МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Магістерська робота

Асистент водія на основі штучних нейронних мереж

Виконав: студент групи ПМіМ-23с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

(підпис) Левкович Р.Ю.
(прізвище та ініціали)

Керівник (підпис) Колос Н.М.
(прізвище та ініціали)

Рецензент (підпис) Ковальчук О.В.
(прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики
Кафедра Дискретного аналізу та інтелектуальних систем
Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Притула М.М.
"31 "серпня 2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Левкович Роман Юрійович
(прізвище, ім'я, по батькові)

1 .Асистент водія на основі ШНМ

керівник роботи Колос Надія Мирославівна, кандидат фізико-математичних наук.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від **" 13" вересня 2022 року № 15**

2. Строк подання студентом роботи **12.12.2022р.**

3. Вихідні дані до роботи
стаття «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks»; документація PyTorch, OpenCV, giomm, Qt; стандарт C++20; документація AUTOSAR;

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)

Вступ; Глибинні нейронні мережі у роботі з фотографіями; Особливості розробки програмного забезпечення для автомобілів; Розробка нейронної мережі; Розробка асистента; Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 31 серпня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз існуючих рішень	01.09-06.09	вик
2.	Аналіз наукових статей та архітектури AUTOSAR	07.09-15.09	вик
3.	Підготовка середовища AUTOSAR	15.09-04.10	вик
4.	Розробка моделі нейромережі	05.10-25.10	вик
5.	Завершення розробки асистента	26.10-10.11	вик
6.	Оформлення роботи	10.11-01.12	вик

Студент _____ Левкович Р.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Колос Н.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська робота складається з вступу, п’яти розділів, висновків і списку використаних джерел. Загальний обсяг складає 44 сторінки, список використаних джерел містить шістнадцять найменувань, робота ілюстрована рисунками та таблицями.

У даній роботі створено модель для знаходження дорожніх знаків на фото, та розроблено систему розпізнавання знаків на основі архітектури AUTOSAR. Рішення складається системи з чотирьох застосунків, та використання моделі, натренованої на двох вибірках даних.

Дорожні знаки несуть групу важливої інформації, яку не можна отримати з інших джерел на дорозі. Це є обмеження швидкості, можливість паркуватись, пріоритет на дорозі, тощо. Розроблена модель аналізує 43 загальні дорожні знаки, і може використовуватись під час розробки автономних автомобілів, так і як незалежна система, яка повідомляє водію про дорожні знаки на панелі приборів.

Ключові слова: PyTorch, C++, вбудовані системи, Linux, нейромережі, системи штучного інтелекту.

Зміст

ВСТУП **3**

1 Огляд існуючих рішень **5**

2 Глибинні нейронні мережі у роботі з фотографіями **7**

2.1 Застосування нейронних мереж з фотографіями 7

2.2 Шари згорткових нейронних мереж 8

2.3 Задача локалізації об’єктів 10

2.4 Faster R-CNN 10

2.5 SSD 11

2.6 Порівняння метаархітектур 12

2.7 Класифікатори 12

2.8 MobileNet 13

2.9 SqueezeNet 14

2.10 Метрики роботи мережі 15

3 Особливості розробки програмного забезпечення для автомобілів . . **18**

3.1 ISO 26262 18

3.2 AUTOSAR 19

3.2.1 Adaptive AUTSOSAR Platform 20

3.2.2 Адаптивні додатки 21

3.3 MISRA 22

4 Розробка нейронної мережі **24**

4.1 Стек технологій 24

4.2 Параметри моделі 25

4.3 Вимоги до вибірки даних 26

4.4 Попередня обробка фото 27

4.5 Навчання нейромережі з GTSDB 27

4.6 Mapillary 28

5 Розробка асистента 32

5.1 AUTOSAR Core Types. AUTOSAR Logging 32

5.2 AUTOSAR Execution Management 34

5.3 Фреймворк побудови аплікацій 37

5.4 Розроблені додатки 38

5.4.1 Генератор даних 38

5.4.2 Розпізнавач знаків 39

5.4.3 Графічний інтерфейс 39

ВИСНОВКИ 40

ВСТУП

При розробці автомобілів велика увага приділяється безпеці всіх учасників дорожнього руху. Для цього створюються різного роду додаткові системи, які мають на меті зменшити вплив людського фактора під час керування. Всі ці системи разом складають удосконалену систему допомоги водію (ADAS — advanced driver-assistance system). Прикладами таких рішень є антиблокувальна система, гальмівний асистент, додаткові камери сліпих зон, тощо.

Кожна з таких систем є окремим, вбудованим програмним продуктом. І якщо на початку створення ADAS обчислювальні можливості були дуже обмеженими, то за останні десять років вбудовані пристрої стали достатньо потужні, щоб енергоефективно виконувати складні математичні обчислювання. Це дозволяє розширити арсенал доступних підходів, які можна застосувати для побудови розумніших асистентів.

Однією з таких важливих інтелектуальних систем, які потребують багато обчислень, є система розпізнавання дорожніх знаків. Її завданням є повідомляти водію про дорожні знаки, які знаходяться на дорозі. Вона є неоціненним додатком як для водіїв далекобійників, людей, які багато подорожують, так і для водіїв-початківців. Це пов’язано з тим, що керування автомобілем вимагає мультизадачності та уважності, а з часом зір людини втомлюється і втрачає пильність. Без такої системи водій може випадково пропустити якісь обмеження, і це призведе до аварійної ситуації.

Багато виробників автомобілів впровадили таку систему в свої автомобілі, як додатковий функціонал в моделях преміум класу, як-от Mercedes-Benz S class чи BMW 7 Series. Більше того, дані системи працюють лише з знаками обмеження швидкості.

Але дана система необхідна не тільки для водія. Багато автовиробників та технологічних гігантів зараз працюють над створенням автономних автомобілів, і дана система є надзвичайно важливою, оскільки це є єдиний спосіб дізнатись

такі параметри, як максимальна швидкість, можливість паркування, чи пріоритет на перехресті.

Тому метою даної роботи є розробка системи для знаходження та класифікації дорожніх знаків за допомогою штучних нейронних мереж. Предметом дослідження є підходи до аналізу зображень дорожніх знаків на основі нейромереж.

1 Огляд існуючих рішень

Задача розпізнавання дорожніх знаків вже тривалий час є в увазі дослідників. Щороку виходять дослідження, які стараються покращити теперішній стан. В даній роботі проаналізовано деякі з них.

Спершу розглянемо роботи [2, 3]. Обидва дослідження розглядають розробку андроїд додатків, які розв'язують дану задачу, але вони використовують різні підходи до даної задачі.

Підхід [3] полягає в використанні алгоритму на основі ознак для знаходження дорожніх знаків, і подальше застосування нейромережі-класифікатора. Дана робота використовує метод Віоли-Джонса для локалізації об'єктів. Це має переваги в тому, що метод локалізації знаків є доволі швидким, і він ніяк не впливає вибір класифікатора. Але в даного підходу є такі мінуси:

- Необхідно окремо навчати локалізатор і класифікатор;
- Обраний метод локалізації є бінарним, тому його необхідно запускати кілька разів, для кожної форми знаку (трикутної, круглої і ромба).

У той час робота [2] використовує сучасний підхід, де використовується тільки нейромережа, на виході якої є квадрат, де знаходиться дорожній знак, та його клас. Як архітектуру використано клас нейромереж YOLO, який вирізняється з інших популярних нейронних мереж для даної задачі. Дана нейромережа розроблена за допомогою спеціального фреймворку Darkness, та виділяє свої, спеціалізовані шари основ. Іншою перевагою даної роботи є використання технології Qt для розробки додатку, яка також популярна для розробки програмного забезпечення автомобілів. Тобто розроблений додаток можна модифікувати та використати в автомобілі.

Інша робота, на яку варто звернути увагу є [4], де розглянуто розробку асистентів для автосимулятора. У результаті розроблено окремі нейромережі, які розпізнають не тільки дорожні знаки, а й інші об'єкти, а саме дорогу та інші ав-

томобілі. Для кожної з цих задач застосовано підхід одної нейромережі на основі архітектури ResNet. Це є дуже глибока архітектура (понад 100 прихованих шарів), і тому для її роботи в режимі близькому до реального часу потрібно багато обчислювальних можливостей. Дана робота адресує це питання за допомогою встановлення вимог, а наявність саме CUDA-сумісної GPU від компанії Nvidida з мінімум 500 спеціалізованих ядер та 4 Гб пам’яті.

2 Глибинні нейронні мережі у роботі з фотографіями

2.1 Застосування нейронних мереж з фотографіями

Глибинні нейронні мережі — це системи, які розроблені для розв'язування певного типу задач. На відміну від класичних алгоритмів, де розробник програмує систему під завдання, нейронні мережі навчаються на групі прикладів.

Класичні глибинні мережі прямого поширення дозволяють апроксимувати довільну функцію, що дозволяє будувати різного роду системи, такі як оцінка платоспроможності, чи рекомендаційні системи.

Але в такого підходу є проблеми з структурованими даними, такими як зображення. У випадку класичної мережі, входом буде все зображення (тензор певного розміру), і до даних буде застосоване певне перетворення. Щоб така модель могла знаходити цільові об'єкти в різних місцях зображення, та з довільним розміром цільового об'єкта необхідно побудувати велику та збалансовану вибірку. Більше того, сама модель має бути великою, щоб вона могла розпізнати ці об'єкти. Очевидно, що з таким підходом задача стає дуже складною.

Для вирішення даної проблеми можна скористатись згортковими нейронними мережами, або іншими підходами машинного навчання. Згорткові нейронні мережі відомі з кінця 90-х, але навчання глибоких нейромереж є ресурсоємкою процедурою, тому вони мали доволі обмежене застосування. Зазвичай використовували невеликі нейронні мережі (до 10 згорткових шарів) для класифікації, або методи які працюють на основі ознак (такі як каскади Хаара), або метод опорних векторів.

Все змінилось на початку 2010-х, коли група дослідників успішно застосувала GPU для тренування нейромереж, та виграли серію змагань з класифікації зображень, та знаходження об'єктів. З цього часу найкращі підходи до розв'язування цих задач базуються саме на згорткових нейромережах. Розглянемо їх де-

тальніше.

2.2 Шари згорткових нейронних мереж

Особливістю згорткових нейронних мереж є введення додаткових шарів: операції згортки та агрегації.

Згортка — це процес додавання елемента зображення до його сусідів, зважених ядром. На рисунку 2.1 показано ілюстрацію одного кроку цього шару.

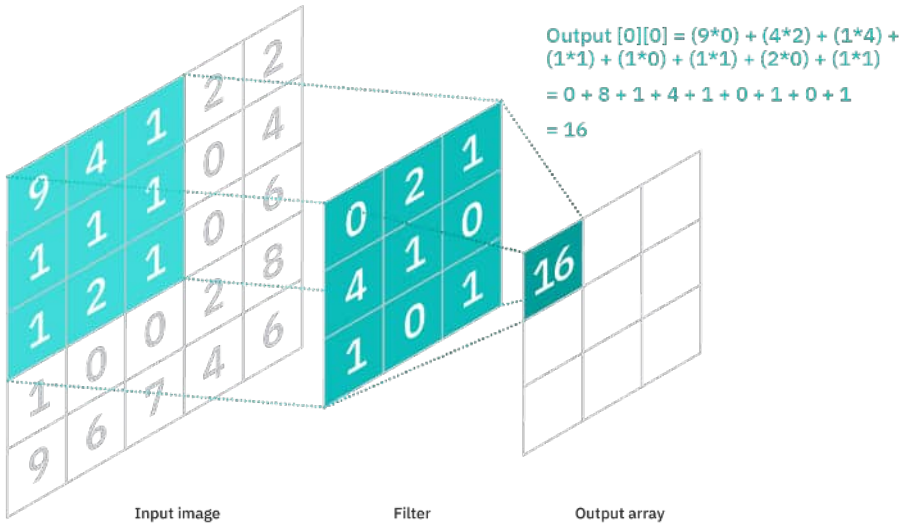

Рисунок 2.1: Ілюстрація оперції згортки

Згортка параметризується розміром ядра, кроком фільтру (який називають страйдом), та доповнення країв — додаткові пікселі по краях зображення.

Доповнення по краях необхідне для того, щоб контролювати розмір вихідного зображення. Якщо застосовувати згортку без нього, то вихідний тензор буде меншим від вхідного, що не завжди є бажаним результатом. З популярних підходів до заповнення є занулення доданих елементів або дзеркальне відображення.

Зазвичай, зображення є триканальними. Тому, щоб застосувати згортку до них, ядра об’єднуються в фільтр. Далі фільтр застосовується до зображення, результат застосування кожного ядра сумується поелементно і отримується один канал. Кожен фільтр відповідатиме одному каналу виходу. В нейромережах використовується багато фільтрів в межах одного шару. Зазвичай, зі зменшенням

розміру вихідної матриці збільшується кількість каналів.

Як видно з опису згорткового шару, він описує лінійне перетворення вхідного зображення. Тобто послідовне застосування кількох згорток підряд можна замінити одною згорткою. Тому до виходу з згорткового шару застосовують активаційну функцію. Зазвичай в ролі такої функції є зрізаний лінійний вузол (ReLU).

Ще одним важливим шаром, який застосовується у згорткових нейронних мережах є шар агрегації даних. В ньому виконується об'єднання кластеру нейронів з одного шару в інший за допомогою певної функції. Найбільш популярними прикладами є максимізаційне агрегування, яке використовує максимальне значення кожного виходу з кластера, і усереднене агрегування, що використовує середнє значення виходу з попереднього кластера. Візуальне зображення роботи даних шарів є на рисунку 2.2.

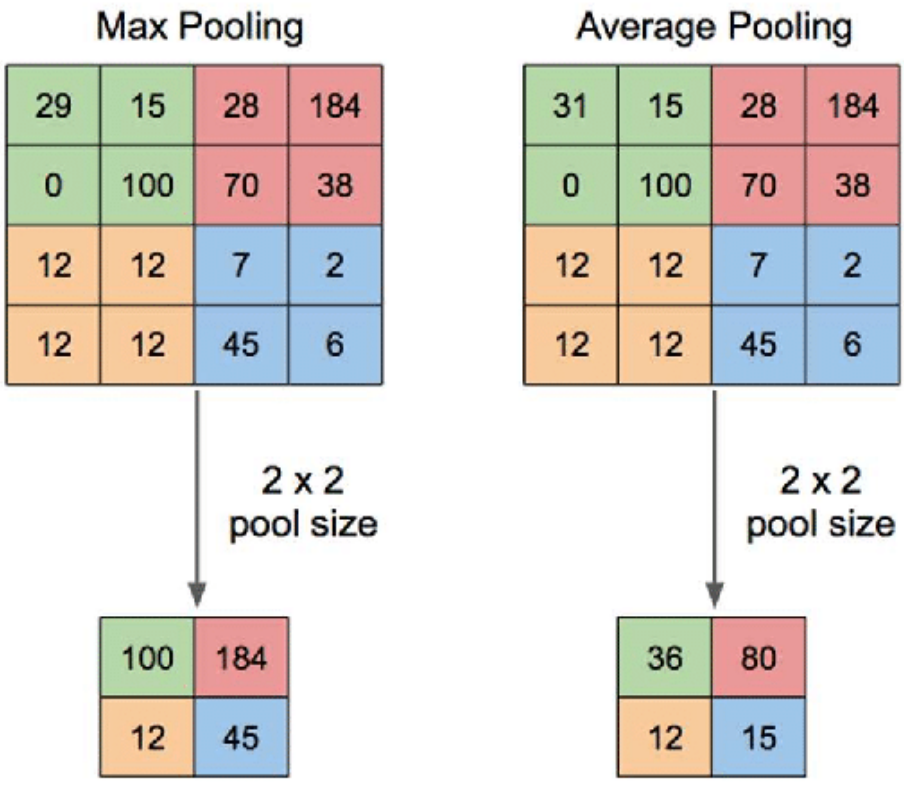

Рисунок 2.2: Ілюстрація оперції агрегації

2.3 Задача локалізації об’єктів

Задача локалізації об’єктів поєднує в собі дві задачі машинного навчання:

- Задача регресії: необхідно знайти регіони, де знаходяться шукані об’єкти;
- Задача класифікації: визначити, до якого класу належить об’єкт в пропонованій області;

Класично, дана задача розв’язувалась комплексно, на основі пошуку певних ознак. У той час застосування підходу з використанням регіонів, де може знаходитись об’єкт одного з шуканих класів, і подальша класифікація регіонів за допомогою глибоких згорткових нейронних мереж є доволі новим підходом, який вперше запропонували в [8].

Дане відкриття дало поштовх для створення нових класів ефективних нейронних мереж, таких як YOLO, SSD та ін. У даній роботі розглянуто моделі Faster R-CNN та SSD (Single Shot Detector). Дані архітектури можна вважати метаархітектурами, оскільки вони описують не одну мережу, а цілий клас нейронних мереж побудований за описаним підходом.

2.4 Faster R-CNN

Дана архітектура, запропонована дослідниками з Microsoft Research, [11], і є розвитком [8]. Даний підхід пропонує замість алгоритму selective search, який використовується для знаходження об’єктів, використати спеціалізовану підмережу.

В результаті, дана архітектура описує єдину глибоку нейронну мережу, яка складається з кількох основних компонентів, як це показано на рисунку 2.3.

В оригінальній роботі [11] розглядаються підходи, де окремо тренується кожна підмережа, або всі разом. Як виявлено дослідниками, другий підхід триває довше, але ніяк не впливає на точність.

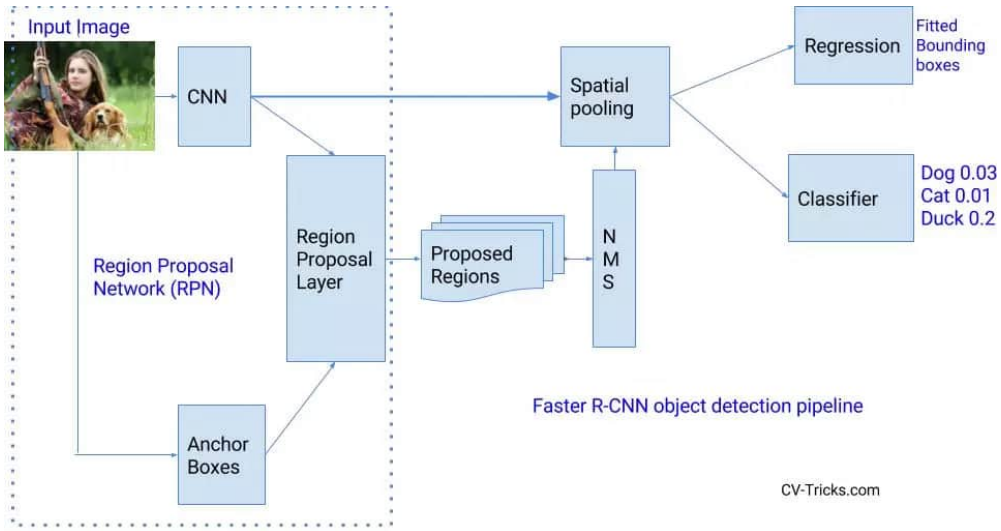

Рисунок 2.3: Архітектура Faster R-CNN

2.5 SSD

Іншим прикладом мета-архітектури нейронних мереж є Single Shot Detector. Її робота є схожою до архітектури обраної в [2]: класифікація і обчислення результату проходить в один крок, що і дало назву мережі.

Іншою особливістю даної мережі є застосування підходу ”стандартних обмежувальних прямокутників”. Даний метод використовує прив’язку групи обмежувальних прямокутників певного розміру до наперед обраних місць вхідного зображення, після чого уточнює положення фінальних регіонів на основі об’єднання інформації з виходів різних шарів мережі, з різними розмірами мапи ознак. Приклад архітектури зображено на риснку 2.4.

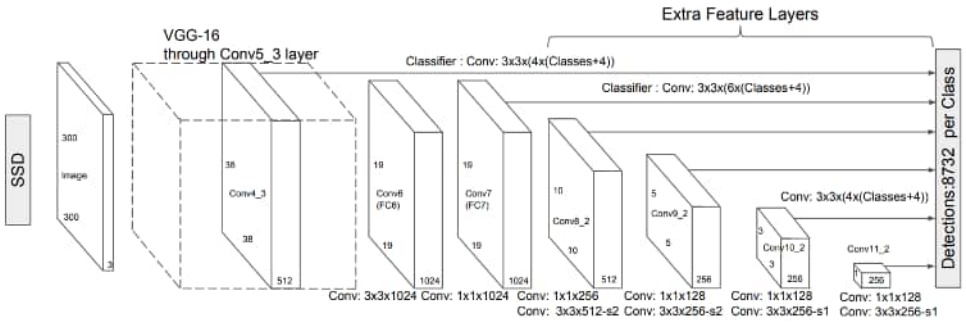

Рисунок 2.4: Приклад архітектури SSD з основою VGG-16

2.6 Порівняння метаархітектур

Як видно з рисунків архітектур, SSD є меншою, та значно більше залежить від архітектури основи. Відповідно, це вимагає зміни додаткових шарів залежно від обраної основи. Прикладом такої варіації є мережа SSDLite, яка розроблена на основі класифікатора MobileNet. Також, SSD намагається поєднати результати виходів з різних рівнів мережі основи, щоб мати можливість знаходити об’єкти всіх розмірів.

Проте, власне робота з невеликими об’єктами є проблемною для даної архітектури, і для вирішення цієї проблеми пропонуються різні підходи, як-от розробка додаткових шарів, максимально довге використання великих мап ознак, тощо.

У той час основною проблемою Faster R-CNN є швидкодія. Дана мережа є доволі великою, тому потребує багато обчислень. З іншого боку, вона показує кращу точність під час тестувань. Тому в даній роботі буде використано саме Faster R-CNN, як основну мережу для знаходження дорожніх знаків.

2.7 Класифікатори

Для того щоб краще визначати велику кількість класів, необхідно збільшувати кількість прихованих шарів глибиних мереж. Це призвело до появи дуже глибинних мереж типу Inception, чи ResNet, які складаються з більш ніж сотні

шарів. Окрім збільшення вимог до обчислювальних потужностей, такий підхід також навантажує когнітивні здібності розробників нейромереж.

Як вирішення даної проблеми пропонується розділення мережі на макроархітектуру, яка складається з окремих високорівневих модулів, та низькорівневих шарів. Самі модулі є комбінацією шарів нейронної мережі (операції згортки, агрегації, активації).

Ціллю даної роботи є запуск нейронних мереж на вбудованих пристроях з обмеженими обчислювальними ресурсами. Для вирішення даної задачі дослідниками розроблено кілька типів архітектур, найпопулярнішою з яких є MobileNet розроблена в Google, та SqueezeNet. Розглянемо особливості кожної з цих архітектур.

2.8 MobileNet

Дана архітектура є результатом роботи дослідників з Google Research [5].

Першою оптимізацією, яку застосували розробники, є відмова від згорток з великими ядрами (як-от, 5×5, 7×7 і більше). На їх місці застосовуються комбінації згорток 3×3 та 1×1, які мають меншу кількість параметрів, відповідно і операцій.

Іншим покращенням роботи мережі є заміна звичайної операції згортки на оптимізований, двокроковий алгоритм, який назвали ”поглиблена згортка”. Перший крок є застосуванням фільтру до кожного каналу окремо, і подальше їх об’єднання за допомогою згортки 1×1.

Також, для побудови мережі розробили спеціальний модуль, linear bottleneck, який спочатку збільшує розмірність входу, застосовує до нього активаційну функцію і зменшує кількість каналів на виході за допомогою 1×1 згортки. Графічне зображення цього блоку є на рисунку 2.5.

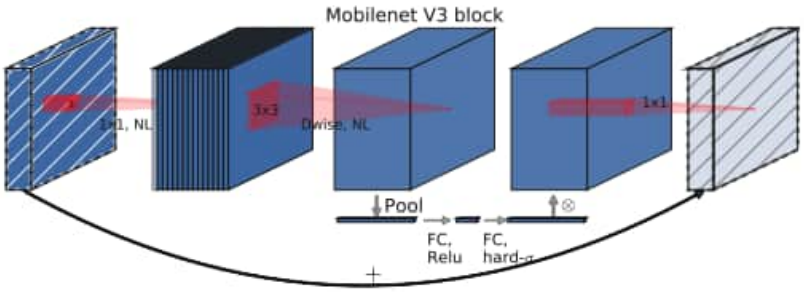

Рисунок 2.5: Одиничний блок MobileNetV3

2.9 SqueezeNet

Автори даної мережі також відмовились від великих у обчисленні згорток розміру 5×5, і використовувати тільки згортки з ядрами розміру 1×1 і 3×3. Також, для того, щоб зберегти більше інформації про вихідне зображення використовуються великі мапи ознак впродовж роботи мережі. Це досягається за допомогою мінімізації використання шарів агрегації даних.

Для швидшої роботи обчислень розроблено блок Fire, де використано підхід аналогічний до ”розрідженого блоку”в архітектурі Inception [15]: вихід з шару стиснення одночасно опрацьовується згортками з розміром ядра 1×1 і 3×3, та об’єднується в один вихід з блоку, як це зображно на 2.6.

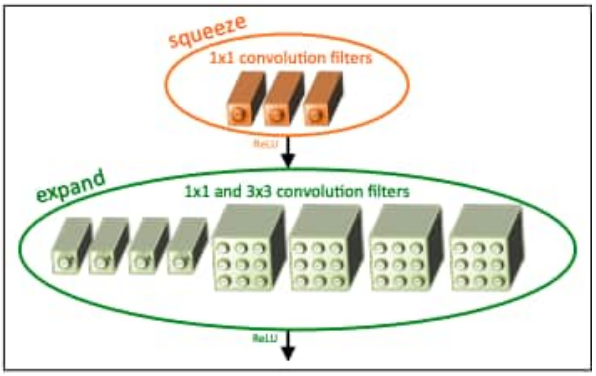

Рисунок 2.6: Модель Fire шару SqueezeNet

2.10 Метрики роботи мережі

Очевидно, що для нейромереж, які займаються класифікацією, метрикою буде відсоток правильно визначених зображень валідаційної вибірки.

З задачею знаходження об’єктів ситуація є складнішою, бо необхідно оцінити не тільки правильність класифікації, а й точність знаходження обмежувального прямокутника шуканого об’єкта.

Тому для оцінювання роботи нейромережі використаємо метрики, які пропонується на PASCAL VOC challenge. Вони базуються на основі таких параметрів як влучність, повнота та міра Жаккара.

Міра Жаккара (Intersection over Union, IoU) — це відношення спільної частини двох фото, до їх об’єднання. Візуально це зображається на рисунку 2.7.

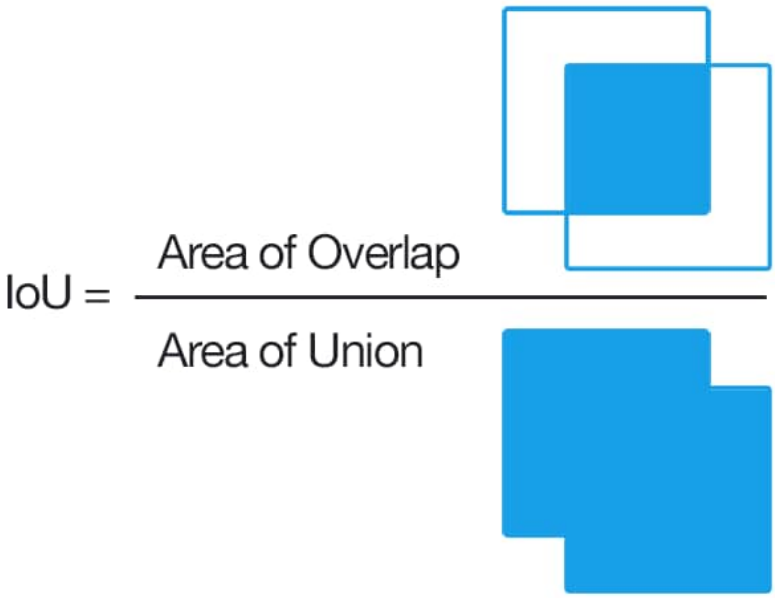

Рисунок 2.7: Візуалізація міри Жаккара

Влучність (Precision) — частка релевантних зразків серед знайдених. Повнота (Recall) — частка загального числа позитивних зразків, які було знайдено. Обчислення значень цих метрик найкраще видно на основі прикладів з матиці помилок, яка зображенна на рисунку 2.8.

		Справжній стан	
		позитивний стан	негативний стан
Прогнозований стан	позитивний прогнозований стан	істинно позитивний (ІП)	хибно позитивний (ХП) помилка I роду
	негативний прогнозований стан	хибно негативний (ХН) помилка II роду	істинно негативний (ІН)

Рисунок 2.8: Матриця помилок

Тоді значення метрик обчислюється так:

$$\text{Влучність} = \frac{\text{ІП}}{\text{ІП} + \text{ХП}} \tag{1}$$

$$\text{Повнота} = \frac{\text{ІП}}{\text{ІП} + \text{ХН}} \tag{2}$$

Основною метрикою роботи мережі є метрика PASCAL VOC:

- mAP@0.5 — середнє влучностей, якщо міра Жаккара більша за 50%;

Щоб детальніше оцінити роботу мережі скористаємось метриками з MS COCO (Common objects in context) вибірки:

- mAP@0.5 (small) — Середня влучностей, для міри Жаккара більшої 50% для пошуку об’єктів розміром до 32×32;
- mAP@0.5 (medium) — Середня влучностей, для міри Жаккара більшої 50% для пошуку об’єктів розміром від 32×32 до 96×96;
- mAP@0.5 (big) — Середня влучностей, для міри Жаккара більшої 50% для пошуку об’єктів розміром від 96×96;

- Recall (max=100) — Повнота знаходження до 100 об’єктів на зображенні;
- Recall (max=10) — Повнота знаходження до 10 об’єктів на зображенні;
- Recall (max=1) — Повнота знаходження до 1 об’єкта на зображенні;

3 Особливості розробки програмного забезпечення для автомобілів

Автомобіль — це об’єднання великої кількості вбудованих систем, які комунікують між собою. На кожну з них накладаються звичні вимоги для побудови вбудованих систем, такі як:

- Мінімальне споживання ресурсів;
- Робота в режимі реального часу;
- Виконання мінімально необхідної кількості функцій;

Додатково, для певних систем додаються вимоги підвищеного рівня безпеки. Це пов’язано з тим, що помилки, або не запуск певних підсистем, призводить до пошкоджень власності людей або нанесенні шкоди людями.

Для правильної оцінки ризиків, розробки програмного забезпечення високої якості, та зменшенні впливу людського фактору розроблено групу стандартів та підходів у розробці програмного забезпечення, серед яких варто виділити такі компоненти:

- ISO 26262 ”Functional safety — road vehicles”
- AUTOSAR
- MISRA C++ coding standard

3.1 ISO 26262

Даний стандарт є адаптацією IEC 61508 для розробки програмного забезпечення автомобілів. Він покриває повний процес розробки:

- Оцінка рівня ризику

- Документація
- Розробка апаратних рішень
- Розробка програмних рішень
- Особливості життєвого циклу впровадження і розробки програмного забезпечення.

Для оцінки рівня ризику вводиться чотири рівня Automotive Safety Integrity Levels, від A до D. Де A є найнижчим, і D — найвищем рівнем безпеки, та QM, який означає відсутність потреби в додаткових вимогах безпеки.

Присвоєння рівня відбувається на основі оцінки таких чинників:

- **Ймовірність події.** Даний чинник описує ймовірність того, що небезпечна подія відбудеться.
- **Контрольованість.** Даний чинник визначає можливість водієм керувати транспортом у випадку поломки системи.
- **Серйозність.** Даний чинник визначає важкість наслідків шкоди для людей та навколишнього середовища у випадку поломки системи.

Відповідно, рівень ASIL можна визначити як:

$$ASIL = \text{Серйозність} \times (\text{Контрольованість} \times \text{Ймовірність})$$

3.2 AUTOSAR

AUTOSAR (Automotive Open System Architecture) — це партнерство, яке об’єднує постачальників автомобільних компонетів, виробників автомобілів та компанії які розробляють програмне забезпечення з ціллю розробити єдину архітектуру автомобільного програмного забезпечення.

Зараз існує 2 версії AUTOSAR: Classic Platform та Adaptive Platform. Класична платформа описує найвищий рівень абстракцій, для запуску додатків на мікроконтроллерах. У свою чергу адаптивна платформа описує високорівневу, сервіс-орієнтовану архітектуру додатків.

У випадку AUTOSAR, не одна версія заміняє іншу, а вони створені, щоб доповнювати одна одну. Класична архітектура використовується для розробки вбудованих систем, які є частиною автомобіля, такі як антиблокувальна система, чи система запуску подушок безпеки. У той час адаптивна платформа розрахована для розробки інтелектуальних систем, типу круїз-контролю та інших.

Оскільки в даній роботі розглядається розробка розумного асистента водія, відповідно, робота затосунків відбуватиметься в межах Adaptive AUTOSAR.

3.2.1 Adaptive AUTSOSAR Platform

Розглянемо модель адаптивної платформи детальніше. На рисунку 3.1 показано основні компоненти даної архітектури.

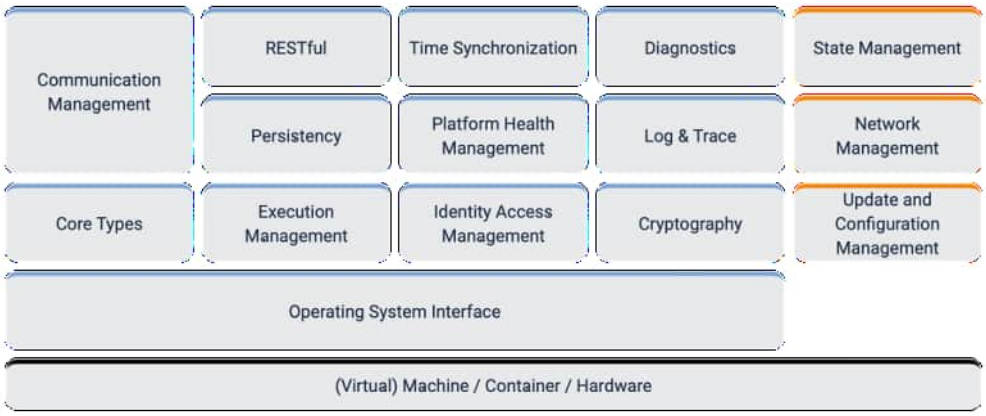

Рисунок 3.1: Модель архітектури Adaptive AUTOSAR

Як видно з рисунку 3.1, платформа працює в межах POSIX-сумісної операційної системи. В реальних автомобілях застосовується спеціалізовані, контейнеризовані embeded Linux системи на основі Yocto.

Також, вводиться уніфікований інтерфейс базових типів, логування та комунікації. Це важливо, тому що компоненти розробляються і тестуються різними

компаніями, і їм необхідний єдиний API, який вони використовують.

Нас тут цікавить Execution Management і State Management, оскільки вони описують не тільки програмний інтерфейс, а й окремі компоненти, які запускаються як частина адаптивної платформи.

Роботу платформи можна описати, як скінченний автомат, який виглядає як на рисунку 3.2.

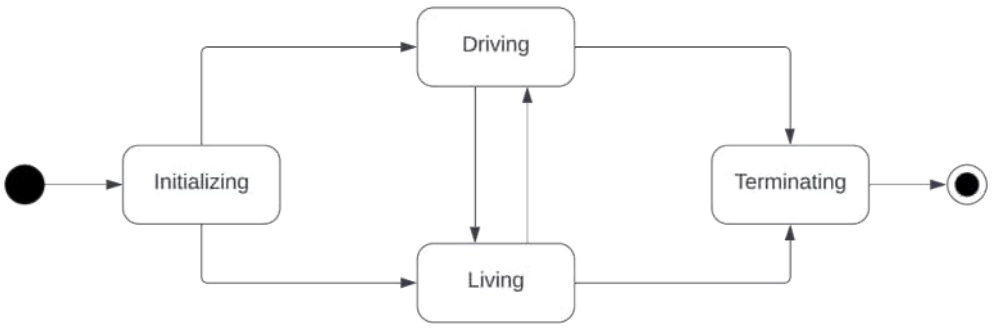

Рисунок 3.2: Основні стани адаптивної платформи

Робота State Manager полягає в тому, що він надає інтерфейс для отримання теперішнього стану, та запит на зміну стану. Сам State Manager визначає, чи можливий перехід в даний стан, сигналізує потребу зміни стану до Executaion Manager, та чекає на підтвердження переходу в новий стан від Execution Manager.

В той час Execution Manager є компонентом, який відповідальний за запуск всіх додатків платформи, та їх зупинку. Залежно від активного стану, Execution Manager запускає і зупиняє певну підмножину додатків.

3.2.2 Адаптивні додатки

Так само як платформа описується скінченним автоматом, кожен додаток теж можна описати як скінченний автомат, який складається з трьох станів, які зображені на рисунку 3.3.

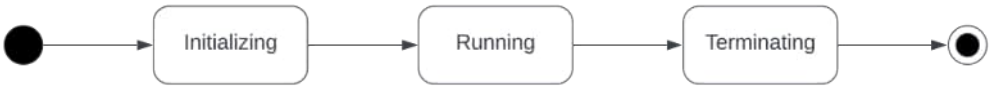

Рисунок 3.3: Скінченний автомат адаптивного додатку

Про перехід до кожного з станів додаток повідомляє до Exection Manager. Для того, щоб завершити роботу додатку, Exection Manager надсилає сигнал SIGTERM. Зазвичай, даний сигнал означає дострокове завершення роботи процесу, якому його надіслали, проте в даному випадку, він вказує, що застосунку необхідно надіслати перехід в стан Terminating та завершити роботу.

3.3 MISRA

MISRA — це об’єднання, яке займається розробкою вимог по написанню безпечного коду на C та C++ для застосунків з підвищеними вимогами до безпеки.

Якщо розглянути дані вимоги детальніше, то основне їх завдання це:

- Позбутись можливих неоднозначностей, які дозволяє стандарт C++
- Попередити можливі помилки розробників
- Допомогти спроектувати якісні системи

Станом на 2022 рік, MISRA все ще вимагає використання C++14 [6], та розробляє вимоги для C++17, хоча остання актуальна версія C++ - 20. В даній роботі використано саме останній стандарт C++, оскільки туди внесли групу змін, які важливі при розробці вбудованих систем:

- Концепти — перевірки етапу компіляції для типів, які використовуються в шаблонних класах та функціях;
- std::format — прийнята в стандарт бібліотека fmt, яка дає простий інтерфейс для форматування рядків з перевірками етапу компіляції;

- Оператор <=> — це універсальний оператор порівняння, який визначає всі операції порівняння для класу;
- std::optional, std::variant, std::span — важливі класи, які допомагають проектуати безпечні інтерфейси;
- бібліотека std::ranges — спрощення написання групи перетворень для контейнерів, що дозволяє зменшити кількість аллокацій пам’яті;
- Можливість робити аллокації пам’яті на етапі компіляції — створення констант рядків, векторів та вказівників, які визначаються на етапі компіляції.

Також, стандарт C++20 додає таку групу важливого функціоналу, який поки не реалізовано в компіляторах повністю:

- Співпрограми — узагальнення виклику функцій, що дозволяє розробляти високоефективні асинхронні додатки;
- Модулі — підтримка бібліотек мовою C++.

4 Розробка нейронної мережі

4.1 Стек технологій

Для імплементації нейронних мереж використано фреймворк PyTorch. Основними перевагами, що виділяють даний фреймворк поміж інших інструментів для глибинного навчання є:

- Це є фреймворк, тобто він дотримується певного підходу у всіх аспектах створення моделей: передача даних, створення моделей, тощо;
- PyTorch динамічно створює граф обчислень, що прискорює процес внесення змін до моделі;
- Широка підтримка спільноти з великою кількістю матеріалів та допоміжних бібліотек;
- Окремий C++ інтерфейс (libtorch) та можливіть перевикористати моделі створені в Python за допомогою допоміжного модуля torchscript;
- Підтримка ARM архітектур, які використовуються для побудови вбудованих пристроїв.

Щоб тренування даної мережі тривало розумний період часу необхідно використовувати графічні карти, з мінімум 4Гб пам’яті. Щоб отримати доступ до таких обчислювальних можливостей використано сервіс Google Collaboratory. Завдяки цьому моделі навчаються значно швидше, але час однієї сесії обмежений добою. Тому, для успішного застосування даного сервісу необхідно отримати правильний баланс між розміром вибірки, та можливістю швидко його розгорнути в даному середовищі.

4.2 Параметри моделі

Для кожного навчання мережі спільним був такий набір гіперпараметрів:

- Навчання відбувається впродовж 100 епох;
- Як оптимізатор використано функцію Adam;
- Коефіцієнт швидкості навчання є змінним, за допомогою косинусної нормалізації.

В ході роботи використано мережу SqueezeNet як основу, з конфігурацією, яка описана в таблиці 4.1.

Таблиця 4.1: Архітектура основи SqueezeNet

№	Шар	ядро/страйд	вхідний	пром. кан.	1x1 кан.	3x3 кан.
1	conv2	3x3/2				
2	maxpool2	3x3/2				
3	fire		64	16	64	64
4	fire		128	16	64	64
5	maxpool2	3x3/2				
6	fire		128	32	128	128
7	fire		256	32	128	128
8	maxpool2	3x3/2				
9	fire		256	48	192	192
10	fire		384	48	192	192
11	fire		384	64	256	256
12	fire		512	64	256	256

Архітектура підмережі генерації регіонів описано в таблиці 4.2.

Таблиця 4.2: Архітектура підмережі генерації регіонів

№	Шар	ядро/страйд	вихід. каналів
1	conv2	$3 \times 3/1$	512
2	ReLU		
3	conv2	$1 \times 1/1$	12
4	conv2	$1 \times 1/1$	48

Архітектура підмережі виходу з нейронної мережі зображена в таблиці 4.3, де класифікатор і регресор описують класичний повнозв'язний шар нейронної мережі.

Таблиця 4.3: Архітектура виходу з нейронної мережі

№	Шар	к-сть вихідних параметрів
1	ReLU	1024
2	ReLU	1024
3	Linear classifier	к-сть класів
4	Linear regressor	4·к-сть класів

4.3 Вимоги до вибірки даних

Дорожні знаки України стандартизовані ратифікацією Віденської конвенції про дорожній рух 1968-го року. Її також ратифікували більшість країн ЄС, Бразилія, та деякі країни Азії. Відповідно, для того, щоб обрана вибірка мала застосування на території України, необхідно вибрати дані з країн, які теж ратифікували дану конвенцію.

Але кожен знак ще має локалізацію, тому в даній роботі розглядаються дорожні знаки, які не мають написів.

4.4 Попередня обробка фото

Для збільшення різноманітності вибірки, розроблено додатковий пайплайн попередньої обробки фото. Перед тим, як фото передається на вхід нейронної мережі, до кожного фото з ймовірністю 50% може бути застосовані наступні трансформації:

- Згладжування
- Зміна контрастності
- Джиттер кольорів

4.5 Навчання нейромережі з GTSDB

GTSDB — це популярна вибірка з 1000 фото дорожніх знаків Німеччини за різних умов. Вона складається з 43 класів знаків, і є класичною вибіркою для знаходження об’єктів. Приклад фото даної вибірки зображено на 4.1.

Рисунок 4.1: Приклад фото з GTSDB

Запустимо тренування нейромережі на даній вибірці даних. Результат зображено в таблиці 4.4.

Таблиця 4.4: Результат навчання на GTSDB

mAP@0.5	42%
AR (max=10)	40%
mAP@0.5 (small)	30%
mAP@0.5 (medium)	64%
mAP@0.5 (big)	65%
AR (max=100)	40%
AR (max=1)	37%

Результат роботи моделі є доволі низьким, зважаючи на кількість параметрів. Можливою проблемою є розмір вибірки. Розглянемо ще одну, більшу вибірку даних.

4.6 Mapillary

Mapillary traffic sign dataset — це вибірка фотографій дорожніх знаків з цілого світу. Приклад фото з данної вибірки є на рисунку 4.2.

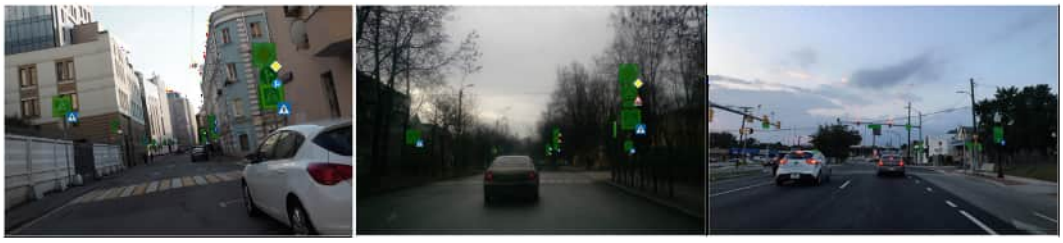

Рисунок 4.2: Приклад фото з Mapillary

Згідно з опублікованою статтею [7], розмір набору даних є 75 000 фотографій, з 400 класів дорожніх знаків. Але лише 40000 з них є повністю аннотованими. Також, частину вибірки становить тестовий набір даних, для якого відсутні публічні аннотації. Якщо розглянути тільки повністю аннотовану частину, то залишиться 30000 фото.

Велика кількість класів зумовлена їх різноманітністю. Кожен дорожній знак характеризується такими категоріями:

- Тип знаку (інформацій, заборонний, тощо)
- Назва самого знаку (головна дорога, стоп, тощо)
- Географія (з якої частини світу цей знак)

Відфільтрувавши знаки, розподіл класів виглядає як на рисунку 4.3.

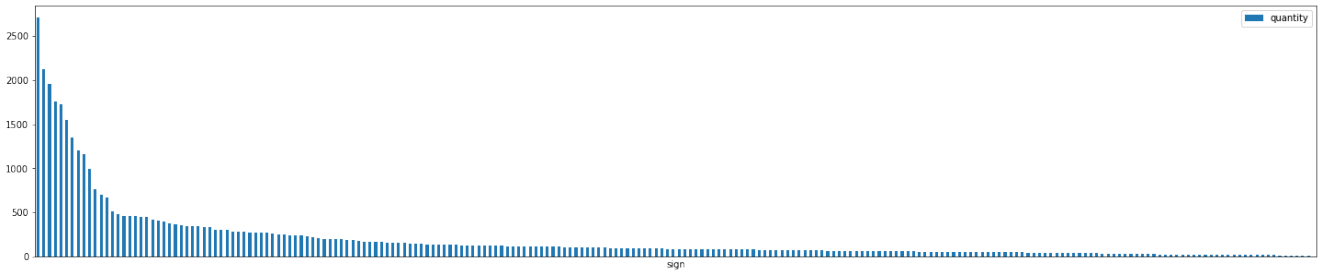

Рисунок 4.3: Розподіл знаків

Як видно з розподілу, кількість класів спадає експоненційно. Це погано впливатиме на результат навчання, оскільки багато класів збільшує розмір мережі, а мала кількість прикладів не дозволить отримати хорошу точність, як було в пункті 4.5. Також, дана вибірка є дуже великою і провести навчання за допомогою Google colaboratory буде дуже важко, тому проведемо навчання лише на частині вибірки.

Як видно з рисунку 4.3, в даній вибірці є група класів, де є понад 1500 прикладів. Це є знаки:

- Напрямок повороту праворуч
- Напрямок повороту ліворуч
- Головна дорога
- В’їзд заборонено

- Дати дорогу

Перші два знаки є інформаційними і не несуть важливої інформації, тому розглянемо тренування на 3 останніх дорожніх знаках. В результаті фільтрації отримано вибірку з 5000 фото.

Нейромережа після навчання показала результат, як у таблиці 4.5.

Таблиця 4.5: Результат навчання на найпопулярніших знаках mapillary

mAP@0.5	62%
AR (max=10)	41%
mAP@0.5 (small)	31%
mAP@0.5 (medium)	63%
mAP@0.5 (big)	35%
AR (max=100)	41%
AR (max=1)	30%

Це вже є кращий результат, який є на рівні з оригінальною роботою [11], де використовувалась значно глибша модель основи. Але 3 класи дорожніх знаків є невеликою вибіркою.

Застосуємо підхід finetuning щоб збільшити вибірку: замінимо шари моделі, які відповідальні за обчислення результатів з 3-х класів, які ми застосували в Mapillary на 44 класи GTSDB, і запустимо навчання знову. Результати наведено в таблиці 4.6.

Таблиця 4.6: Результат finetuning

mAP@0.5	73%
AR (max=10)	70%
mAP@0.5 (small)	58%
mAP@0.5 (medium)	70%
mAP@0.5 (big)	70%
AR (max=100)	65%
AR (max=1)	58%

Для перевірки стабільності моделі, перевіримо її роботу на перехресному затверджуванні методом виключення по 100. В результаті, середнє значення метрик записано в таблиці 4.7.

Таблиця 4.7: Результат перехресної перевірки

mAP@0.5	66%
AR (max=10)	70%
mAP@0.5 (small)	54%
mAP@0.5 (medium)	63%
mAP@0.5 (big)	65%
AR (max=100)	60%
AR (max=1)	47%

Як видно з таблиць 4.7, 4.6 та 4.4, попереднє навчання на великій вибірці з достатньою кількістю прикладів дозволяє отримати значно кращий результат.

5 Розробка асистента

Розробимо систему асистента водія, яка працюватиме на основі Adaptive AUTOSAR. Вона складатиметься з таких додатків

- Графічний інтефейс (GUI)
- Розпізнавач дорожніх знаків
- Генератор даних

Фінальна архітектура всієї системи зображенна на рисунку 5.1.

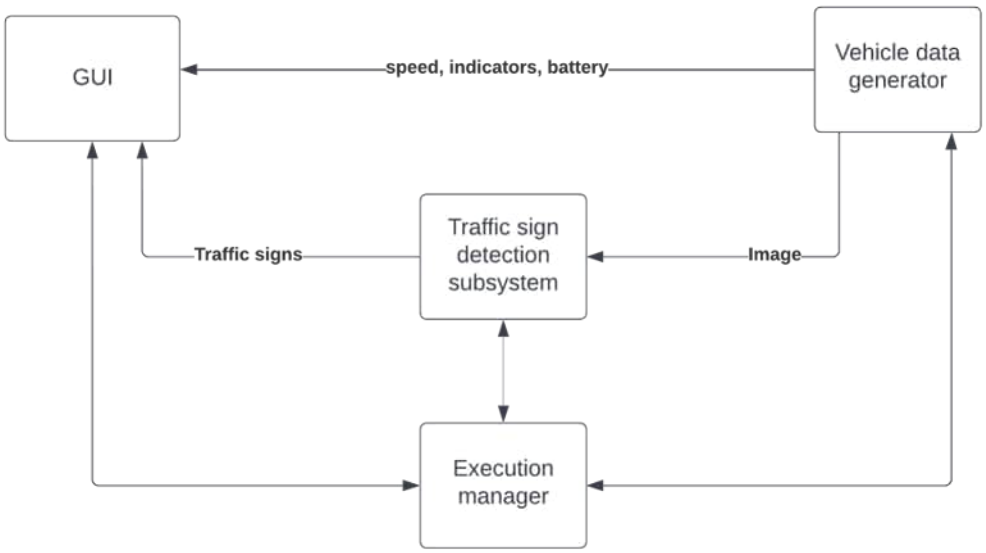

Рисунок 5.1: Архітектура системи

Стандарт AUTOSAR описує лише інтерфейси та функціональні вимоги до компонентів. Тому спершу розглянемо імплементацію та модифікацію необхідних специфікацій: Core Types [12], Logging [13] та Execution Management [14].

5.1 AUTOSAR Core Types. AUTOSAR Logging

У зв’язку з історичними причинами, C++ розробники часто не використовують стандартну бібліотеку. Тому, щоб AUTOSAR могли використовувати всі розробники, необхідно ввести набір обов’язкових типів: array, vector, map, optional,

variant, string, span, future, promise. Кожен з них повинен мати інтерфейс, який відповідає тому, що описаний в стандартній бібліотеці. В даній ситуації, без обмеження загальності, можна використати типи стандартної бібліотеки (STL).

Іншим важливим елементом роботи додатку є логування. AUTOSAR описує роботу логера, як глобальний сінглтон, який надає окремі методи для надсилання повідомлень різних рівнів критичності. Кожне повідомлення повинне включати такі додаткові елементи: ідентифікатор додатку, час надсилання повідомлення, рівень важливості повідомлення.

Також, логер повинен давати можливіть записувати дані в різні місця серед яких:

- Файл
- Термінал
- Віддалений сервер

Для побудови повідомлень пропонується спеціальний об’єкт типу LogStream, який поводиться по аналогії з std::cout. Даний підхід має 2 проблеми:

1. Щоб залогувати інформацію про об’єкт певного класу необхідно додати спеціалізацію оператору « для класу LogStream, і окремо робити спеціалізацію для std::ostream;
2. Робота з потоками даних є неефективною в C++.

Альтернативою для роботи з потоками є форматування рядків. У C++14 є тільки один спосіб робити форматування рядків — синтаксис printf, але він є нетривіальним у використанні, та в ньому відсутні перевірки етапу компіляції. Тому його доволі рідко використовують. Натомість в C++20 додали функцію std::format, яка має синтаксис схожий до printf, але вона дозволяє використовувати користувацькі типи, і має перевірки етапу компіляції.

Тому, в даній роботі для логування використано бібіліотеку spdlog, яка відповідає загальній ідеї роботи логування, яка пропонується в AUTOSAR, але працює значно швидше.

5.2 AUTOSAR Execution Management

Як описано в розділі 3.2, вся система описується як скінченний автомат, де в кожному стані працює певний набір додатків. Для визначення списку застосунків, які необхідно запустити, Execution Manager повинен прочитати машинний маніфест, де описано список наявних станів, та маніфест кожного застоснуку окремо.

AUTOSAR описує кожен маніфест як файл формату arxml, який є звичайним xml документом. Відповідно до вимог, при побудові документу використовуються тільки значення в тегах, без атрибутів, а типізація параметрів відбувається за допомогою спеціалізованих тегів. У результаті отримується багатослівний документ, де важко отримати необхідну інформацію. Для того, щоб спростити побудову документу, використаємо формат JSON with comments для створення конфігураційних файлів.

Для контролю за станами, запуском додатків та їх моніторингу створено 3 окремих компоненти: Execution Manager, State Manager, Watchdog. У кожного з цих компонентів є певна відповідальність: запуск інших процесів, зберігання загального стану системи, моніторинг інших процесів.

Для побудови системи на основі AUTOSAR можна використати два підходи: де кожен компонент є окремим застосунком, або об’єднати їх в один процес. Кожен з цих підходів має свої плюси і мінуси.

Плюси першого підходу:

- Розділення відповідальності процесів;
- Дозволяє будувати систему де менше залежностей між компонентами;
- Ізоляція дозволяє розділити розробку кожного компонента між окремими

командами розробників;

Мінуси першого підходу:

- Збільшується кількість коду, який необхідно підтримувати;
- Необхідно вводити спеціальні типи аплікацій та керувати їх реєстраціями;
- Істотно збільшується кількість міжпроцесної комунікації;
- У випадку критичної зупинки одного з системних додатків система все одно не може нормально існувати.

У той час плюси другого підходу:

- Єдиний основний процес, який легше моніторити;
- Зовнішні виклики заміняються методами;
- Запуск застосунків відбувається швидше, бо відсутні реєстрації спеціальних додатків.

Мінуси другого підходу:

- Аварійна зупинка одного компонента одразу впливає на інші;
- Робота монолітного додатку ускладнюється через потребу синхронізації потоків.

Зваживши за і проти обох підходів, у даній роботі використано другий підхід, оскільки в даного проекту лише один розробник, та у випадку аварійної зупинки одного з системних сервісів система все одно повинна перезапускатись.

Для зовнішньої комунікації розроблено відповідні інтерфейси, які описано в [14]. AUTOSAR вимагає використання POSIX викликів для комунікації. Основними підходами для побудови комунікацій є пайпи та сокети. Кожен з цих підходів має свої плюси та мінуси, але їх об’єднує один недолік: це низькорівневі примітиви, транспортного рівня моделі OSI, які передають масиви даних, без певного

формату. У той час протокол рівня застосунку залишається на розгляд вендора, з вимогою надання RPC інтерфейсу. Тому в даній роботі використано DBus для комунікацій між процесами.

Даний механізм побудований на основі сокетів, та є де-факто стандартом для міжпроцесних комунікацій в розробці користувацьких застосунків з графічним інтерфейсом в Linux. Для C++ існує кілька основних підходів для роботи з DBus:

1. libdbus
2. Qt
3. giomm

Перша бібліотека надає низькорівневий інтерфейс, і не рекомендується для використання в застосунках розробниками.

Qt — це фреймворк, основне використання якого це побудова графічних інтерфейсів, але додатково він надає можливість взаємодіяти з великою кількістю інших речей, таких як міжпроцесна комунікація, взаємодія з базами даних, тощо. Використання даного фреймворка вимагає багато додаткових витрат, бо він використовує велику кількість автогенерованого коду, який необхідний для правильної роботи фреймворку.

Giomm — це C++ бібліотека, розроблена спільнотою GTK для комунікації між процесами в графічних оболонках Linux, таких як GNOME, Cinnamon, LXDE, тощо. Для роботи даної бібліотеки необхідно створити додатковий пул потоків, який використовуватиметься для обробки вхідних повідомлень, і це всі додаткові витрати на використання цієї бібліотеки. Тому в даній роботі використано саме її.

Для імплементації Execution manager, кожен з компонентів є окремим об’єктом, який існує впродовж всього життя процесу. У кожного об’єкта є свій робочий

потік, та публічний інтерфейсу для комунікації з іншими об□єктами. Фінальна архітектура рішення зображена на рисунку 5.2.

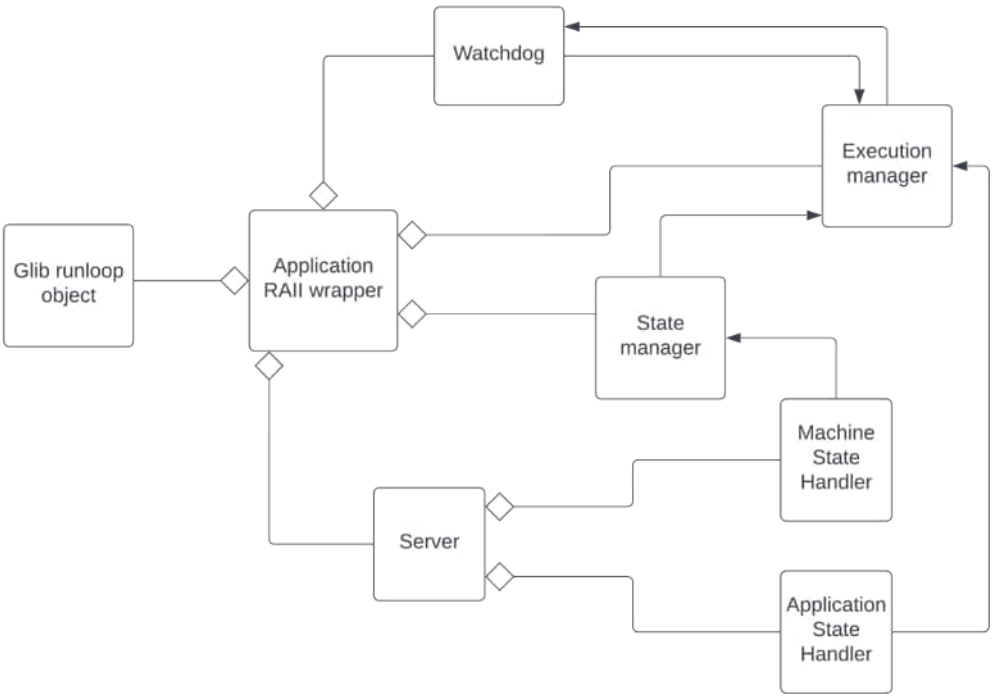

Рисунок 5.2: Архітектура Execution Manager

5.3 Фреймворк побудови аплікацій

Як було проаналізовано в 3.2, кожен додаток є скінченним автоматом, та має спільну поведінку. Винесемо всі необхідні спільні компоненти в один фреймворк, який надаватиме кістяк для побудови додатків, а кожен окремий застосунок буде спеціалізовувати необхідну поведінку.

Фреймворк складається з таких основних компонентів:

- Глобальний клас застосунку;
- Контейнер об’єктів середовища;
- Базовий клас об’єкту стану;
- Об’єкт машини станів.

Щоб уникнути витоків пам’яті, та правильно звільняти всі ресурси, якими користується додаток використано підхід RAII, тому весь життєвий цикл програми відбувається в межах класу Application, який в конструкторі ініціалізовує необхідні компоненти, і далі їх звільняє в деструкторі. Якщо під час роботи програми вона аварійно завершиться через можливу необроблену виняткову ситуацію, то все одно всі ресурси правильно звільняться.

Також, якщо додатку необхідно виконати додаткові кастомізації на різних етапах життя додатку, надається окремий клас ApplicationDelegate, в якому можна додати поведінку, яка виконається на таких етапах:

- Під час ініціалізації програми
- Перед запуском Running State,
- Після роботи Running State
- Під час деініціалізації

Для створення додатку необхідно передати об’єкт машини станів, та Application Delegate.

5.4 Розроблені додатки

5.4.1 Генератор даних

Даний сервіс імітує роботу сенсорів автомобіля, і генерує дані, які б генерував автомобіль під час руху: швидкість, роботу індикаторів, заряд батареї. Також, він генерує дані, які відповідають отриманню даних з передньої камери автомобіля. Цю функцію виконує відео, яке передається параметром командного рядка на старті сервісу.

Генератор даних є прикладом класичного сервісу-постачальника, який надає дані для довільної кількості споживачів. Тому методи публічного інтерфейсу створені за підходом публікації-підписка.

5.4.2 Розпізнавач знаків

Даний сервіс використовує модель, яку навчили в розділі 4, скомпільовану за допомогою torchscript, та на основі вхідного зображення отримує дорожні знаки, які знаходяться на дорозі, і передає їх до графічного інтерфейсу.

Модель Faster RCNN є великою, і обробка на CPU займає значний період часу. Тому даний сервіс пропускає частину зображень, беручи на вхід зображення, яке прийшло одразу після того, як обробили попереднє.

5.4.3 Графічний інтерфейс

Для розробки графічного інтерфейсу використано фреймворк Qt та мову QML для створення розмітки елементів. Даний сервіс працює впродовж всього життя системи, та є класичним прикладом споживача: він підписується на повідомлення від генератора даних, та розпізнавача знаків, і показує їх водію.

Вигляд графічного інтерфейсу зображено на рисунку 5.3.

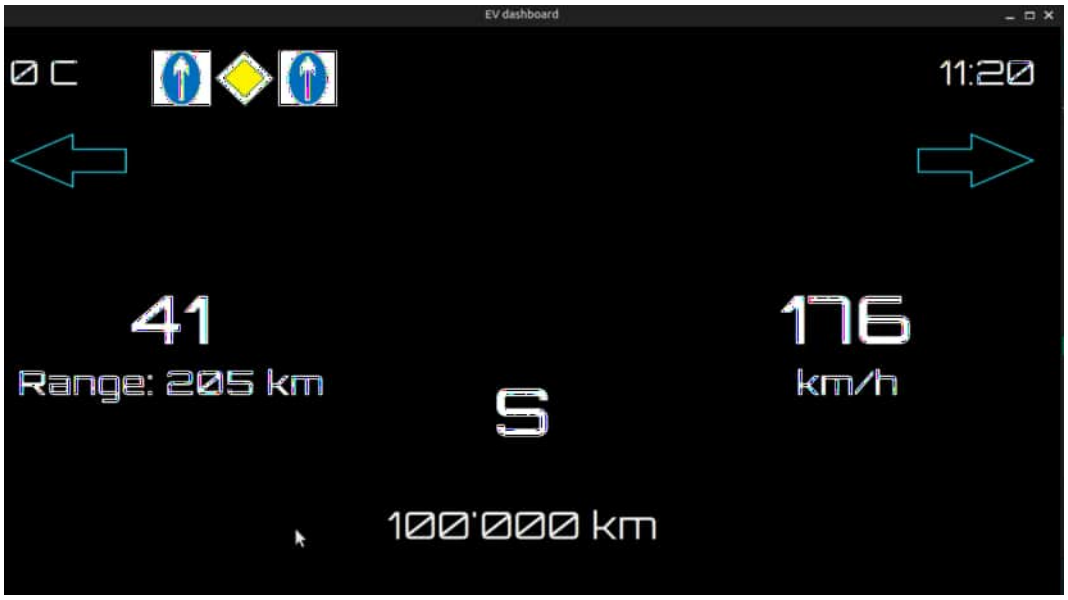

Рисунок 5.3: Вигляд графічного інтерфейсу

ВИСНОВКИ

В ході даної роботи розглянуто розробку системи розпізнавача дорожніх знаків автомобіля. Перед виконанням завданння досліджено інші наукові дослідження на дану тему, та оцінено використані підходи.

Для розв'язування завдання розглянуто сучасний підхід до знаходження об'єктів за допомогою згорткових нейронних мереж. Проаналізовано популярні архітектури які використовуються для даної задачі, та зосереджено увагу на архітектурі Faster R-CNN з основою SqueezeNet, як оптимальне рішення для даної проблеми.

В ході роботи виконано навчання нейронної мережі за допомогою двох вибірок даних: GTSDB та Mapillary. Основною метрикою роботи мережі обрано метрику PASCAL VOC, і після перехресної перевірки влучність мережі становить 66%.

Проведено детальний огляд особливостей розробки програмного забезпечення для автомобілів. Як основу архітектури обрано AUTOSAR, оскільки це відкрита система, яка використовується в автомобілях. Також, під час аналізу запропоновано використати JSON як формат конфігураційних файлів, та C++20 як основний стандарт, що дало такі переваги:

- Менші конфігураційні файли
- Прямолінійна робота з конфігураційними файлами
- Покращилась виразність коду
- Вдалось зменшити складність коду за допомогою нового функціоналу

Додатково, запропоновано використання DBus як системи комунікацій рівня застосунків, та розроблено фреймворк для швидкої розробки адаптиних додатків AUTOSAR.

Підсумком розробки системи є такі результати:

- Розроблено систему розпізнавання дорожніх знаків;
- Розроблено мінімально життєздатну симуляцію роботи автомобіля.

Цю систему в подальшому можна використовувати під час інших досліджень розробки інтелектуального програмного забезпечення для автомобілів.

Література

[1] Левкович Р. РОЗРОБКА АСИСТЕНТА ВОДІЯ НА ОСНОВІ ШНМ / Левкович Р. // XV International Scientific and Practical Conference (12-14 жовтня, м.Ванкувер, Канада) — с.101 - 107. Режим доступу: https://sci-conf.com.ua/xv-mizhnarodna-naukovo-praktichna-konferentsiya-innovations-and-prospects-of-world-science-12-14-10-2022-vankuver-kanada-arhiv/

[2] Бабій, Ю.В. Алгоритми розпізнавання дорожніх знаків для мобільних пристроїв : магістерська дис. : 123 Комп'ютерна інженерія / Бабій Юрій Васильович. — Тернопіль, 2020. — 86с. [Електронний ресурс] – [Цит. 2022, жовтеньз]. – Режим доступу: http://dspace.wunu.edu.ua/handle/316497/40539

[3] Фастовець, Є. Р. Система розпізнавання дорожніх знаків на основі нейронної мережі : дипломна робота бакалавра : 6.050103 Програмна інженерія / Фастовець Євгеній Русланович. – Київ, 2019. – 69 с. [Електронний ресурс] – [Цит. 2022, жовтень]. – Режим доступу: https://ela.kpi.ua/handle/123456789/28820

[4] Чапалюк, М. В. Застосування глибинних нейронних мереж для розпізнавання образів при навчанні в автосимуляторах : магістерська дис. : 122 Комп'ютерні науки / Чапалюк Максим Володимирович. - Київ, 2019. - 87 с. [Електронний ресурс] – [Цит. 2022, жовтень]. – Режим доступу: https://ela.kpi.ua/handle/123456789/32192

[5] Andrew Howard: Searching for MobileNetV3 / Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam — [Цит. 2022, жовтень]. — Режим доступу: https://arxiv.org/abs/1905.02244

[6] AUTOSAR C++14 code guidelines [Електронний ресурс] – [Цит. 2022, листопад]. – Режим доступу

https://www.autosar.org/fileadmin/user_upload/standards/adaptive/18-10/AUTOSAR_RS_CPP14Guidelines.pdf

[7] Ertler E. The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale / Christian Ertler, Jerneja Mislej, Tobias Ollmann, Lorenzo Porzi, Gerhard Neuhold, Yubin Kuang — 2020 – [Цит. 2022, жовтень]. – Режим доступу: https://arxiv.org/1234.5123

[8] Gradient Based Leadning Applied to Document Recognition / Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haffner — 2018 — [Цит. 2022, жовтень] — Режим доступу: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

[9] Houben S. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark / Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing and Christian Igel — 2013 – [Цит. 2022, листопад] – Режим доступу: https://benchmark.ini.rub.de/gtsdb_news.html

[10] Iandola F. N. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size / Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer – [Цит. 2022, жовтень]. – Режим доступу: https://arxiv.org/abs/1602.07360

[11] Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun [Електронний ресурс] – [Цит. 2022, жовтень]. – Режим доступу: https://arxiv.org/abs/1506.01497

[12] Specification of Core Types for Adaptive Platform [Електронний ресурс] – [Цит. 2022, листопад]. – Режим доступу https://www.autosar.org/fileadmin/user_upload/standards/adaptive/19-11/AUTOSAR_SWS_CoreTypes.pdf

[13] Specification of Log and Trace for Adaptive Platform [Електронний ресурс] – [Цит. 2022, листопад]. – Режим доступу https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_SWS_AdaptiveLogAndTrace.pdf

[14] Specification of Execution Management [Електронний ресурс] – [Цит. 2022, листопад]. – Режим доступу https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_SWS_ExecutionManagement.pdf

[15] Szegedy C. Going deeper with convolutions / Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich — 2014 – [Цит. 202, листопад]. – Режим доступу: https://arxiv.org/abs/1409.4842

[16] Working Draft, Standard for Programming Language C++ [Електронний ресурс] — 2022 – [Цит. 2022, листопад]. – Режим доступу: https://eel.is/c++draft/