмеження відсутнє. Така ж ситуація відбувається якщо виявлено 3.29 знак з більшим або меншим значенням обмеження Х. Тобто робота модуля полягатиме в правильній обробці й визначенні того, який знак діє на даний момент.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Модуль детектування

У ході виконання роботи було розглянуто нейронні мережі для детектування дорожніх знаків двох типів: Faster RCNN, YOLOv5.

3.1.1 Вибір набору даних

Більшість наборів згаданих у першій частині даної роботи містять лише частину розмічених зображень, тобто фактично не містять корисної інформації. До корисної інформації відносяться метадані для кожного зображення про те які знаки є на зображенні та координати їх bounding box.

Найбільш підходящими для виконання задачі детектування дорожніх знаків є Traffic Sign Detection та German Traffic Sign Detection Benchmark. Обидва набори даних мають лише розмічені зображення у різних форматах.

Traffic Sign Detection містить всього 877 зображень для 4 класів знаків стоп, обм, пішохідний перехід та світлофора. Зважаючи на суттєве обмеження кількості класів, цей набір не може використовуватись для повноцінної роботі автомобільного помічника, але є чудовим для створення робочого прототипу застосунку.

Даний набір даних містить 2 директорії для зображень та анотацій до них. Директорія зображень містить roadX.png файли, а директорія анотацій містить roadX.xml файли. X-номер зображення, щоб мати можливість зв’язати його з анотаціями.

Приклад вмісту road100.xml:

```
<annotation>
    <folder>images</folder>
    <filename>road100.png</filename>
    <size>
        <width>400</width>
        <height>385</height>
        <depth>3</depth>
```

```
    </size>
    <segmented>0</segmented>
    <object>
      <name>speedlimit</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <occluded>0</occluded>
      <difficult>0</difficult>
      <bndbox>
        <xmin>35</xmin>
        <ymin>5</ymin>
        <xmax>363</xmax>
        <ymax>326</ymax>
      </bndbox>
    </object>
</annotation>
```

У даному XML файлі є батьківський тег <annotation>, який містить усю інформацію, необхідну для визначення положення знаку на зв'язаному зображенні. Теги <folder>, <filename>, <size> показують метаінформацію про зображення – положення на файловій системі та розширення зображення.

Тег <object> є найбільш цікавим в рамках виконання задачі. Він описує положення та тип знака на зображенні. Таких тегів може бути декілька – по одному для кожного знака, якщо їх більше одного на зображенні. <name> визначає назву класу, до якого належить знак, у даному випадку це знак обмеження швидкості, <bndbox> описує bounding box знака – містить xmin, ymin, xmax, ymax – координати лівого нижнього кута та правого верхнього кута знака на зображенні.

Інший набір даних – German Traffic Sign Detection Benchmark має 600 розмічених зображень для 43 типів знаків. Слід зауважити, що відношення

кількості класів до кількості зображень не є найкращим через те, що цей набір даних першочергово використовувався лише для задач класифікації знаків і містить понад 30 тисяч зображень для задачі класифікації. Зважаючи на це, розподіл кількості знаків на зображеннях не буде рівномірним:

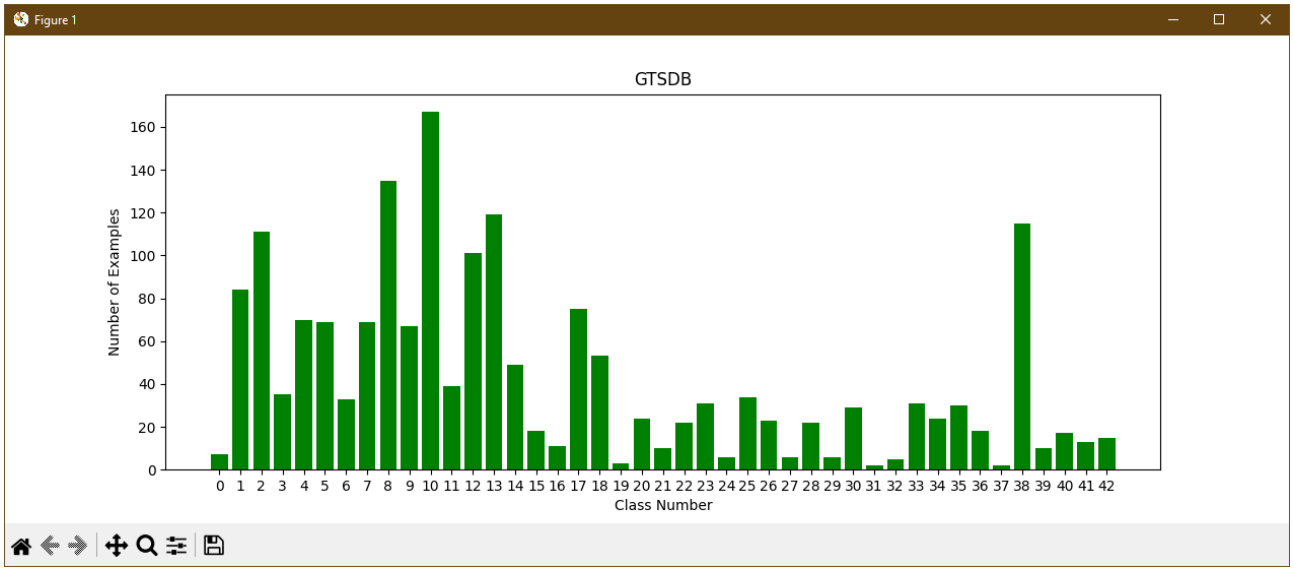

Рисунок 2 – Розподіл кількості знаків наборі GTSDB

Звідси робимо висновок, що детектування найкраще виконуватиметься для знаків під номерами 1, 2, 8, 10, 12, 13, 38. Серед цих знаків найбільшу кількість мають знаки пріоритету та деякі знаки обмеження швидкості.

Метадані цього датасету описані зовсім іншим способом – тут є така ж спільна директорія для усіх зображень, пронумерованих від 0 до 599 з розширенням .ppm, а також є файл gt.txt. Цей файл спільний для усіх зображень і записує дані про кожен знак з нового рядка.

#ImgNo#.ppm;#leftCol#;##topRow#;#rightCol#;#bottomRow#;#ClassID

ImgNo – номер зображення, до якого належить даний знак;

leftCol та topRow – лівий верхній кут bounding box;

rightCol та bottomRow – правий нижній кут bounding box;

ClassID – номер класу (тип знака).

Варто підмітити, що у даному записі немає можливості як для XML додати ще один тег <object> у випадку наявності декількох знаків на одному зображенні,

тому в даному записі й введене додаткове значення, яке ідентифікує приналежність запису до фото – ImgNo. Приклад опису декількох знаків:

00000.ppm;774;411;815;446;11

00001.ppm;983;388;1024;432;40

00001.ppm;386;494;442;552;38

00001.ppm;973;335;1031;390;13

00002.ppm;892;476;1006;592;39

Як бачимо, наведений опис 5 знаків. На першому зображенні є 1 знак з класом 11, на третьому також 1 знак з класом 39, а на другому є 3 знаки з класами 40, 38, 13.

Тобто для отримання корисної інформації з наборів даних необхідно створити власні засоби для вичитування їх, при цьому, кожен набір даних може містити власний формат запису, який треба обробляти самостійно для отримання потрібного формату вхідних даних в нейронну мережу.

В ході виконання роботи було створено Python-скріпти для вичитування обидвох наведених способів зберігання даних про знаки на зображеннях.

3.1.2 Використання Faster R-CNN та YOLOv5

Усі реалізації нейромереж в даній роботі були виконані з використанням мови програмування Python та бібліотеки нейромереж PyTorch.

Першим етапом при створенні мережі є отримання в програмі набору даних так як це було описано в попередньому розділі.

Використовуючи зручну бібліотеку PyTorch можна з мінімальними зусиллями створювати мережі на основі існуючих state of the art архітектур, а також використовувати уже переднавчені системи. У випадку використання Faster R-CNN достатньо завантажити переднавчену мережу та передати їй на навчання власний набір даних

torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False, progress=True, num_classes=num_classes, pretrained_backbone=True)

У даному випадку використовується архітектура на основі ResNet-50-FPN. Перший параметр визначає чи завантажувати повністю переднавчену мережу, другий для виводу прогресу завантаження, третій визначає кількість класів на виході та четвертий вказує чи завантажувати переднавчену основу (у даному випадку ResNet-50-FPN) [20].

Далі достатньо виставити параметри навчання та передати клас з набором власних вхідних даних, який також був створений після вичитування їх з файлів анотацій. Для прискорення навчання в бібліотеці PyTorch є методи, які дозволяють використовувати відеокарту при обчисленнях замість процесора. Це досягається завдяки технології CUDA на відеокартах компанії NVIDIA. На виході навчена мережа зберігається у вигляді .pkl файлу, який надалі можна використовувати для детектування.

Таким чином було отримано навчену мережу, яка далі передається на детектування. Даний етап роботи не відрізняється для YOLOv5 – вичитується збережена мережа, вмикається камера, кожне зображення передається на вхід мережі, отриманий результат – виявлені знаки та модифіковане зображення виводиться на екран. Частота виведення на екран залежить від швидкості обробки зображень мережею.

Слід зауважити, що дана реалізація детектування дорожніх знаків є не надто ефективною за швидкістю як це уже було зазначено в теоретичних дослідженнях, тому така реалізація згодиться лише для навчального використання. Хоч вона й здатна детектувати успішно більшість знаків та показувати їх bounding box, але швидкість її роботи заставляє бажати кращого. При виконанні роботи використовувався ноутбук з процесором Intel Core i5-7200U та відеокартою NVIDIA GeForce 940MX. Дане обладнання є досить малопотужним, але при використанні мережі YOLOv5, воно показує результати кращі приблизно в 20-30 разів. В той час, як Faster R-CNN встигає обробляти кадри з швидкістю приблизно 1 FPS, YOLOv5 показує 20-30 FPS.

Використання мережі YOLOv5 також спрощене до звичайного її завантаження та, у випадку необхідності, модифікації певних шарів. На репозиторії GitHub компанії Ultralytics у вільному доступі є проект yolov5. Після його завантаження можна просто запустити скріпт detect.py і мережа буде працювати на виявлення об'єктів набору даних COCO, тобто в даному випадку мережа теж завантажується переднавченою.

Задача полягає в тому, щоб перенавчити мережу на власний набір даних.

Для цього потрібно отрмати інфомрацію з набору даних так, як це було описано раніше й переформатувати їх у потрібний для мережі YOLOv5 формат. Справа в тому, що мережа використовує дещо інший підхід до визначення bounding box, а саме не кути, а центр зображення та висота й ширина від ного.

Рисунок 13 – Приклад розмітки об’єктів в YOLOv5

На наведеному рисунку можна побачити візуалізацію вигляду формату розміти зображення для YOLOv5. Об’єкт визначається п’ятьма значеннями – класом, x координатою центру, y координатою центру, шириною, висотою. Приклад вигляду опису об'єкта:

41 0.5540441176470589 0.568125 0.016911764705882352 0.02875

34 0.5536764705882353 0.596875 0.016176470588235296 0.02875

11 0.5536764705882353 0.534375 0.023529411764705882 0.03875

У даному випадку можемо зробити висновок, що зображення містить 3 знака – 41, 34, 11, побачити кординати їх центрів та ширину та висту відносно центрів. Координати та висота завжди знаходяться в межах від 0 до 1, тобто вхідні дані повинні бути нормалізовані.

Таким чином, потрібно створити текстовий файл розміти для кожного з зображень та помістити їх в директорію labels.

Наступним кроком є налаштування інформації про набір даних для YOLOv5. Це виконується додаванням файлу з форматом .yaml у шлях yolov5/data. Вміст файлу:

train: ../GTSDByolov5/train/images

val: ../GTSDByolov5/valid/images

test: ../GTSDByolov5/test/images

nc: 43

names: ['10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '6', '9', 'speed limit 100 (prohibitory)', 'speed limit 120 (prohibitory)', 'speed limit 20 (prohibitory)', 'speed limit 30 (prohibitory)', 'speed limit 50 (prohibitory)', 'speed limit 60 (prohibitory)', 'speed limit 70 (prohibitory)', 'speed limit 80 (prohibitory)']

Тут описано місце, де знаходяться зображення для навчання, тестування та оцінки, кількість класів (nc: 43) та назви класів, які будуть відображатись. Насправді класи будуть пронумеровані від 0 до 42, але це не завжди зручно, тому можна добавити власні лейбли, які будуть відображатись замість ID класу.

Далі за шляхом yolov5/data/hyps можна побачити велику кількість інших .yaml файлів. Ці файли містять гіперпараметри для навчання нейронної мережі.

Можна використовувати один з існуючих, або створити власний чи модифікувати наявні.

Крім цього, можна обрати різні варіанти YOLOv5 архітектури. Всього є 4 опції – yolo5s.yaml, yolo5m.yaml, yolo5l.yaml, tolo5x.yaml. Це перелік збільшення складності та розміру архітектури YOLO. Для менш потужного обладнання можна обирати меншу версію, яка буде працювати швидше з меншою точністю, також є можливість описати власний варіант архітектури та використати його.

Щоб навчити мережу залишається запустити команду з вибраними аргументами, описаними вище. Наприклад:

python train.py --img 640 --cfg yolov5s.yaml --hyp hyp.scratch.yaml --batch 32 --epochs 100 --data road_sign_data.yaml --weights yolov5s.pt --workers 24 --name gtsdb17

Дана команда запустить процес навчання мережі з архітектурою yolo5s.yaml з гіперпараметрами навчання hyp.scratch.yaml на наборі даних, описаному в road_sign_data.yaml.

Після виконання процедури навчання, її результати зберігаються за шляхом yolov5/runs/train/<name>, де <name> - назва що вказувалась при запуску процедури навчання.

Наприклад, найкращий навчений варіант мережі при виконанні роботи збережений за шляхом yolov5/runs/train/gtsdb17 і містить наступні дані:

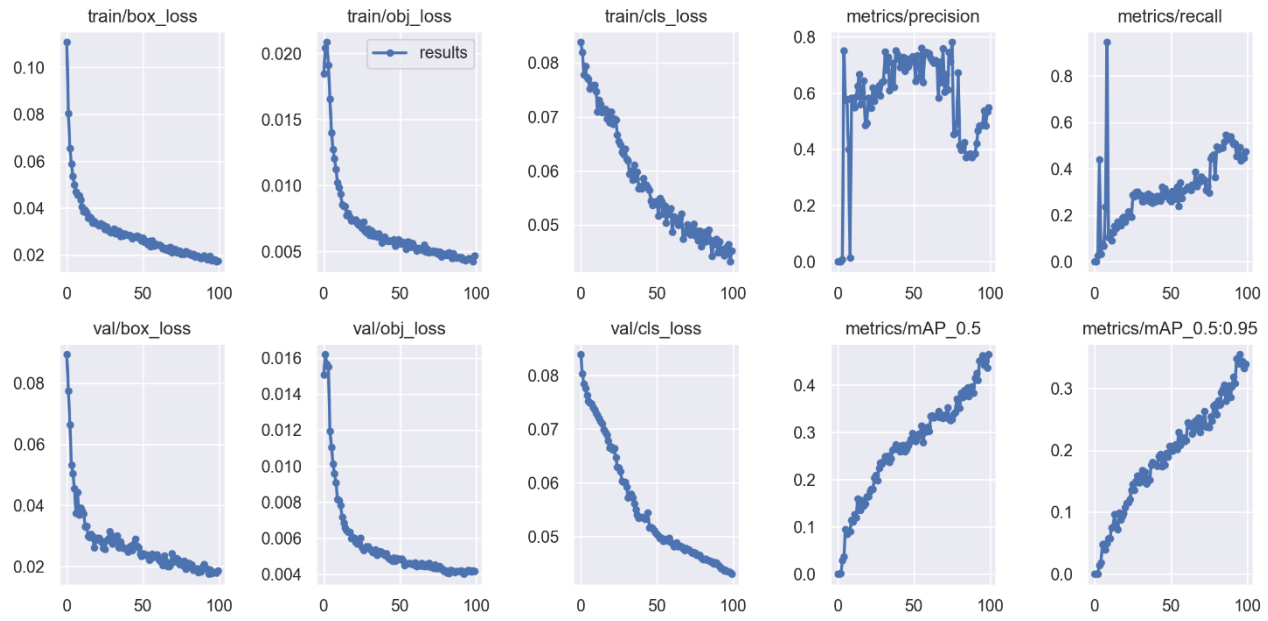

Рисунок 14 – Графіки навчання мережі YOLOv5

epoch,	train/box_loss,	train/obj_loss,	train/cls_loss,	metrics/precision,	metrics/recall,	metrics/mAP_0.5,	metrics/mAP_0.5:0.95,	val/box_loss,	val/obj_loss,	val/cls_loss,	x/lr0,	x/lr1,	x/lr2
0,	0.11072,	0.01847,	0.083848,	0,	0,	0,	0,	0.089404,	0.015069,	0.083888,	0.070227,	0.0033081,	0.0033081
1,	0.080266,	0.020422,	0.082032,	0,	0,	0,	0,	0.077556,	0.016187,	0.080271,	0.040174,	0.0065883,	0.0065883
2,	0.065446,	0.020853,	0.077808,	0.00069228,	0.024691,	0.0014684,	0.00041521,	0.066468,	0.015648,	0.078397,	0.010068,	0.0098152,	0.0098152
3,	0.058808,	0.019125,	0.079359,	0.0071661,	0.44095,	0.028095,	0.013811,	0.053162,	0.015509,	0.077602,	0.00976,	0.00976,	0.00976
4,	0.053536,	0.016546,	0.077319,	0.7511,	0.032922,	0.036233,	0.018332,	0.050465,	0.011935,	0.076365,	0.00976,	0.00976,	0.00976
5,	0.050023,	0.014001,	0.076994,	0.57615,	0.074899,	0.094508,	0.047748,	0.045484,	0.011041,	0.075182,	0.00968,	0.00968,	0.00968
6,	0.046875,	0.012734,	0.075221,	0.57726,	0.067683,	0.084412,	0.048411,	0.03747,	0.010121,	0.07489,	0.0096,	0.0096,	0.0096
7,	0.045615,	0.012024,	0.075939,	0.39888,	0.23678,	0.090516,	0.039182,	0.044276,	0.0095959,	0.074762,	0.00952,	0.00952,	0.00952
8,	0.045278,	0.011229,	0.075642,	0.012805,	0.94609,	0.090501,	0.049567,	0.036955,	0.0090756,	0.074047,	0.00944,	0.00944,	0.00944
9,	0.043534,	0.0102,	0.075943,	0.58227,	0.10536,	0.11438,	0.057312,	0.039259,	0.0081551,	0.073573,	0.00936,	0.00936,	0.00936
10,	0.040508,	0.0098326,	0.074781,	0.58172,	0.12551,	0.11082,	0.057326,	0.038331,	0.0080894,	0.07296,	0.00928,	0.00928,	0.00928
11,	0.038568,	0.0093586,	0.070944,	0.54865,	0.090257,	0.1261,	0.074845,	0.037333,	0.0078161,	0.072587,	0.0092,	0.0092,	0.0092
12,	0.039261,	0.0085504,	0.073162,	0.586,	0.12551,	0.11959,	0.076832,	0.033035,	0.0071759,	0.071972,	0.00912,	0.00912,	0.00912
13,	0.038051,	0.0084212,	0.072304,	0.62554,	0.1535,	0.15985,	0.096312,	0.033228,	0.0068594,	0.071452,	0.00904,	0.00904,	0.00904
14,	0.035843,	0.0084053,	0.070807,	0.66768,	0.14442,	0.14521,	0.076382,	0.029913,	0.0065919,	0.07098,	0.00896,	0.00896,	0.00896
15,	0.036122,	0.007743,	0.071049,	0.55974,	0.17078,	0.13492,	0.071851,	0.029611,	0.0064343,	0.069924,	0.00888,	0.00888,	0.00888
16,	0.035793,	0.0078946,	0.071413,	0.60129,	0.16667,	0.15586,	0.099169,	0.030593,	0.006328,	0.06941,	0.0088,	0.0088,	0.0088
17,	0.033692,	0.0076024,	0.069629,	0.64395,	0.15556,	0.14488,	0.087478,	0.029864,	0.0063507,	0.068918,	0.00872,	0.00872,	0.00872
18,	0.03433,	0.0073153,	0.070967,	0.48589,	0.18683,	0.14886,	0.094064,	0.026109,	0.0059876,	0.067847,	0.00864,	0.00864,	0.00864
19,	0.033426,	0.00738,	0.068921,	0.49423,	0.19177,	0.1634,	0.09819,	0.028668,	0.006047,	0.066544,	0.00856,	0.00856,	0.00856
20,	0.033242,	0.0073701,	0.070969,	0.58308,	0.17217,	0.16384,	0.10809,	0.029349,	0.0060067,	0.066697,	0.00848,	0.00848,	0.00848
21,	0.032502,	0.0074122,	0.068685,	0.57346,	0.19177,	0.17565,	0.1145,	0.029379,	0.0057906,	0.066119,	0.0084,	0.0084,	0.0084
22,	0.033472,	0.0071059,	0.06971,	0.54747,	0.21464,	0.18007,	0.11524,	0.028427,	0.0056733,	0.066405,	0.00832,	0.00832,	0.00832
23,	0.032997,	0.0069817,	0.069438,	0.61979,	0.19462,	0.17957,	0.12109,	0.027945,	0.0058137,	0.064687,	0.00824,	0.00824,	0.00824

Рисунок 15 – .csv файл з даними про навчання по всіх епохах

Рисунок 16 – Приклад оцінки мережі

Директорію weights з файлами навченої мережі best.pt (найкращий знімок ваг мережі за весь процес навчання) та last.pt (знімок ваг мережі після виконання останньої ітерації навчання). Надалі можна використовувати ці файли для запуску детектування.

Запустити мережу можна наступною командою:

python detect.py --source 0 --weights runs/train/gtsdb17/weights/best.pt --conf 0.25 --name yolo_road_det

У цій команді –source 0 вказує що джерелом отримання вхідних зображень буде камера (значення можна змінити на шлях до зображення для його обробки або на директорію чи відео для обробки всіх зображень), --weights – вказує шлях до навченої мережі, --conf – відсікає детектування з впевненістю меншою 0.25, --name – місце для збереження результатів [21].

Рисунок 17 – Приклад роботи мережі в реальному часі

3.2 Модуль обробки детектованих об’єктів та інтерфейс користувача

У попередньому розділі було описано як була створена мережа та як запускати виявлення знаків на камері. На виході на інтерфейс відображається відео-потік з bounding box виявлених знаків. Такі вихідні дані є безумовно корисними при розробці застосунку, але не при використанні її на дорозі. Водій не повинен дивитись на інтерфейс та шукати чи виявився там знак і відмалювався bounding box.

Для вирішення цієї проблеми був створений модуль обробки детектованих об’єктів та інтерфейс користувача, який показує лише корисну інформацію та не заставляє надовго відволікатись від дороги водію.

У попередніх розділах було описано, що модуль обробки детектоваих об’єктів на вхід приймає результати виявлення мережі у текстовому вигляді. Зараз мережа повертає відопотік, який не підходить для вирішення задачі, тому потрібно змінити тип виводу. Для цього у мережі YOLOv5 є додатковий аргумент командного рядку --save-txt. Достатньо вписати його під час запуску команди виявлення python detect.py і результати детектування будуть також записуватись у вигляді текстових документів за шляхом yolov5/runs/detect/<name>/labels. Створена директорія labels спочатку пуста але

коли YOLOv5 виявить хоча б і знак на кадрі, то створить текстовий файл з назвою – номером кадру. Вмістом файлу будуть виявлені знаки у YOLOv5 форматі, такому ж до якого приводився вхідний набір даних. Таким чином, створена директорія міститиме по файлу для кожного кадру, на якому будуть виявлені знаки.

Для створення модуля обробки детектованих об’єктів була використана мова програмування Java.

Задача полягає в тому, щоб певним чином вичитувати усі створювані текстові файли нейронною мережею та записувати результат їх виявлення в пам’ять Java-програми.

Для цього можна скористатись вбудованими можливостями Java з пакету java.nio.file. Даний пакет містить інтерфейс WatchService, який здатний спостерігати за зареєстрованими об'єктами на випадок їх змін та подій. Наприклад, менеджер файлів може використовувати сервіс спостереження для моніторингу каталогу на наявність змін, щоб оновити відображення списку файлів, коли файли створюються або видаляються. Така функціональність повністю підходить для вирішення задачі.

Для цього в Java створюється об’єкт Path з шляхом до директорії labels – де мережа створює текстові файли та на цей об'єкт реєструється наш сервіс спостереження зі значенням параметра ENTRY_CREATE, тобто на створення файла.

```
directory.register(watchService,
StandardWatchEventKinds.ENTRY_CREATE)
```

Далі створюється новий потік для спостереження за змінами в папці та запускається вічний цикл у ньому, де з інтервалом в 1 секунду програма дивитиметься чи не відбулась подія створення файлу. Якщо відбулась, то зі створеного файлу вичитуються усі перші слова кожного рядка (фактично номери класів виявлених знаків). Таким чином в Java ми отримуємо список виявлених знаків останнього обробленого нейронною мережею кадру. Якщо подія не відбулась, то отримаємо пустоту – немає виявлених знаків.

Проте, робити якісь висновки лише на основі одного кадру ще й у рамках задачі детектування об’єктів на дорозі є дуже поганим підходом. Такий підхід призводить до багатьох помилкових виявлень знаків, а неправильна інформація для водія на дорозі може бути критичною і привести до непоправних наслідків. Для уникнення таких ситуацій було прийнято рішення в модулі обробки детектованих об’єктів приймати до уваги набір останніх кадрів - ковзне вікно. У такий спосіб ми позбуваємось одиничних неправильних розпізнавань та показуватимемо інформацію водієві тільки про чітко виявлені знаки на послідовності кадрів.

Для реалізації такої поведінки було створено об’єкт класу CircularFifiQueue з бібліотеки Apache Commons Colletions. Цей клас є реалізацією звичайної черги Queue в Java, але з фіксованим розміром та циклічним заповненням. Це означає що можна визначити розмір такої колекції, наприклад у роботі використовується 5 і при спробі додати шостий елемент, останній який був у колекції на момент додавання видаляється, а першим стає елемент який ми додали. Це реалізація ковзного вікна. У роботі воно використовується коли ми отримали новий файл з виявленими знаками і хочемо додати їх до ковзного вікна. Таким чином, створене ковзне вікно має місце на 5 кадрів, кожен з яких може місити декілька виявлених знаків.

Усі подальші рішення, які будуть робитись в Java програмі виконуватимуться саме на основі 5 останніх кадрів, а не лише одного. Отже, перший потік програми на виході дає певну чергу queue, яка повністю відображає результат останніх 5 виявлених мережею кадрів.

Самі по собі 5 останніх не несуть корисної інформації, тому потрібен ще один потік виконання для вилучення з ковзного вікна потрібної інформації. У новому потоці запускається вічний цикл, який на кожній ітерації рахує кількість виявлених знаків на усіх кадрах (кожен кадр може містить неповторювані значення класів знаків, бо нам важливо лише наявність знаку на кадрі, а не їх кількість). Тобто утворюється асоціативний масив (Map) з ключами – кодами знаку і значеннями – кількістю цих знаків. Наприклад якщо queue містить на

перших трьох кадрах знак стоп і на першому знак увага, то асоціативний масив міститиме такі значення:

«4»-3

«8»-1

Використовуючи створений асоціативний масив можна зробити певні рішення. Вводиться показник впевненості в рішенні мережі. Це значення у скількох кадрах з ковзного вікна повинен бути знак, щоб можна було зробити висновок, що знак був виявлений правильно. Експериментальним шляхом було прийнято рішення виставити цей параметр рівним 80%, тобто якщо ковзне вікно має розмір 5, то потрібно щоб знак був на останніх 4 кадрах з 5 для того щоб бути впевненим що він виявлений правильно. Це викликає невеликий лаг в роботі системи, але враховуючи швидкість роботи нейронної мережі, він буде абсолютно непомітний для людини. З іншого боку ми отримуємо в рази надійнішу систему. Отже, цей потік виконання програми остаточно на виході показує виявлені знаки в реальному часі.

Останнім етапом розробки програми є додавання зручного користувацького інтерфейсу. Для цього в Java існує багато бібліотек, але найбільш сучасною та поширеною є JavaFX.

Для створення інтерфейсу достатньо розширити клас javafx.application.Application і перевизначити метод start(Stage) цього класу. В перевизначеному методі створюємо вікно та додаємо на нього пусті місця для відображення зображень виявлених знаків. Для заповнення цих пустих місць потрібно при виявленні нових знаків заповнювати їх даними. Для цього в попередньому потоці виконання, в місці де ми вирішуємо чи виявили знак на основі показника впевненості в рішенні мережі викликатимемо метод, який заповнюватиме пусті місця для зображень відповідними файлами. Також було створено допоміжний метод з відповідниками коду класу до зображення на файловій системі.

Фактично застосунок готовий до використання, але є певні незручності в його користуванні. Основна проблема полягає в тому, що відображаються знаки

на екрані лише тоді, коли в 4 з 5 останніх кадрів виявлено знак і як тільки водій проїжджає знак, він також зникає з екрану. Вирішити цю проблему можна додаванням певного відліку, після якого знак зникатиме з екрану. Для цього крім виявлених знаків буде запам’ятовуватись і час їх виявлення. Якщо знак знову був виявлений на ковзному вікні, час обновлюється, якщо ж не виявлений і пройшло 10 секунд від виявлення, знак зникає. Це дозволяє бачити знак весь час поки його виявила нейронна мережа і ще 10 секунд після цього.

ВИСНОВКИ

Отже, у першому розділі було розглянуто яким чином нейромережні технології можуть допомогти вирішувати проблеми під час руху в автомобілі, розглянуто теорію детектування та уже існуючі методи виявлення дорожніх знаків на зображеннях. Було досліджено які набори даних, як правило, використовуються для вирішення даної задачі.

У другому розділі було визначено які функції повинна виконувати розроблювана система, була обрана загальна архітектура. Повністю була проаналізована інформація, яка повинна зберігатись застосунком. Наступним кроком було спроектовано простий інтерфейс користувача. Обрано найбільш підходящу архітектуру нейронної мережі для поставленої задачі.

У третьому розділі було створено інтерфейс користувача засобами мови програмування Java, а саме бібліотеки JavaFX. Детектування знаків мережами Faster R-CNN та YOLOv5 реалізоване на мові Python і використовується для передачі інформації про виявлені знаки на кадрах вхідного відео-потоку. Виявлені знаки передаються на модуль обробки детектованих об’єктів та обробляються для подальшого відображення користувачу в найзручнішому вигляді.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Солдатова О.П. Применение сверточной нейронной сети для распознавания рукописных цифр. Компьютерная оптика. 2010. № 2. С. 252–259.

2. Каковкин П.А. Применение алгоритмов глубокого обучения для локализации и распознавания дорожных знаков на изображениях. Высокие технологии в современной науке и технике: сборник научных трудов IV Международной научно-технической конференции молодых ученых, аспирантов и студентов. Томск: ТПУ, 2015. С. 360–364.

3. Якимов П.Ю. Отслеживание дорожных знаков в видеопоследовательности с использованием скорости. Компьютерная оптика. 2015. № 5. С. 795–800.

4. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image: patent No. US6711293B1 Appl. No. 09/519,893; Filed 06.03.2000; publ. 23.03.2004, 20 p.

5. Dalal N., Triggs B., Histograms of oriented gradients for human detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Montbonnot, France 2005. P. 886-893

6. Bay H., Ess A., Tuytelaars T., Luc Van Gool, Speeded Up Robust Features, Computer Vision and Image Understanding, 2008, doi: 10.1016/J.CVIU.2007.09.014

7. Schapire R. E., Singer Y., Improved Boosting Algorithms Using Confidence-rated Predictions. Machine Learning. Netherlands: Kluwer Acamedic Publishers, 1999, P. 297–336

8. Митчелл Б.Ф. Нахождение ближайшей к началу координат точки Вестник ЛГУ. 1971. № 19. С. 38–45

9. https://autoconsulting.ua/article.php?sid=21029

10. https://www.bmwofminnetonka.com/bmw-adapt-to-speed-limit-automatically/

11. https://www.volvocars.com/en-th/support/manuals/v90/2018w46/driver-support/road-sign-information/road-sign-information-and-sign-display

12. https://arxiv.org/abs/1311.2524

13. https://link.springer.com/article/10.1023/B:VISI.0000022288.19776.77

14. https://arxiv.org/abs/1504.08083

15. https://medium.com/@bigdataschool/3-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0-%D0%B4%D0%B5%D1%82%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BE%D0%B2-c-deep-learning-r-cnn-fast-r-cnn-%D0%B8-faster-r-cnn-acdf6380fd33

16. Sanchez S. A., Romero H. G., Morales A. D., A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework, 2020

17. Yali Nie, Paolo Sommella, Mattias O'Nils, Consolatina Liguori, Jan Lundgren, Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks, 2019

18. https://www.datacamp.com/blog/yolo-object-detection-explained

19. https://blog.roboflow.com/yolov4-versus-yolov5/

20. https://sidthoviti.com/traffic-sign-detection-using-faster-rcnn-gtsdb/

21. https://blog.paperspace.com/train-yolov5-custom-data/

ДОДАТКИ

main.py (Faster R-CNN)

```
import os

import cv2
import numpy as np
import torch
import torchvision
from PIL import Image
from torch.utils import data
from torchvision.transforms import functional as F

import transforms as T
import utilss
from engine import train_one_epoch, evaluate

txt = np.genfromtxt('train/gt.txt', delimiter=';', dtype=None, encoding=None)

loadModel = True

# EXTRACTING SIGNS INFO TO MAP (IMAGE->ALL SIGN ANNOTATIONS)

dic = {}
for i in range(0, len(txt)):
    img_name = txt[i][0]
    target = [txt[i][1], txt[i][2], txt[i][3], txt[i][4], txt[i][5]]
    clas = txt[i][-1]
    if img_name in dic:
        dic[img_name].append(target)
```

```
    else:
        dic[img_name] = [target]
print("Number of signs: " + str(len(dic)))


# DATA DISTRIBUTION PLOT

# cls_lst = {}
#
# for i in dic:
#     for j in dic[i][:]:
#         # print(len(dic[i]))
#         for k in range(len(dic[i])):
#             clss = dic[i][:][k][-1]
#             if clss in cls_lst:
#                 cls_lst[clss] += 1
#             else:
#                 cls_lst[clss] = 1
#
# print(cls_lst)
#
# xx = []
# yy = []
#
# for i in sorted(cls_lst.keys()):
#     xx.append(str(i))
#     yy.append(cls_lst[i])
#
# x_pos = [i for i, _ in enumerate(xx)]
#
```

```
# plt.bar(x_pos, yy, color='green')
# plt.xlabel("Class Number")
# plt.ylabel("Number of Examples")
# plt.title("GTSDB")
# plt.xticks(x_pos, xx)
#
# plt.show()

# EXTRACTING ONLY IMAGES WITH ANNOTATIONS

# pt = glob('./train/images/*.ppm')
#
# len(pt)
# #Copying into new directory
# for i in range(len(dic)):
#     ofile = r'./train/images/{}'.format(list(dic)[i])
#     target = r'./train/imagesf/{}'.format(list(dic)[i])
#     shutil.copyfile(ofile, target)
# #Check if len(dic) == number of images in folder
# print(len(glob('./train/imagesf/*.ppm')))
# len(dic)

# CUSTOM DATASET IMPLEMENTATION
class myDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.imgs = list(sorted(os.listdir(os.path.join(root, "imagesf"))))

    def __getitem__(self, idx):
```

```
img_path = os.path.join(self.root, "imagesf", self.imgs[idx])
img = Image.open(img_path).convert("RGB")
objects = dic[self.imgs[idx]]
boxes = []
labels = []
for obj in objects:
    name = obj[-1]
    labels.append(np.int64(name))
    xMin = np.float64(obj[0])
    yMin = np.float64(obj[1])
    xMax = np.float64(obj[2])
    yMax = np.float64(obj[3])
    boxes.append([xMin, yMin, xMax, yMax])

boxes = torch.as_tensor(boxes, dtype=torch.float32)
labels = torch.as_tensor(labels, dtype=torch.int64)

image_id = torch.tensor([idx])
area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
iscrowd = torch.zeros((len(objects),), dtype=torch.int64)

target = {}
target["boxes"] = boxes
target["labels"] = labels
target["image_id"] = image_id
target["area"] = area
target["iscrowd"] = iscrowd

if self.transforms is not None:
    img, target = self.transforms(img, target)
```

```
        return img, target

    def __len__(self):
        return len(self.imgs)


def get_transform(train):
    transforms = []
    # converts the image, a PIL image, into a PyTorch Tensor
    transforms.append(T.ToTensor())
    return T.Compose(transforms)


# DEFINING MODEL
os.environ['TORCH_HOME'] = './'

root = r'./train'

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

num_classes = 44
dataset = myDataset(root, get_transform(train=True))
dataset_test = myDataset(root, get_transform(train=False))

# split the dataset in train and test set
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-100])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-100:])
```

```
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=1, shuffle=True,  # num_workers=4,
    collate_fn=utilss.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=1, shuffle=False,  # num_workers=4,
    collate_fn=utilss.collate_fn)

# Define model
if loadModel:
    model = torch.load('./train10.pkl')
else:
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False,
progress=True,
                                                                 num_classes=num_classes,
pretrained_backbone=True)

model.to(device)

print("Model loaded")

# TRAINING MODEL
if not loadModel:
    params = [p for p in model.parameters() if p.requires_grad]
    optimizer = torch.optim.SGD(params, lr=0.0005,
                                momentum=0.9, weight_decay=0.0005)
    lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer,
T_0=1, T_mult=2)
    num_epochs = 10
```

```
    losses = []
    loss_box_reg = []
    loss_rpn_box_reg = []
    loss_classifier = []
    loss_objectness = []

    stat0 = []
    stat1 = []
    stat2 = []
    stat3 = []
    stat4 = []
    stat5 = []
    stat6 = []
    stat7 = []
    stat8 = []
    stat9 = []
    stat10 = []
    stat11 = []

    for epoch in range(num_epochs):
        metrics = train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=50)
        losses.append(float(str(metrics.meters['loss']).split(" ")[0]))
        loss_box_reg.append(float(str(metrics.meters['loss_box_reg']).split(" ")[0]))
        loss_rpn_box_reg.append(float(str(metrics.meters['loss_rpn_box_reg']).split(" ")[0]))
        loss_classifier.append(float(str(metrics.meters['loss_classifier']).split(" ")[0]))
        loss_objectness.append(float(str(metrics.meters['loss_objectness']).split(" ")[0]))

        # Update the learning rate
```

```
lr_scheduler.step()
_, metric_logger = evaluate(model, data_loader_test, device=device)
stat = _.coco_eval['bbox'].stats

# Append all stats
stat0.append(stat[0])
stat1.append(stat[1])
stat2.append(stat[2])
stat3.append(stat[3])
stat4.append(stat[4])
stat5.append(stat[5])
stat6.append(stat[6])
stat7.append(stat[7])
stat8.append(stat[8])
stat9.append(stat[9])
stat10.append(stat[10])
stat11.append(stat[11])

# SAVING THE MODEL
torch.save(model, r'train1.pkl')

torch.save(model.state_dict(), 'train1.pth')
torch.save({
    'epoch': epoch,
    "model_state_dict": model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
}, 'ckpt1.pth')

print('')
print('==========================================')
```

```
        print('')
    print("Done!")


def showbbox(model, img):
    model.eval()
    with torch.no_grad():
        prediction = model([img.to(device)])

    print(prediction)
    b = prediction[0]['boxes']
    s = prediction[0]['scores']

    # Apply Non-maximum suppression
    keep = torchvision.ops.nms(b, s, 0.1)

    img = img.permute(1, 2, 0)
    img = (img * 255).byte().data.cpu()
    img = np.array(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    classes = {0: ' Speed limit (20km/h)',
               1: ' Speed limit (30km/h)',
               2: ' Speed limit (50km/h)',
               3: ' Speed limit (60km/h)',
               4: ' Speed limit (70km/h)',
               5: ' Speed limit (80km/h)',
               6: ' End of speed limit (80km/h)',
               7: ' Speed limit (100km/h)',
               8: ' Speed limit (120km/h)',
               9: ' No passing',
```

10: ' No passing veh over 3.5 tons',

11: ' Right-of-way at intersection',

12: ' Priority road',

13: ' Yield',

14: ' Stop',

15: ' No vehicles',

16: ' Veh > 3.5 tons prohibited',

17: ' No entry',

18: ' General caution',

19: ' Dangerous curve left',

20: ' Dangerous curve right',

21: ' Double curve',

22: ' Bumpy road',

23: ' Slippery road',

24: ' Road narrows on the right',

25: ' Road work',

26: ' Traffic signals',

27: ' Pedestrians',

28: ' Children crossing',

29: ' Bicycles crossing',

30: ' Beware of ice/snow',

31: ' Wild animals crossing',

32: ' End speed + passing limits',

33: ' Turn right ahead',

34: ' Turn left ahead',

35: ' Ahead only',

36: ' Go straight or right',

37: ' Go straight or left',

38: ' Keep right',

39: ' Keep left',

```
            40: ' Roundabout mandatory',
            41: ' End of no passing',
            42: ' End no passing veh > 3.5 tons'}

    for k in range(len(keep)):
        xMin = round(prediction[0]['boxes'][k][0].item())
        yMin = round(prediction[0]['boxes'][k][1].item())
        xMax = round(prediction[0]['boxes'][k][2].item())
        yMax = round(prediction[0]['boxes'][k][3].item())

        label = prediction[0]['labels'][k].item()
        print("Label is: {}\n===\n(Xmin, Ymin, Xmax, Ymax) = ({}, {}, {}, {})
\n===".format(label, xMin, yMin, xMax, yMax))
        colors = np.random.uniform(0, 255, size=(43, 3))
        if label in classes:
            pt1 = (xMin, yMin)
            pt2 = (xMax, yMax)
            print("Class Label: " + classes[label])
            score = prediction[0]['scores'][k].item()
            print("Score: " + str(score))
            print("\n==============\n")
            color = list(colors[label])
            cv2.rectangle(img, pt1, pt2, color, thickness=2)
            cv2.putText(img, classes[label] + "-" + str(round(score, 2)), (xMin, yMin),
cv2.FONT_HERSHEY_SIMPLEX, 0.7,
                        color, thickness=2)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    return img
```

```
capture = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
videoWriter = cv2.VideoWriter('video.avi', fourcc, 30.0, (640, 480))

while (True):
    ret, frame = capture.read()
    frame = F.to_tensor(frame)
    frame = showbbox(model, frame)
    frame = F.to_pil_image(frame)
    frame = np.array(frame)
    if ret:
        cv2.imshow('video', frame)
        videoWriter.write(frame)
    if cv2.waitKey(1) == 27:
        break

capture.release()
videoWriter.release()
```

datasetXmlConverter.py

```
import os
import os
import random
import xml.etree.ElementTree as ET

import matplotlib.pyplot as plt
import numpy as np
from PIL import Image, ImageDraw
from tqdm import tqdm
```

```
def extractInfoFromXml(xml_file):
    root = ET.parse(xml_file).getroot()

    infoDict = {}
    infoDict['bboxes'] = []

    for elem in root:
        if elem.tag == "filename":
            infoDict['filename'] = elem.text
        elif elem.tag == "size":
            image_size = []
            for subElem in elem:
                image_size.append(int(subElem.text))
            infoDict['image_size'] = tuple(image_size)
        elif elem.tag == "object":
            bBox = {}
            for subElem in elem:
                if subElem.tag == "name":
                    bBox["class"] = subElem.text

                elif subElem.tag == "bndbox":
                    for subSubElem in subElem:
                        bBox[subSubElem.tag] = int(subSubElem.text)
            infoDict['bboxes'].append(bBox)

    return infoDict


classNameToIdMapping = {"trafficlight": 0,
```

```
                "stop": 1,
                "speedlimit": 2,
                "crosswalk": 3}


def convertToYolov5(infoDict):
    printBuffer = []
    for b in infoDict["bboxes"]:
        try:
            classId = classNameToIdMapping[b["class"]]
        except KeyError:
            print("Invalid Class. Must be one from ", classNameToIdMapping.keys())

        b_center_x = (b["xmin"] + b["xmax"]) / 2
        b_center_y = (b["ymin"] + b["ymax"]) / 2
        b_width = (b["xmax"] - b["xmin"])
        b_height = (b["ymax"] - b["ymin"])

        image_w, image_h, image_c = infoDict["image_size"]
        b_center_x /= image_w
        b_center_y /= image_h
        b_width /= image_w
        b_height /= image_h
        printBuffer.append(
            "{} {:.3f} {:.3f} {:.3f} {:.3f}".format(classId, b_center_x, b_center_y,
b_width, b_height))
    save_file_name = os.path.join("Road_Sign_Dataset/annotations",
infoDict["filename"].replace("png", "txt"))
    print("\n".join(printBuffer), file=open(save_file_name, "w"))
```

```
annotations = [os.path.join('Road_Sign_Dataset/annotations', x) for x in os.listdir('Road_Sign_Dataset/annotations') if
        x[-3:] == "xml"]
annotations.sort()

for ann in tqdm(annotations):
    info_dict = extractInfoFromXml(ann)
    convertToYolov5(info_dict)
annotations = [os.path.join('Road_Sign_Dataset/annotations', x) for x in os.listdir('Road_Sign_Dataset/annotations') if
        x[-3:] == "txt"]

random.seed(0)

classIdToNameMapping = dict(zip(classNameToIdMapping.values(), classNameToIdMapping.keys()))



def plotBoundingBox(image, annotationList):
    annotations = np.array(annotationList)
    w, h = image.size

    plotted_image = ImageDraw.Draw(image)

    transformedAnnotations = np.copy(annotations)
    transformedAnnotations[:, [1, 3]] = annotations[:, [1, 3]] * w
    transformedAnnotations[:, [2, 4]] = annotations[:, [2, 4]] * h
```

```
    transformedAnnotations[:, 1] = transformedAnnotations[:, 1] -
(transformedAnnotations[:, 3] / 2)
    transformedAnnotations[:, 2] = transformedAnnotations[:, 2] -
(transformedAnnotations[:, 4] / 2)
    transformedAnnotations[:, 3] = transformedAnnotations[:, 1] +
transformedAnnotations[:, 3]
    transformedAnnotations[:, 4] = transformedAnnotations[:, 2] +
transformedAnnotations[:, 4]

    for ann in transformedAnnotations:
        obj_cls, x0, y0, x1, y1 = ann
        plotted_image.rectangle(((x0, y0), (x1, y1)))

        plotted_image.text((x0, y0 - 10), classIdToNameMapping[(int(obj_cls))])

    plt.imshow(np.array(image))
    plt.show()



# Get any random annotation file
for i in range(10):
    annotationFile = random.choice(annotations)
    with open(annotationFile, "r") as file:
        annotationList = file.read().split("\n")[:-1]
        annotationList = [x.split(" ") for x in annotationList]
        annotationList = [[float(y) for y in x] for x in annotationList]

    # Get the corresponding image file
    image_file = annotationFile.replace("labels", "images").replace("txt", "png")
    assert os.path.exists(image_file)
```

```
    # Load the image
    image = Image.open(image_file)

    # Plot the Bounding Box
    plotBoundingBox(image, annotationList)
```

gtsdbSplitToFolders.py

```
import os
import shutil

from sklearn.model_selection import train_test_split

dataset = 'gtsdb'
images = [os.path.join('%s/images' % dataset, x) for x in os.listdir(dataset) if x[-3:] == "ppm"]
annotations = [os.path.join('%s/labels' % dataset, x) for x in os.listdir('%s/labels' % dataset) if x[-3:] == "txt"]

images.sort()
annotations.sort()

train_images, val_images, train_annotations, val_annotations = train_test_split(images, annotations, test_size=0.2,
                                                                random_state=1)
val_images, test_images, val_annotations, test_annotations = train_test_split(val_images, val_annotations,
                                                            test_size=0.5, random_state=1)
```

```
def move_files_to_folder(list_of_files, destination_folder):
    for f in list_of_files:
        try:
            shutil.move(f, destination_folder)
        except:
            print(f)
            assert False


move_files_to_folder(train_images, 'images/train')
move_files_to_folder(val_images, 'images/val/')
move_files_to_folder(test_images, 'images/test/')
move_files_to_folder(train_annotations, 'labels/train/')
move_files_to_folder(val_annotations, 'labels/val/')
move_files_to_folder(test_annotations, 'labels/test/')
```

data.yaml

```
train: ../GTSDByolov5/train/images
val: ../GTSDByolov5/valid/images
test: ../GTSDByolov5/test/images

nc: 43
names: ['10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '6', '9', 'speed limit 100 (prohibitory)', 'speed limit 120 (prohibitory)', 'speed limit 20 (prohibitory)', 'speed limit 30 (prohibitory)', 'speed limit 50 (prohibitory)', 'speed limit 60 (prohibitory)', 'speed limit 70 (prohibitory)', 'speed limit 80 (prohibitory)']
```

hyp.scratch.yaml

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
```

```
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
copy_paste: 0.0
```

com.eyo.go.App.java

```
package com.eyo.go;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
import org.apache.commons.collections4.queue.CircularFifoQueue;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
```

```
import java.util.Map;
import java.util.Queue;
import java.util.Set;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

//python detect.py --source 0 --save-txt --weights runs/train/gtsdb17/weights/best.pt --conf 0.25 --name yolo_road_det
public class App extends Application {
    private static final Queue<Set<String>> queue = new CircularFifoQueue<>(5);
    private static Map<String, Long> signsToShow = new HashMap<>();
    private static boolean closed = false;

    public static void main(String[] args) throws IOException {
        WatchService watchService;
        try {
            watchService = FileSystems.getDefault().newWatchService();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        final Path directory = Paths.get("E:\\dyplom_master\\yolov5_my\\yolov5\\runs\\detect\\yolo_road_det15\\labels");
        while (!closed) {
            if (directory.toFile().exists()) break;
        }
        try {
            directory.register(watchService, StandardWatchEventKinds.ENTRY_CREATE);
```

```
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        final WatchService finalWatchService = watchService;
        Runnable runnable = () -> {
            while (true) {
                WatchKey key;
                try {
                    key = finalWatchService.poll(1, TimeUnit.SECONDS);
                    if (key == null) {
                        synchronized (queue) {
                            queue.add(Collections.emptySet());
                        }
                        continue;
                    }
                } catch (Exception e) {
                    break;
                }
                for (WatchEvent<?> event : key.pollEvents()) {
                    WatchEvent.Kind<?> kind = event.kind();
                    if (kind == StandardWatchEventKinds.OVERFLOW) {
                        continue;
                    }
                    Path newFilePath = directory.resolve((Path) event.context());
                    BufferedReader br;
                    try {
                        br = new BufferedReader(new
InputStreamReader(Files.newInputStream(newFilePath.toFile().toPath())));
                        Thread.sleep(20);
                        Set<String> records = new HashSet<>();
```

```
                while (true) {
                    if (!br.ready()) {
                        break;
                    }
                    String line = br.readLine();
                    records.add(line.substring(0, line.indexOf(" ")));
                }
                if (!records.isEmpty()) {
                    synchronized (queue) {
                        queue.add(records);
                    }
                }
            } catch (IOException | InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        key.reset();
    }
};
new Thread(runnable).start();
new Thread(() -> {
    while (!closed) {
        boolean changed = false;
        Map<String, Integer> countPerSign = new HashMap<>();
        synchronized (queue) {
            for (Set<String> recognized : queue) {
                if (recognized == null) continue;
                for (String sign : recognized) {
                    Integer integer = countPerSign.get(sign);
                    countPerSign.put(sign, integer == null ? 1 : integer + 1);
```

```
                }
            }
        }
        long endOfCycle = System.currentTimeMillis();
        Set<String> signsAppeared = countPerSign.keySet().stream().filter(sign ->
countPerSign.get(sign) > 4).collect(Collectors.toSet());
        for (String sign : signsAppeared) {
            if (!signsToShow.containsKey(sign)) {
                changed = true;
            } else {
                Long signTime = signsToShow.get(sign);
                changed = signsToShow.values().stream().anyMatch(aLong ->
signTime < aLong);
            }
            signsToShow.put(sign, System.currentTimeMillis());
        }
        for (Iterator<Map.Entry<String, Long>> iterator =
signsToShow.entrySet().iterator(); iterator.hasNext(); ) {
            Map.Entry<String, Long> next = iterator.next();
            if (signsAppeared.contains(next.getKey())) continue;
            if (endOfCycle - next.getValue() > 10_000) {
                changed = true;
                iterator.remove();
            }
        }
        if (changed) {
            Platform.runLater(App::updateImageViews);
            System.out.println("-------" + signsToShow.keySet() + "---------");
        }
        countPerSign.clear();
```

```
        }
    }).start();
    launch(args);
    closed = true;
    finalWatchService.close();
}

private static final List<ImageView> imageViews = new ArrayList<>();

private static void updateImageViews() {
    String[] strings = signsToShow.keySet()
            .stream()
            .sorted((o1, o2) -> signsToShow.get(o2).compareTo(signsToShow.get(o1)))
            .toArray(String[]::new);
    int i = 0;
    for (; i < strings.length; i++) {
        Image image = getImageForValue(strings[i]);
        ImageView imageView = imageViews.get(i);
        imageView.setImage(image);
    }
    for (; i < imageViews.size(); i++) {
        imageViews.get(i).setImage(null);
    }
}

@Override
public void start(Stage primaryStage) {
    HBox hbox = new HBox();
    hbox.setSpacing(10);
    for (int i = 0; i < 7; i++) {
```

```
            ImageView imageView = new ImageView();
            imageView.setFitWidth(100);
            imageView.setPreserveRatio(true);
            hbox.getChildren().add(imageView);
            imageViews.add(imageView);
        }
        StackPane root = new StackPane(hbox);
        primaryStage.setOnCloseRequest(event -> {
            Platform.exit();
        });
        Scene scene = new Scene(root, 800, 600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private static Image getImageForValue(String value) {
        StringBuilder sb = new StringBuilder("E:\\dyplom_master\\yolov5_my\\yolov5\\images\\");
        switch (value) {
            case "0":
                sb.append("10");
                break;
            case "1":
                sb.append("11");
                break;
            case "2":
                sb.append("12");
                break;
            case "3":
                sb.append("13");
```

```
        break;
    case "4":
        sb.append("14");
        break;
    case "5":
        sb.append("15");
        break;
    case "6":
        sb.append("16");
        break;
    case "7":
        sb.append("17");
        break;
    case "8":
        sb.append("18");
        break;
    case "9":
        sb.append("19");
        break;
    case "10":
        sb.append("20");
        break;
    case "11":
        sb.append("21");
        break;
    case "12":
        sb.append("22");
        break;
    case "13":
        sb.append("23");
```

```
    break;
case "14":
    sb.append("24");
    break;
case "15":
    sb.append("25");
    break;
case "16":
    sb.append("26");
    break;
case "17":
    sb.append("27");
    break;
case "18":
    sb.append("28");
    break;
case "19":
    sb.append("29");
    break;
case "20":
    sb.append("30");
    break;
case "21":
    sb.append("31");
    break;
case "22":
    sb.append("32");
    break;
case "23":
    sb.append("33");
```

```
            break;
        case "24":
            sb.append("34");
            break;
        case "25":
            sb.append("35");
            break;
        case "26":
            sb.append("36");
            break;
        case "27":
            sb.append("37");
            break;
        case "28":
            sb.append("38");
            break;
        case "29":
            sb.append("39");
            break;
        case "30":
            sb.append("40");
            break;
        case "31":
            sb.append("41");
            break;
        case "32":
            sb.append("42");
            break;
        case "33":
            sb.append("6");
```

```
        break;
    case "34":
        sb.append("9");
        break;
    case "35":
        sb.append("7");
        break;
    case "36":
        sb.append("8");
        break;
    case "37":
        sb.append("0");
        break;
    case "38":
        sb.append("1");
        break;
    case "39":
        sb.append("2");
        break;
    case "40":
        sb.append("3");
        break;
    case "41":
        sb.append("4");
        break;
    case "42":
        sb.append("5");
        break;
    default:
        return null;
```

```
        }
        return new Image(sb.append(".png").toString());
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.eyo.go</groupId>
  <artifactId>masterDyplom</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>11</source>
          <target>11</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <packaging>jar</packaging>

  <properties>
```

```
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-collections4</artifactId>
      <version>4.1</version>
    </dependency>
  </dependencies>
</project>
```