Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ПРОГРАМНО–АПАРАТНИЙ КОМПЛЕКС ДЛЯ АВТОПІЛОТАЖУ ЗАСОБАМИ НЕЙРОМЕРЕЖЕВОГО МОДЕЛЮВАННЯ ТА ГЕНЕТИЧНИХ АЛГОРИТМІВ

Виконав: студент 2 курсу, групи 2КІ-23м
спеціальності 123 – «Комп'ютерна інженерія»
освітня програма – Комп'ютерна інженерія
Хорошун В.Р.
(прізвище та ініціали)

Керівник: к.т.н., проф.
Азарова А.О.
(прізвище та ініціали)

Опонент: завідувач кафедри МБІС
к.т.н., доцент Карпінець В. В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«___» _____2024 р

Вінниця ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій та комп'ютерної інженерії

Кафедра Обчислювальної техніки

Галузь знань 12 - Інформаційні технології

Освітній рівень - магістр

Спеціальність 123 - «Комп'ютерна інженерія»

Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф Азаров О.Д.

« 18 » вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Хорошуну Владиславу Руслановичу

1 Тема проєкту «Програмно–апаратний комплекс для автопілотажу засобами нейромережевого моделювання та генетичних алгоритмів», керівник проекту Азарова А. О., к.т.н., проф, затверджені наказом вищого навчального закладу від «18» вересня 2024 р. №310.

2 Строк подання студентом проєкту «25» грудня 2024 р.

3 Вихідні дані до проєкту: існуючі технічні засоби для реєстрування відеоінформації, вхідні дані з камер за допомогою DirectX 11, максимальна кількість оброблюваних кадрів — 20 на секунду, використання моделі YOLO v6 для оброблення зображень, нейромережевий аналіз за допомогою багатошарового перцептрону, оптимізація навчання за допомогою генетичних алгоритмів.

4 Текстова частина роботи містить вступ, огляд актуальності автоматизованого керування транспортом із використанням нейромереж, аналіз існуючих підходів до створення таких систем, вибір технологій і методів для розроблення, опис процесу створення програмно-апаратного комплексу, рекомендації щодо впровадження системи в експлуатацію та економічний аналіз.

5 Перелік графічного матеріалу: Архітектура YOLOv6, програма сервер для отримання даних з камер, штучний нейрон, схема архітектури багатошарового перцептрона, інтерфейс програми автопілоту.

6 Консультанти розділів роботи наведені в табл. 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Азарова А. О., к.т.н., проф		
5	Небава М.І., к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання « 18 » вересня 2024 року

8 Календарний план виконання приведений в табл. 2.

Таблиця 2 — Консультанти роботи

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	20.09.2024	виконано
2	Огляд інформаційних джерел	21.09.2024	виконано
3	Огляд і аналіз штучного інтелекту	28.09.2024	виконано
4	Розробка програмно апаратного комплексу для забезпечення автоматизованого керування автомобілем	10.10.2024	виконано
5	Підготовка матеріалів та опис розробки	01.11.2024	виконано

6	Оформлення пояснювальної записки та ілюстративного матеріалу	05.11.2024	виконано
7	Оцінка комерційного потенціалу розробки	10.11.2024	виконано
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	12.11.2024	виконано

Студент _____ Хорошун В.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Азарова А.О.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____ Небава М.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004

Хорошун В. Р. Програмно–апаратний комплекс для автопілотажу засобами нейромережевого моделювання та генетичних алгоритмів. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, Вінниця: ВНТУ, 2024 — 147 с.

На укр. мові. Бібліогр. 12 назв; рис.: 25; табл. 10; ліст. коду 5.

У дипломній роботі розглянуто теоретичні основи та практичні аспекти створення програмно-апаратного комплексу для автоматизованого керування автомобілем. Проведено аналіз нейромережевих підходів та генетичних алгоритмів, які використовуються для забезпечення ефективного моделювання та прийняття рішень у реальному часі.

Запропоновано комплексний підхід до поєднання методів штучного інтелекту для управління автомобілем у динамічному середовищі, що включає розпізнавання об’єктів, ухвалення рішень та адаптивне керування. Результати дослідження показали, що створений комплекс дозволяє значно підвищити точність та швидкість реагування системи, а також забезпечити безпечне та ефективне керування транспортним засобом.

Ключові слова: програмно-апаратний комплекс, автоматизоване керування, нейромережі, генетичні алгоритми, штучний інтелект.

ABSTRACT

Khoroshun V.R. Software and Hardware Complex for Automated Vehicle Control Using Neural Network Modeling and Genetic Algorithms. Master's qualification work in specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2024 — 134 pages.

The master's thesis explores the theoretical foundations and practical aspects of developing a software and hardware complex for automated vehicle control. The analysis focuses on neural network approaches and genetic algorithms used to ensure effective modeling and decision-making in real-time.

A comprehensive approach to combining artificial intelligence methods for vehicle control in a dynamic environment is proposed. This includes object recognition, decision-making, and adaptive control. The research results demonstrate that the developed complex significantly improves the system's accuracy and response speed, ensuring safe and efficient vehicle operation.

Keywords: software and hardware complex, automated control, neural networks, genetic algorithms, artificial intelligence.

ЗМІСТ

ВСТУП

В епоху розвитку цифрового суспільства особливої важливості набувають системи автоматичного керування складними процесами та об’єктами. Одним із таких можливих напрямків їх застосування є автомобільна промисловість. На сьогодні опрацьовано значний досвід у галузі безпілотних транспортних засобів.

Проблеми розроблення та вдосконалення автономних систем керування привернули увагу провідних науковців у галузі штучного інтелекту та нейромережевого моделювання, зокрема І. Гудфеллоу, Я. Лекуна, А. Карпати, Д. Сільвера, а також фахівців у сфері еволюційних алгоритмів, таких як К. Голдберг, Д. Фогель і М. Мітчелл [1–7]. У їхніх роботах обґрунтовано переваги нейронних мереж для вирішення завдань розпізнавання образів та важливість генетичних алгоритмів для оптимізації навчання.

Варто зауважити, що нейронні мережі глибокого навчання є ефективними для розпізнавання об'єктів у реальному часі, що важливо для безпілотних транспортних засобів, авіації, робототехніки та інших високотехнологічних галузей. Потреба в ефективних методах навчання, таких як генетичні алгоритми, залишається актуальною, оскільки вони дозволяють значно скоротити час навчання нейронних мереж, що є критично важливим для систем з обмеженими ресурсами або в реальному часі.

Таким чином, розроблення методів, що інтегрують генетичні алгоритми в процес навчання нейронних мереж, є важливим для оптимізації роботи автономних систем. Це відкриває нові можливості для підвищення ефективності й надійності не лише в транспортних системах, а й у таких галузях, як авіація, управління дронами, робототехніка та автоматизація виробництва, що зумовлює **актуальність** досліджень, реалізованих у даній магістерській роботі.

Метою роботи є покращення системи автоматизованого керування автомобілем за допомогою багатошарового перцептрону з прискореним навчанням на

основі генетичних алгоритмів для забезпечення більш ефективної роботи нейронних мереж.

Для досягнення вищеокресленої мети було поставлено та розв`язано такі задачі:

1. Аналіз сучасних тенденцій та технологій штучного інтелекту як засобу створення продуктивної системи автоматизованого керування автомобілем.
2. Застосування автоматизованої (засобами бібліотеки YOLO V6) мережі глибокого навчання для вивчення дорожньої ситуації з метою отримання вхідних параметрів для багатошарового перцептрону.
3. Побудова багатошарового перцептрону для реалізації системи автоматичного управління автомобілем засобами нейронних мереж.
4. Удосконалення процесу навчання багатошарового перцептрону на основі генетичних алгоритмів.
5. Здійснення програмної реалізації запропонованої системи автопілотажу.
6. Тестування розробленого програмно-апаратного комплексу.
7. Доведення економічної доцільності створеного програмно апаратного комплексу.
8. Висновки та рекомендації

Об'єктом дослідження є процес створення безпілотних систем засобами штучного інтелекту.

Предметом дослідження слугує розроблення підходу до створення програмно апаратного комплексу для автоматизованого керування автомобілем засобами нейромережевого моделювання та генетичних алгоритмів.

Наукова новизна роботи полягає в удосконаленні системи безпілотного керування транспортним засобом шляхом застосування дуального підходу, який враховує оптимізацію навчання багатошарового перцептрону засобами генетичного алгоритму, що, на відміну від існуючих підходів, дозволяє значно прискорити такий

процес та підвищити ефективність роботи відповідного програмно апаратного комплексу, який реалізує таку нейронну мережу.

Практична цінність роботи зумовлюється можливістю застосування програмно апаратного комплексу для продуктивного вирішення задач автопілотажу в режимі реального часу з підвищеною точністю.

Апробація. Результати роботи апробовано на науковій конференції (м. Вінниця, 2024) [8].

Публікації. За результатами дослідження було опубліковано 2 тез та статтю у фаховому журналі “Інформаційні технології та комп'ютерна інженерія” [9].

1 АНАЛІЗ СУЧАСНИХ ТЕНДЕНЦІЙ ТА ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ЯК ЗАСОБУ СТВОРЕННЯ ПРОДУКТИВНОЇ СИСТЕМИ АВТОМАТИЗОВАНОГО КЕРУВАННЯ АВТОМОБІЛЕМ

1.1 Аналіз недоліків та переваг існуючих систем автопілотажу

Автономне керування є важливою складовою майбутнього транспорту, здатною значно змінити автомобільні системи та підвищити безпеку на дорогах. Сучасні системи автопілотів пропонують низку переваг, серед яких покращена безпека, зменшення стресу та втоми водіїв, а також потенціал для підвищення ефективності дорожнього руху. Водночас ці технології мають і свої недоліки, які вимагають вирішення.

Однією з основних переваг автономних транспортних систем є підвищена безпека. Автопілоти здатні оперативно реагувати на зміни дорожніх умов, розпізнавати перешкоди та дорожні знаки, що зменшує ймовірність аварій. Наприклад, система може самостійно утримувати автомобіль у смузі, регулювати швидкість та виконувати об’їзд перешкод, що особливо важливо на довгих маршрутах або в умовах інтенсивного руху, коли водій може втратити концентрацію.

Крім того, автономні системи дозволяють знизити рівень стресу і втоми водіїв, оскільки вони беруть на себе рутинні функції, такі як підтримка постійної швидкості або контроль за рухом в пробках. В таких умовах втома водія – одна з головних причин аварій, і автономне керування може зменшити цей ризик. Це, зокрема, важливо для довгих поїздок, де можливість перерви на відпочинок обмежена.

Однак, є й суттєві виклики, що виникають при впровадженні технології автономного керування. Одним із таких є питання безпеки в складних погодних умовах, таких як сильний дощ, сніг або туман. Сенсори автономних систем можуть втратити свою ефективність в таких умовах, що ставить під загрозу безпеку руху. Це означає, що, хоча системи можуть бути ефективними в звичних умовах, їхня робота в екстремальних ситуаціях все ще потребує вдосконалення [10].

Порівняльний аналіз систем автопілотів дозволяє оцінити різні підходи до реалізації автономного керування в сучасних транспортних засобах. Основними гравцями на цьому ринку є компанії, які займаються розробкою безпілотних автомобілів, такі як Waymo, Cruise, Tesla та інші. Кожна з цих компаній використовує різні технології і підходи до забезпечення автономності транспорту, що визначає ефективність і потенціал цих систем для впровадження в реальний світ.

Waymo – одна з найбільш розвинутих компаній у сфері безпілотних технологій, що належить Google. Їхня система автопілота використовує велику кількість датчиків, камер та лазерних сканерів (LIDAR), що дозволяє створювати високоточні 3D карти навколишнього середовища. Це забезпечує надзвичайно точне розпізнавання об'єктів і перешкод, що є важливим фактором для безпеки руху. Waymo активно тестує свої системи в міських умовах, де складність дорожнього руху значно вища, а також працює над покращенням роботи своїх технологій в умовах поганої видимості, зокрема, вночі. Однією з головних переваг системи Waymo є її здатність працювати в складних умовах та високий рівень безпеки, який був підтверджений численними тестами на дорогах [11, 12].

Cruise, що є підрозділом General Motors, також активно розвиває систему автопілота, але робить акцент на інтеграції безпілотних технологій у вже існуючі транспортні засоби. На відміну від Waymo, Cruise орієнтована на швидке впровадження технологій в реальні умови міського середовища. Компанія має на меті забезпечити автономне керування в межах міста з максимальною ефективністю і безпекою, але при цьому в їхній системі використовується менш кількість датчиків та камер, ніж у Waymo [13]. Це дозволяє знижувати вартість системи та підвищувати економічну доступність технології. Однак менше число сенсорів може знижувати точність системи, зокрема, у складних погодних умовах або в умовах низької видимості.

Tesla – це одна з найбільш відомих компаній у сфері безпілотного транспорту, яка використовує різні технології, такі як камери, радар і ультразвукові сенсори, для

забезпечення автономності своїх автомобілі. Однак на відміну від Waymo і Cruise, Tesla зробила акцент на використання лише камер і програмного забезпечення для обробки зображень, що є суттєвою різницею в порівнянні з іншими системами, які використовують комбінацію камер, LIDAR і радарів [14]. Це дозволяє знижувати вартість і спрощує інтеграцію технології в існуючі транспортні засоби, однак система Tesla не досягла такого рівня точності і надійності, як у Waymo, зокрема у складних умовах. Tesla активно працює над вдосконаленням своїх алгоритмів і програмного забезпечення, але на даний момент її система автопілота вимагає постійного нагляду водія, оскільки не досягає повної автономії [15].

Системи автопілотів різних компаній мають свої переваги та недоліки, які в першу чергу визначаються використаними технологіями та підходами до розробки. Якщо Waymo пропонує надзвичайно точну та надійну систему, яка підходить для використання в складних умовах міського середовища, то Cruise орієнтована на швидке впровадження автономних таксі в реальний світ, з акцентом на економічну вигідність і доступність технології. Tesla, в свою чергу, надає можливість автономного керування для звичайних автомобілів, але з обмеженнями щодо повної автономії, що вимагає нагляду водія.

Загалом, автономне керування є потужним інструментом для розвитку інтелектуальних транспортних систем. Однак для того, щоб ці технології стали дійсно ефективними, необхідно врахувати як технічні, так і соціальні виклики, і поступово адаптувати їх до реальних умов.

Продуктивність, точність розпізнавання та час реакції є ключовими аспектами, що визначають ефективність автономних систем керування транспортними засобами. Кожен з цих елементів грає важливу роль у забезпеченні безпеки, ефективності і зручності використання таких систем у реальному житті.

Точність розпізнавання є критично важливою для забезпечення безпеки в автономних транспортних системах. Технології розпізнавання мають важливу роль у визначенні дорожніх знаків, перешкод, пішоходів, інших транспортних засобів і зміни

дорожніх умов. Системи, які використовують LIDAR та камери, забезпечують високу точність завдяки здатності створювати тривимірні моделі навколишнього середовища, що дозволяє краще ідентифікувати об'єкти та їхні відносні рухи. Однак важливо врахувати, що точність може змінюватися залежно від погодних умов, таких як туман або дощ, що може вплинути на ефективність сенсорів, особливо в системах, які покладаються тільки на камери.

Час реакції системи автопілота — це час, необхідний для обробки отриманих даних і виконання відповідних дій, таких як гальмування, змінення траєкторії руху або ухилення від перешкоди. Чим коротший час реакції, тим швидше система може реагувати на зміни ситуації на дорозі. У тестах, проведених з різними системами, таких як Waymo, Cruise і Tesla, час реакції варіюється в залежності від складності умов і типу оброблюваної інформації. Наприклад, в міських умовах з високим трафіком і різноманіттям перешкод час реакції має критичне значення для уникнення аварій. Водночас, швидкість реакції також залежить від того, чи має система можливість виконувати багато операцій одночасно — обробка сенсорних даних, планування траєкторії і виконання команд керування автомобілем.

Продуктивність систем автопілота визначається здатністю системи обробляти великі обсяги даних у реальному часі. Вона залежить від кількості і типу сенсорів, які використовуються, а також від потужності обчислювальних ресурсів. Сучасні системи, такі як Waymo та Tesla, використовують комбінацію камери, радарів і LIDAR-сенсорів для збору інформації про навколишнє середовище. Чим більше сенсорів і чим точніші дані, тим вищою є загальна продуктивність системи. Наприклад, система Waymo використовує велику кількість датчиків, що дозволяє створювати високоточні карти навколишнього середовища, що суттєво підвищує загальну ефективність і можливості адаптації до складних умов [16].

Системи автопілотажу, що використовуються в автономних транспортних засобах, базуються на передових технологіях для розпізнавання об'єктів та прийняття рішень. Система автопілота Tesla є однією з найбільш передових технологій

автономного водіння, яка поєднує у собі численні датчики, потужні алгоритми штучного інтелекту та складні математичні моделі для забезпечення безпечного і ефективного керування автомобілем. Технологічна інфраструктура цієї системи включає різноманітні сенсори, які працюють синхронно, обробляючи величезний потік даних в реальному часі, що дозволяє автомобілю здійснювати складні маневри з високою точністю та в умовах змінного навколишнього середовища.

Автомобілі Tesla оснащені вісьмома камерами, що забезпечують панорамний 360-градусний огляд навколо транспортного засобу [17]. Камери займаються фіксацією різних елементів навколишнього середовища, таких як дорожні знаки, дорожня розмітка, інші транспортні засоби та пішоходи. Зображення, отримані від камер, піддаються глибокій обробці з використанням алгоритмів комп’ютерного зору для точної інтерпретації ситуації на дорозі.

Встановлений на передній частині автомобіля радар здатен виявляти об'єкти попереду на великій відстані, навіть у складних погодних умовах, таких як дощ чи туман. Радари забезпечують додатковий рівень надійності при виявленні перешкод, а також визначають швидкість і напрямок руху інших об'єктів, що є важливим для прийняття своєчасних рішень [17].

Tesla використовує дванадцять ультразвукових датчиків, розташованих по периметру автомобіля. Ці сенсори ефективно працюють на невеликій відстані, дозволяючи виявляти об’єкти, що знаходяться поблизу автомобіля, що надзвичайно важливо для точного паркування та маневрування у вузьких просторах [17].

Серцем системи автопілота є високопродуктивний комп'ютер, який обробляє дані з усіх сенсорів і камер в реальному часі. Алгоритми машинного навчання та глибокого навчання використовуються для створення 3D-моделей навколишнього середовища, що дозволяє автомобілю приймати рішення, які відповідають вимогам безпеки та ефективності [18].

Процес роботи системи автопілота Tesla складається з кількох етапів, починаючи з збору даних і закінчуючи прийняттям рішень щодо керування

автомобілем. Першим етапом є збір даних з камер, радарів та ультразвукових датчиків, що дозволяє створити детальну картину навколишнього середовища. Обробка цих даних відбувається на обчислювальному модулі, який використовує алгоритми глибокого навчання для розпізнавання об'єктів, визначення їх швидкості, напрямку та інших характеристик. Потім, на основі обробленої інформації, автопілот приймає рішення щодо маневрів, таких як прискорення, гальмування, зміна смуги руху, а також обходження перешкод [18].

Для виконання таких завдань важливим є процесор штучного інтелекту, який обробляє отримані зображення, що фіксуються камерами транспортного засобу. Цей процесор використовує ці зображення як вхід для навченої моделі машинного навчання, що прогнозує траєкторії руху та інші дії для автоматичного керування транспортним засобом. У цьому процесі створюється тривимірне представлення об'єкта (наприклад, лінії смуги руху), яке корелюється з реальними даними — так званою наземною правдою. Це представлення використовує окремі кадри зображень, для яких зібрані тренувальні дані. Використання наземної правди на всьому відеоряді допомагає моделі точніше передбачати траєкторії руху та покращує її навчання, забезпечуючи високу точність і надійність системи автопілота [18].

Процес навчання моделі машинного навчання на основі зібраних тренувальних даних дозволяє системі автопілота приймати правильні рішення в реальному часі, оптимізуючи керування автомобілем в різноманітних дорожніх умовах. Це дає змогу автомобілю виконувати складні маневри, не вимагаючи втручання водія, і забезпечує безпеку та ефективність автономного водіння.

Окрім того, Tesla активно оновлює своє програмне забезпечення, що дозволяє постійно покращувати функціональність системи автопілота. Оновлення можуть включати вдосконалення алгоритмів розпізнавання, поліпшення роботи сенсорів або зміну параметрів керування для підвищення безпеки та комфорту водіїв.

Tesla володіє низкою патентів, які описують технології, що використовуються в системі автопілота. Одним з таких патентів є система автономного водіння, яка

комбінує камери, радар і ультразвукові датчики для забезпечення повного огляду навколишнього середовища [18].

Система автопілота Tesla є прикладом комплексної інтеграції різноманітних технологій в єдину екосистему для забезпечення безпеки та ефективності автономного водіння. Її розвиток продовжує сприяти еволюції автономних транспортних засобів, адже кожне оновлення програмного забезпечення або вдосконалення алгоритмів надає нові можливості для покращення її функціональності та здатності до адаптації в різних дорожніх умовах.

Waymo є однією з найвідоміших компаній у сфері автономного водіння, яка активно розвиває технології для безпілотних автомобілів. Їх система автопілота поєднує кілька складних компонентів для забезпечення безпечного та ефективного руху на дорозі.

Сенсори створюють детальну тривимірну картину оточення, що дає змогу точно оцінювати відстань до різноманітних об'єктів, включаючи інші автомобілі, пішоходів та перешкоди. Завдяки високій точності вимірювань, лідари здатні забезпечити стабільну роботу навіть в умовах обмеженої видимості або складних погодних умов.

Камери високої роздільної здатності виконують важливу функцію розпізнавання об'єктів, та інших транспортних засобів. Зображення, отримане з камер, служить основою для прийняття рішень щодо маневрів, таких як зміна смуги руху чи уповільнення.

Радари, в свою чергу, відіграють важливу роль у вимірюванні відстаней до об'єктів навіть за поганих погодних умов, таких як дощ, сніг або туман. Завдяки здатності працювати за таких умов, радари забезпечують автономним системам стабільність і надійність, зменшуючи ймовірність виникнення небезпечних ситуацій через погану видимість або інші атмосферні фактори.

Ультразвукові датчики використовуються для виявлення об'єктів, що знаходяться на близькій відстані. Ці датчики дозволяють отримувати точну інформацію про об'єкти в безпосередній близькості до автомобіля, що гарантує

високий рівень точності в процесі маневрування в обмежених просторах, таких як паркувальні місця чи вузькі вулиці [19].

Всі ці сенсори працюють у зв'язці з потужним комп'ютерним модулем, який обробляє величезний обсяг даних в реальному часі. Алгоритми машинного навчання, що застосовуються на цьому етапі, дозволяють автомобілю правильно реагувати на різноманітні ситуації на дорозі, розпізнаючи об'єкти, визначаючи їх швидкість та траєкторію руху.

Процес роботи автопілота Waymo складається з кількох етапів. Спочатку він збирає дані з усіх сенсорів, створюючи детальну картину навколишнього середовища. Далі комп'ютерний модуль аналізує ці дані і приймає рішення щодо подальших дій. Це може бути прискорення, гальмування, зміна смуги руху або навіть об'їзд перешкод. Важливою частиною цього процесу є постійне оновлення програмного забезпечення, що дозволяє покращувати функціональність і безпеку автопілота. Кожне оновлення базується на аналізі реальних поїздок, що дозволяє значно підвищити ефективність системи [19].

Однією з головних переваг автопілота Waymo є безпека. Компанія розробляє свої технології з акцентом на забезпечення максимального рівня безпеки як для пасажирів, так і для інших учасників дорожнього руху. Крім того, завдяки багаторічному досвіду та мільйонам миль реальних поїздок, Waymo досягла високого рівня надійності. Їх технологія демонструє здатність працювати в найрізноманітніших умовах, включаючи нічний час та складні погодні умови.

Waymo також має великий досвід в симуляціях, що дозволяє значно покращити систему ще до того, як автомобілі виїдуть на реальні дороги. Це дозволяє компанії постійно вдосконалювати свої технології, забезпечуючи ще більшу точність та надійність роботи системи автопілота.

Таким чином, Waymo продовжує бути лідером у розробці автономних автомобілів, постійно вдосконалюючи свої технології і забезпечуючи новий рівень безпеки та ефективності на дорогах.

Усі ці системи використовують різноманітні архітектури нейронних мереж і їх модифікації для вирішення специфічних завдань, таких як оброблення сенсорних даних, розпізнавання об'єктів, прогнозування руху, адаптивне керування та взаємодія з навколишнім середовищем. Отже, в наступному підрозділі розглянемо найпоширеніші архітектури в системах автономного водіння.

1.2 Вивчення найпоширеніших архітектур нейронних мереж, що використовуються у системах автономного водіння

Однією з найпоширеніших архітектур у системах автономного водіння є конволюційні нейронні мережі (CNN), які спеціалізуються на обробленні зображень та відео. Схематичне зображення архітектури зображено на рис. 1.1. Ці мережі здатні виявляти й класифікувати різні об'єкти на зображеннях, що є необхідним для виявлення пішоходів, інших транспортних засобів, дорожніх знаків і розмітки. Вони використовуються для збирання інформації з камер, що є одним із головних сенсорів у багатьох системах автопілота [20].

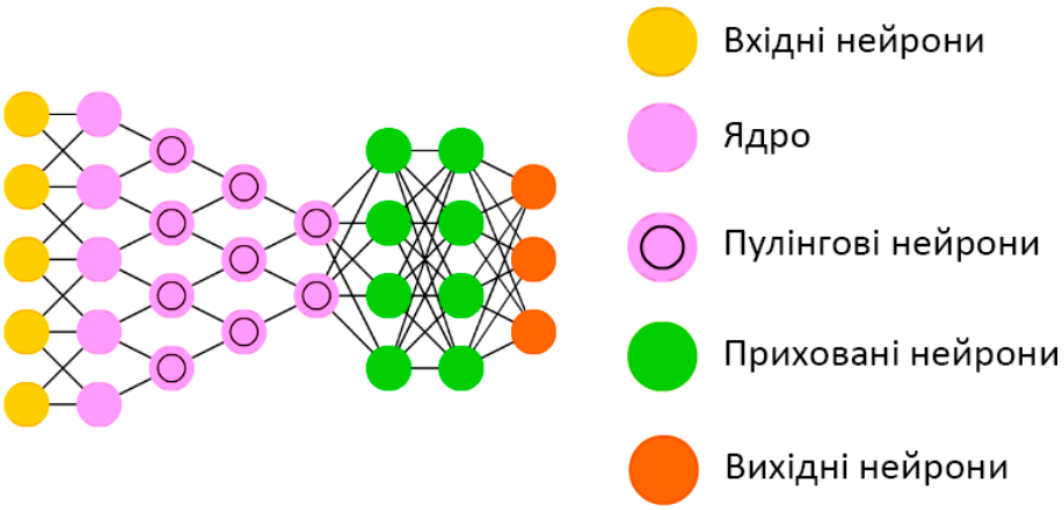


Рисунок 1.1 — Схематичне зображення архітектури Convolutional Neural Networks

Один із прикладів використання CNN у системах автопілота — це HydraNet в Tesla. Це багатозадачна архітектура, яка одночасно вирішує кілька завдань, таких як розпізнавання об'єктів, визначення їхнього типу (пішоходи, машини, знаки тощо), а

також прогнозування їх руху. HydraNet здатна обробляти дані в реальному часі, що критично важливо для прийняття рішень під час руху автомобіля [21].

Рекурентні нейронні мережі (RNN) застосовуються для обробки послідовних даних, таких як відео або часові ряди. Схематичне зображення архітектури зображено на рисунку 1.2. Вони особливо корисні для задач, де важливо враховувати тимчасову залежність між подіями. Наприклад, для передбачення траєкторії руху інших учасників дорожнього руху, таких як пішоходи, велосипедисти або інші автомобілі, необхідно враховувати їхнє попереднє місцезнаходження та швидкість.

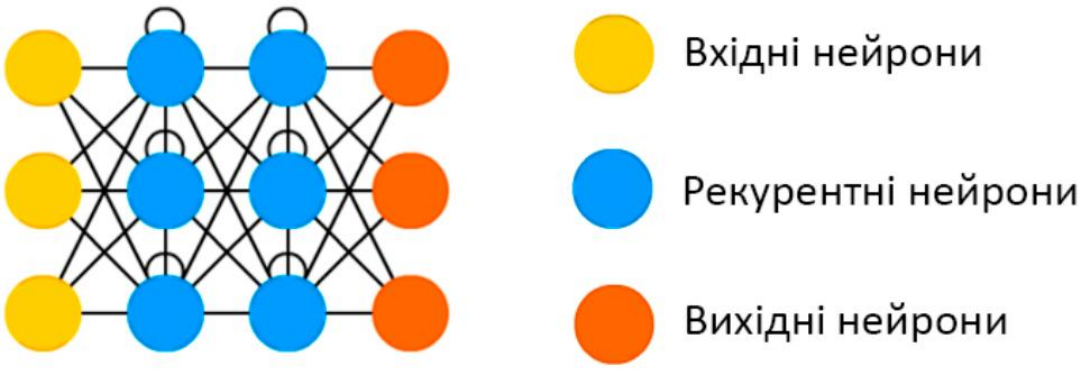


Рисунок 1.2 — Схематичне зображення архітектури Recurrent Neural Networks

У системах автономного водіння активно використовуються LSTM (Long Short-Term Memory) мережі, які є спеціалізованим типом RNN, здатним зберігати інформацію протягом тривалого часу. Завдяки цьому, вони можуть прогнозувати не тільки поточний стан, але й майбутнє положення об'єктів на дорозі, що дає змогу автопілоту передбачати дії інших учасників руху і планувати маневри [20]. Схематичне зображення архітектури зображено на рис. 1.3.

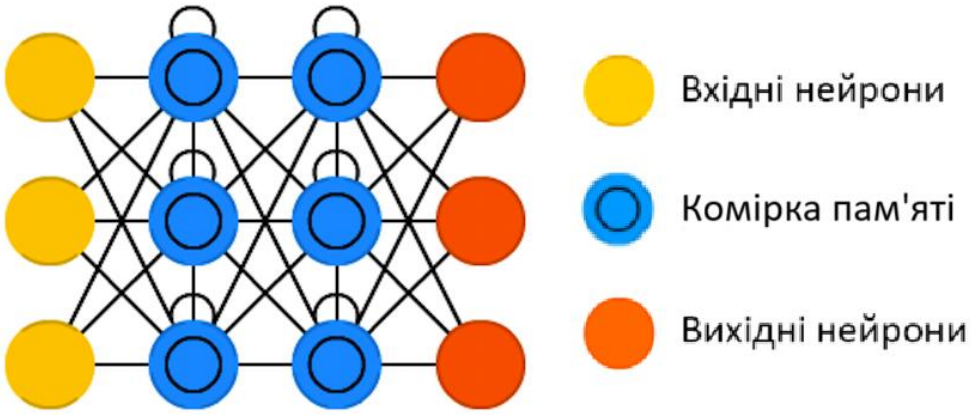


Рисунок 1.3 — Схематичне зображення архітектури Long Short-Term Memory

Глибоке підкріплювальне навчання (Deep Reinforcement Learning, DRL) є потужним методом, який використовує агентів для вивчення оптимальних стратегій поведінки через взаємодію з середовищем. Цей підхід часто застосовується в автономних автомобілях для навчання системи на основі зворотного зв'язку з навколишнім середовищем. У таких системах агент отримує винагороду або покарання залежно від того, чи є його поведінка безпечною та ефективною [21].

Наприклад, система Cruise використовує DRL для адаптивного круїз-контролю, який дозволяє автомобілю автоматично регулювати швидкість і дистанцію до інших транспортних засобів, забезпечуючи безпеку та комфорт під час руху. DRL також допомагає в адаптації до різних дорожніх умов та ситуацій, таких як зміна дорожньої обстановки або аварійні ситуації [15].

End-to-End архітектури являють собою нейронні мережі, які безпосередньо перетворюють сенсорні дані в команди керування автомобілем без проміжних етапів. Це означає, що замість того, щоб виконувати кілька окремих завдань, система автономного водіння використовує одну мережу для обробки всіх вхідних даних і прийняття рішень про рух автомобіля [22].

Waymo використовує подібну архітектуру, яка складається з серії нейронних мереж, що працюють разом для забезпечення безпечного та ефективного автономного водіння. Ці мережі навчено на величезних обсягах даних, зібраних під час реальних поїздок, що дозволяє системі передбачати різні дорожні ситуації та приймати рішення

про маневри в реальному часі. End-to-End підхід дозволяє значно спростити модель, зменшити її складність і підвищити ефективність, оскільки всі етапи обробки даних та прийняття рішень інтегровані в одну систему.

Гібридні архітектури поєднують різні типи нейронних мереж для досягнення кращої продуктивності. Це може бути, наприклад, комбінація CNN для обробки зображень і RNN для обробки послідовних даних, що дозволяє враховувати часову залежність при прогнозуванні траєкторій руху. Такий підхід дає змогу комбінувати переваги кожної з архітектур, забезпечуючи високу точність та швидкість обробки даних [21].

У таких системах, як Waymo, гібридні архітектури дозволяють створювати точні карти навколишнього середовища, прогнозувати поведінку інших учасників дорожнього руху та адаптуватися до змінюваних умов. Наприклад, зображення з камер можна обробляти за допомогою CNN, щоб виявити об'єкти на дорозі, а RNN використовуються для прогнозування, як ці об'єкти будуть рухатися [13].

Огляд бібліотек для штучного зору. Штучний зір активно розвивається завдяки впровадженню інноваційних алгоритмів та потужних бібліотек, які забезпечують високу ефективність розпізнавання й аналізу візуальних даних. Серед широкого спектра доступних рішень особливої уваги заслуговують бібліотеки для реалізації об'єктного виявлення, що є важливим елементом таких сфер, як автономні транспортні засоби, відеоспостереження та інтелектуальні системи моніторингу.

YOLOv6 (You Only Look Once version 6) — це сучасний одностадійний метод об'єктного виявлення, розроблений компанією Meituan [24]. Ця версія поєднує високу швидкість роботи з точністю, яка раніше була характерна для складніших моделей, і підходить для задач реального часу, таких як відеоспостереження, автономні транспортні засоби та аналіз поведінки. Модель може працювати на різних пристроях, включаючи мобільні.

Архітектура YOLOv6 включає:

— Основа New CSP-Darknet53 для вилучення характеристик із зображення

— Шийка SPPF та New CSP-PAN для аналізу об'єктів різних розмірів

— Голова модифікація YOLOv3 для точного визначення координат і класів об'єктів [15].

Ключові особливості:

— Bidirectional Concatenation (BiC) покращує точність локалізації

— Anchor-Aided Training (AAT) універсальний підхід до виявлення об'єктів різних розмірів

— Self-Distillation дозволяє компактним версіям досягати високої продуктивності з меншими вимогами до обладнання [24].

Завдяки цим рішенням YOLOv6 є ефективним вибором для широкого спектра застосувань у реальному часі.

OpenCV, або Open Source Computer Vision Library, є потужною відкритою бібліотекою, розробленою для задач комп'ютерного зору, машинного навчання та обробки зображень. Вона стала важливим інструментом для розробників і дослідників завдяки своїй масштабності, універсальності та здатності працювати в реальному часі. OpenCV містить понад 2500 алгоритмів, що охоплюють широкий спектр завдань, включаючи розпізнавання об'єктів, аналіз відео, калібрування камер і навіть використання нейронних мереж [25]. Спочатку створена компанією Intel, бібліотека продовжує активно підтримуватися та розвиватися. На її основі побудовано безліч комерційних і дослідницьких проектів.

Історія OpenCV починається з 1999 року, коли бібліотека була представлена як дослідницький проект компанії Intel [25]. Метою розробників було створення інструменту, який би прискорив впровадження складних обчислювальних алгоритмів, пов'язаних із комп'ютерним зором. Перша альфа-версія бібліотеки була презентована у 2000 році на конференції IEEE з комп'ютерного зору та розпізнавання шаблонів (Computer Vision and Pattern Recognition) [25]. У 2006 році вийшла перша стабільна версія OpenCV 1.0, що стала основою для багатьох проектів. Згодом бібліотека

еволюціонувала, додаючи нові функції та підтримуючи сучасні технології. Оновлення відбуваються регулярно, з випусками нових версій приблизно кожні пів року.

Однією з важливих переваг OpenCV є її багатоплатформність. Вона підтримує численні мови програмування, включаючи C, C++, Python, Java, і навіть асемблер. Завдяки цьому бібліотека стала доступною для широкого кола розробників, незалежно від їхніх технологічних вподобань [26]. Крім того, OpenCV працює на різноманітних операційних системах, таких як Windows, Linux, macOS, Android, iOS та інші, що забезпечує універсальність у розробці. Сучасні версії бібліотеки також включають підтримку GPU-прискорення, що робить її придатною для задач, які вимагають обробки великих обсягів даних у реальному часі.

Структура OpenCV складається з кількох основних модулів, кожен з яких виконує специфічні функції. Наприклад, модуль Core містить базові елементи бібліотеки, Imgproc відповідає за функції оброблення зображень, а Highgui забезпечує можливості для захоплення відео та відображення зображень. Інші важливі модулі включають Calib3d, який дозволяє виконувати калібрування камери та тривимірну реконструкцію, а також Objdetect, який використовується для розпізнавання об'єктів. Особливо слід відзначити модуль Dnn (Deep Neural Networks), який дозволяє інтегрувати та використовувати переднавчені нейронні мережі. Для графічного моделювання додано модуль Gapi, що розширює можливості комп'ютерного зору [17].

Функціональність OpenCV надзвичайно багата. Вона дозволяє виконувати широкий спектр завдань, починаючи від базових операцій, таких як фільтрація, згладжування або обрізання зображень, і закінчуючи складними алгоритмами розпізнавання об’єктів і аналізу відео. Наприклад, OpenCV широко використовується для розпізнавання облич, що є основою багатьох систем безпеки. Крім того, бібліотека може аналізувати відеопотоки в реальному часі, що відкриває великі можливості для розробки додатків, пов’язаних із моніторингом і відеоспостереженням. Завдяки можливостям нейронних мереж OpenCV використовується для тренування моделей і розпізнавання шаблонів, що є критично важливим для багатьох сучасних AI-рішень.

Сфери використання OpenCV охоплюють різноманітні галузі. Бібліотека є основним інструментом у проектах, пов’язаних із розпізнаванням облич, що знаходить застосування в системах контролю доступу, мобільних додатках та навіть у маркетингу. Обробка відео є ще одним важливим напрямом, адже OpenCV дозволяє аналізувати рух, виявляти аномалії та створювати спеціальні ефекти. У галузі робототехніки OpenCV допомагає розробникам створювати системи навігації та обробки даних, отриманих від камер.

TensorFlow — це популярна відкрита бібліотека програмного забезпечення, яка використовується для машинного навчання (ML) та штучного інтелекту (AI). Її розробником є компанія Google, яка створила цей інструмент для вирішення широкого спектру завдань, пов’язаних із розробкою, навчанням і використанням моделей нейронних мереж [26]. TensorFlow дозволяє будувати складні моделі ML, використовуючи різноманітні середовища, включаючи центральні процесори (CPU), графічні процесори (GPU), а також спеціалізовані апаратні засоби, такі як Tensor Processing Unit (TPU) [27].

TensorFlow було офіційно представлено у листопаді 2015 року як наступник попередньої бібліотеки DistBelief, яка використовувалася в Google для внутрішніх проєктів. DistBelief мала обмеження, які ускладнювали її масштабування та застосування в зовнішніх проєктах. TensorFlow, на відміну від свого попередника, була створена з урахуванням універсальності та можливості використання не тільки в Google, але й іншими компаніями та розробниками [28].

TensorFlow має модульну архітектуру, яка складається з кількох ключових компонентів:

— TensorFlow Core Основна бібліотека, яка надає базові функції для створення та навчання моделей нейронних мереж. Вона забезпечує роботу з математичними операціями, такими як обчислення градієнтів, оптимізація та обробка тензорів [27]

— Keras це високорівневий API, інтегрований у TensorFlow, що спрощує побудову, налаштування та навчання моделей. Keras надає зручний і інтуїтивно зрозумілий інтерфейс для початківців і досвідчених користувачів [27]

— TensorFlow Extended (TFX) це комплекс інструментів, які покривають весь життєвий цикл машинного навчання, включаючи підготовку даних, навчання, розгортання моделей та моніторинг їх роботи [27]

— TensorFlow Lite це полегшена версія TensorFlow, призначена для роботи на мобільних пристроях і вбудованих системах, де обмежено ресурси [27]

— TensorFlow.js це версія TensorFlow, яка працює безпосередньо в браузері за допомогою JavaScript. Це дозволяє запускати моделі ML у веб-додатках без необхідності встановлення серверної інфраструктури [27]

TensorFlow забезпечує широкий набір можливостей, які роблять її універсальним інструментом для машинного навчання:

Бібліотека підтримує різні архітектури нейронних мереж, зокрема згорткові (CNN), рекурентні (RNN) та багатошарові персептрони (MLP). Це дозволяє реалізувати різноманітні підходи до вирішення задач [28].

TensorFlow може використовуватися на різних пристроях — від мобільних телефонів до кластерів серверів. Завдяки підтримці GPU та TPU, бібліотека здатна забезпечувати ефективне навчання навіть дуже великих моделей [27].

Інструменти розроблення: TensorFlow пропонує інструменти, які спрощують розробку та навчання моделей. Наприклад, TensorBoard забезпечує візуалізацію навчального процесу, а TensorFlow Hub дозволяє повторно використовувати попередньо навчені моделі.

1.3 Принципи роботи нейронних мереж, застосовуваних у системах автопілотажу

Штучний інтелект (ШІ) є однією з найважливіших і найдинамічніших сфер сучасної науки та технологій. Він займається розробкою комп’ютерних систем,

здатних до інтелектуальної діяльності, навчання, прийняття рішень та виконання завдань, які раніше вважалися виключно людськими. ШІ знаходить широке застосування в різних галузях, таких як медицина, транспорт, фінанси, робототехніка, ігрова індустрія тощо.

Однією з ключових характеристик ШІ є здатність аналізувати, розуміти та приймати рішення, використовуючи великі масиви даних. Для цього потрібні потужні алгоритми та архітектури, які забезпечують ефективну обробку й використання інформації.

Перцептрон — це одна з найперших моделей штучного нейрона, яку створив Френк Розенблатт у 1958 році. Це досить проста структура, яка заклала основи для розвитку нейронних мереж і працює як лінійний класифікатор. Графічний вигляд перцептрона показано на рисунку 1.4 [29].

Уявімо, що перцептрон — це модель, яка допомагає визначити, до якого класу належить певний сигнал. Він працює так: отримує вхідні дані (наприклад, числа або інші характеристики), а потім множить їх на так звані ваги. Ваги — це коефіцієнти, які визначають важливість кожного вхідного значення.

Далі отриманий результат проходить через спеціальну функцію, яку називають пороговою функцією активації. Ця функція вирішує, чи відповідає сигнал певному класу. Наприклад, якщо результат перевищує заданий поріг, система вважає, що сигнал належить до одного класу, а якщо ні — до іншого.

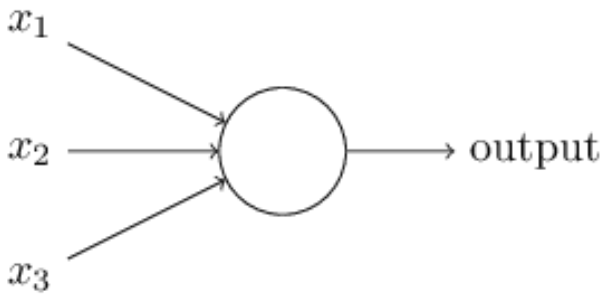


Рисунок 1.4 — Схематичне зображення перцептрона

Френк Розенблатт запропонував просте правило для обчислення вихідного значення в перцептроні. Він увів поняття "значущості", яке в подальшому стало називатися "вагою" для кожного вхідного значення.

Щоб зрозуміти, як працює перцептрон, уявімо, що ми маємо набір вхідних значень x, і кожне з них помножується на відповідну вагу w. Потім результати цих множень додаються разом

Вихідне значення *output* залежить від того, чи перевищує ця сума задане порогове значення *threshold*, яке також є дійсним числом. Результат роботи персептрона визначається за таким правилом яке показано в формулі 1.1.

$$output = \begin{cases} 0 \ \ if \ \sum_{i=1}^{N} x_i w_i \ \leq \ threshold \\ 0 \ \ if \ \sum_{i=1}^{N} x_i w_i \ \geq \ threshold \end{cases} \tag{1.1}$$

Це означає, що якщо зважена сума входів не перевищує поріг, вихід персептрона дорівнює 0. Якщо ж ця сума більша за поріг, вихід дорівнює 1.

І цього правила цілком достатньо для роботи! Змінюючи порогове значення *threshold* та вектор вагів, можна створювати різні моделі прийняття рішень.

Попри простоту, персептрон має обмеження. Він не здатний розв'язувати складні задачі, де класи не можна розділити прямою лінією. Наприклад, класична задача XOR (ексклюзивне "або") є нелінійно роздільною, і персептрон із нею не справляється [30].

Однак, попри ці недоліки, персептрон став важливим кроком у розвитку нейронних мереж. Його ідеї лягли в основу складніших моделей, які тепер можуть вирішувати набагато більш комплексні задачі.

FeedForward Neural Network (FFNN), або мережа прямого поширення, є однією з базових архітектур нейронних мереж, яка складається з трьох основних компонентів: вхідного шару, прихованих шарів (одного або більше) та вихідного шару. У цій мережі інформація передається лише в одному напрямку — від входу до виходу,

без зворотного зв'язку. Схематичне зображення архітектури зображено на рис. 1.5.

Рисунок 1.5 — Схема архітектури Feed Forward

Кожен нейрон у мережі виконує кілька ключових операцій: він отримує вхідні сигнали, обчислює їхню зважену суму, додає зсув (bias), а потім застосовує активаційну функцію для визначення вихідного сигналу. Цей процес повторюється на кожному шарі мережі, і в результаті на виході формується фінальний результат [31].

Завдяки такій архітектурі FFNN здатна моделювати складні нелінійні залежності між вхідними та вихідними даними. Це робить її ефективним інструментом для вирішення широкого спектра задач, таких як класифікація, регресія, розпізнавання образів, обробка зображень, комп’ютерний зір і багато інших.

Багатошаровий перцептрон (MLP) є однією зі специфічних реалізацій FFNN. Основною характеристикою MLP є обов’язкова наявність хоча б одного прихованого шару, тоді як FFNN у загальному випадку може включати або не включати приховані шари. Ще одна відмінність полягає у зв’язках між нейронами: MLP зазвичай є повнозв’язаною мережею, де кожен нейрон у шарі пов’язаний з усіма нейронами сусідніх шарів. У FFNN такі зв’язки можуть бути організовані інакше, залежно від потреб конкретної задачі [32].

Таким чином, FFNN є більш універсальним поняттям, яке охоплює як прості, так і складні архітектури. MLP, своєю чергою, є однією з ключових реалізацій FFNN, що відзначається багатошаровою структурою та повною зв’язаністю нейронів. Це

дозволяє ефективно використовувати MLP для задач, які вимагають детального аналізу складних патернів у даних.

Рекурентні нейронні мережі (Recurrent Neural Networks, або RNN) – це особливий тип нейронних мереж, який використовує зворотний зв'язок між нейронами для обробки даних. Схематичне зображення архітектури зображено на рисунку 1.6. Завдяки такій архітектурі мережа може зберігати інформацію про попередні стани та використовувати її під час аналізу наступних вхідних даних [31, 33, 34].

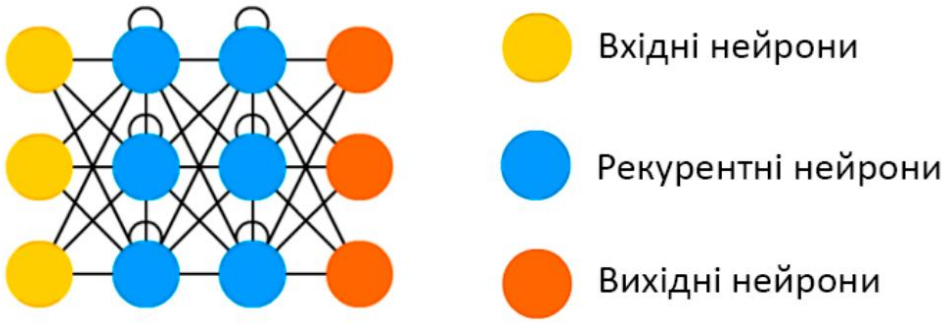


Рисунок 1.6 — Схематичне зображення архітектури Recurrent Neural Networks

Основний принцип роботи RNN полягає в тому, що нейрон отримує поточний вхідний сигнал разом із попереднім станом, обробляє їх і формує вихідний сигнал та новий стан. Цей новий стан передається далі, забезпечуючи можливість враховувати контекст і залежності між елементами послідовності.

Завдяки своїм особливостям RNN є ефективним інструментом для роботи з послідовними даними, такими як текст, числові ряди, аудіосигнали чи музика. Вона дозволяє виявляти зв'язки між елементами послідовностей, що робить її незамінною у задачах обробки природної мови, аналізу часових рядів, генерації тексту, мелодій тощо.

Зауважимо, що робота згорткових нейронних мереж базується на використанні фільтрів, які виконують операцію згортання для виявлення локальних ознак, таких як межі, кути чи текстури. Після згортки застосовується пулінговий шар, який зменшує

розмірність даних, зменшуючи кількість параметрів і підвищуючи стійкість до малих змін у зображенні. CNN також можуть містити повнозв’язані шари для остаточної класифікації чи регресії.

Ця архітектура широко використовується в задачах комп'ютерного зору, таких як класифікація зображень, розпізнавання об’єктів та сегментація. Схематичне зображення CNN наведено на рис. 1.1.

Кожна з цих архітектур як описано було вище, має розширення, які дозволять їм вирішувати більш складні задачі та працювати зі специфічними типами данних.

Розширення архітектури рекурентних нейронних мереж (RNN) включають різні підходи, спрямовані на вирішення специфічних задач та покращення продуктивності нейромереж.

Було розглянуто в магістерській роботі основи роботи RNN, що вимагає подальшого огляду їхніх розширень.

Одним із найпопулярніших розширень є Long Short-Term Memory (LSTM), яке використовує спеціальні комірки пам’яті для роботи з довготривалими залежностями. У структурі LSTM передбачено три типи воріт: забуття, оновлення та виведення. Ці ворота дозволяють мережі вирішувати, яку інформацію зберігати, яку видаляти і яку використовувати для прогнозів. Завдяки такій структурі LSTM здатна ефективно обробляти послідовності даних, що мають довготривалі залежності. Наприклад, вона широко застосовується в обробці тексту, де контекст може простягатися на кілька речень чи навіть абзаців [35].

Інше важливе розширення – це Gated Recurrent Unit (GRU). GRU має спрощену архітектуру порівняно з LSTM, що робить її швидшою в навчанні. Вона використовує лише два типи воріт: оновлення та скидання. Попри свою простоту, GRU забезпечує високу ефективність у задачах обробки послідовностей і часто конкурує з LSTM за точністю [36].

Для задач, де важливо враховувати контекст з обох напрямків послідовності, розроблено Bidirectional RNN (BRNN). У цій архітектурі використовуються дві окремі

RNN, які працюють у протилежних напрямках: одна аналізує дані з початку до кінця, а інша – з кінця до початку. Це дозволяє враховувати як попередній, так і наступний контекст, що є критичним у таких завданнях, як машинний переклад чи розпізнавання мови.

Ще одним цікавим підходом є Hierarchical RNN (HRNN). Вона використовує ієрархічну структуру для моделювання послідовностей різного рівня складності. Наприклад, текст може бути представлений як ієрархія: символи утворюють слова, слова – речення, а речення – абзаци. Завдяки цьому HRNN може ефективно працювати з великими обсягами тексту чи іншими даними, що мають складну структуру.

Розширення архітектури RNN не обмежуються лише послідовностями. Наприклад, у згорткових нейронних мережах (CNN), які розглядались у пункті 1.1, також впроваджено інноваційні підходи. Один із них – Dilated Convolution, який дозволяє розширювати поле зору моделі без збільшення кількості параметрів. Це досягається за рахунок збільшення інтервалів між елементами фільтра. Такий підхід є особливо корисним для роботи з великими зображеннями або завданнями, де потрібно враховувати широкий контекст [35].

Для оброблення зображень різного масштабу часто використовується Spatial Pyramid Pooling (SPP). Цей метод дозволяє моделі враховувати об’єкти різного розміру на одному зображенні. Замість стандартного пулінгу застосовуються пірамідальні шари, що зменшують залежність моделі від фіксованих розмірів вхідних даних. Це робить SPP корисним для задач, пов’язаних із класифікацією та сегментацією зображень.

Ще одним важливим компонентом сучасних CNN є Inception Module. Він дозволяє обробляти ознаки різного масштабу за допомогою паралельних згорткових шарів із фільтрами різних розмірів. Завдяки цьому модуль одночасно виявляє локальні й глобальні ознаки, що підвищує ефективність і точність моделі. Крім того, Inception Module включає 1x1 згортку, яка зменшує кількість параметрів, що допомагає уникати перенавчання.

І, наостанок, варто згадати Residual Connections (Резидуальні зв’язки), які стали революційним рішенням для глибоких нейронних мереж. Вони дозволяють передавати інформацію напряму між шарами, оминаючи проміжні обчислення. Це розв’язує проблему зникнення градієнта, що часто виникає під час навчання дуже глибоких мереж, і полегшує тренування моделей із сотнями шарів.

Таким чином, розширення архітектури RNN та CNN забезпечують гнучкість, ефективність і точність при роботі з різними типами даних і задачами. Вони дозволяють адаптувати нейронні мережі до конкретних умов і значно розширюють їхні можливості.

Марковський ланцюг — це математична модель, що використовується для опису випадкових послідовностей, де кожен наступний стан залежить лише від попереднього з певною ймовірністю переходу [33]. На рис. 1.4 наведено схематичне зображення цього розширення

Рисунок 1.4 — Схематичне зображення Markov Chain (Марковського ланцюга)

У контексті нейронних мереж Марковські ланцюги застосовуються для аналізу й моделювання послідовних даних, таких як текст, фінансові часові ряди, траєкторії руху, генетичні послідовності тощо [33].

Основою Марковського ланцюга є набір станів і ймовірностей переходу між ними. Кожен стан відображає конкретну конфігурацію або положення системи, а

ймовірності переходу визначають, з якою ймовірністю відбуватиметься зміна стану. У кожен момент часу система переходить у новий стан, залежно від поточного стану та відповідної ймовірності [37].

У сфері нейронних мереж Марковські ланцюги використовуються для виконання різноманітних завдань. Наприклад:

— прогнозування майбутніх станів системи або послідовності на основі попередніх даних і ймовірностей переходу;

— генерація нових послідовностей, які зберігають схожі характеристики до вихідних навчальних даних;

— розпізнавання, класифікація й аналіз послідовних даних із урахуванням їхньої структури.

Одним із прикладів інтеграції Марковських ланцюгів у нейронних мережах є використання Recurrent Neural Networks (RNN). Ці мережі мають зворотні зв’язки, що формують цикли, завдяки яким інформація передається з попередніх кроків на наступні. Це дає змогу зберігати контекст і враховувати залежності між станами під час аналізу або генерації даних. Завдяки цьому RNN ефективно моделює послідовності, забезпечуючи глибше розуміння контексту й взаємозв’язків у даних.

1.4 Аналіз недоліків та переваг існуючих методів навчання нейронних мереж для побудови системи автоматизованого керування автомобілем

Розгляд методів навчання штучного інтелекту спрямований на аналіз основних підходів і алгоритмів, які є фундаментом для створення інтелектуальних систем. Кожен із цих методів має свої унікальні переваги та обмеження, що визначають їх застосування залежно від характеру задачі, доступності даних і специфіки використання. Глибоке дослідження цих підходів дає змогу краще зрозуміти механізми навчання штучного інтелекту та вибрати найбільш оптимальний метод для конкретної задачі.

Штучний інтелект використовує різноманітні підходи до навчання для імітації людських когнітивних процесів і розв'язання складних завдань. Навчання з учителем базується на аналізі прикладів із заздалегідь визначеними вхідними та вихідними значеннями, що дозволяє знаходити зв’язки між ними. У навчанні без учителя модель працює з невпорядкованими даними, шукаючи приховані закономірності й структури без наявності очікуваних результатів. Підсилене навчання орієнтоване на взаємодію агента з середовищем, де рішення оцінюються через винагороди або покарання. Генетичні алгоритми, натхненні принципами природного відбору, використовують генетичні оператори для пошуку оптимальних рішень шляхом схрещування і мутації параметрів.

Метод зворотного поширення помилки є ключовим у навчанні багатошарових нейронних мереж. Цей алгоритм дозволяє ефективно налаштовувати ваги моделі, мінімізуючи похибку між прогнозованим і очікуваним результатами. Процес включає обчислення градієнтів функції втрат відносно ваг кожного шару, після чого ці градієнти використовуються для корекції ваг у напрямку зменшення помилки. Завдяки цьому методу нейронні мережі можуть автоматично навчатися складним залежностям у великих наборах даних, що робить його основою багатьох сучасних інтелектуальних систем [33].

Ці методи активно застосовуються в багатьох сферах, таких як комп'ютерний зір, обробка природної мови, автономні транспортні засоби та робототехніка. Комбінування цих підходів і їх адаптація під специфічні завдання дозволяє створювати потужні й ефективні моделі, здатні розв'язувати широкий спектр проблем.

Сучасні методи навчання нейронних мереж включають широкий спектр технік, які спрямовані на підвищення точності, ефективності та швидкості навчання моделей. Ці підходи враховують особливості задачі, обсяг і якість даних, а також апаратні можливості, що робить їх універсальними інструментами для створення нейронних мереж.

Глибоке навчання базується на використанні багатошарових нейронних мереж, що дозволяють моделі автоматично вилучати значущі особливості з даних. Чим більше шарів у мережі, тим більше рівнів абстракції вона здатна розпізнавати. Наприклад, згорткові нейронні мережі (CNN) ідеально підходять для аналізу зображень, бо автоматично розпізнають структури, такі як краї, текстури або форми. Рекурентні нейронні мережі (RNN) ефективні для роботи з послідовними даними, такими як текст або часові ряди, оскільки вони враховують попередній контекст. Трансформери, які використовуються в сучасних мовних моделях, як GPT або BERT, забезпечують високий рівень продуктивності в задачах обробки природної мови.

Переносне навчання дозволяє повторно використовувати моделі, попередньо навчені на великих наборах даних, для вирішення нових, схожих задач. Наприклад, модель ResNet, навчену для розпізнавання загальних об'єктів, можна використовувати для класифікації конкретних видів рослин, лише додатково навчити кілька фінальних шарів. Це значно скорочує час навчання та знижує потребу в великих наборах даних.

Регуляризація – це набір технік, які допомагають уникнути перенавчання моделі, тобто ситуації, коли вона занадто добре "запам’ятовує" навчальні дані, але погано працює з новими. Популярними методами є L1 та L2 регуляризація, які обмежують ваги нейронів, щоб зробити модель більш узагальненою. Dropout тимчасово "відключає" випадкові нейрони під час навчання, що стимулює мережу працювати ефективніше навіть за відсутності певних даних.

Щоб покращити стійкість моделі до варіацій у даних, застосовується аугментація. Це метод створення нових прикладів шляхом внесення змін до наявних. Для зображень це можуть бути повороти, обрізка, зміна кольору чи яскравості. Для тексту використовують синонімічні заміни, додавання шуму або перемішування слів.

Сучасні алгоритми оновлення ваг, як-от Adam чи RMSprop, автоматично налаштовують швидкість навчання для кожного параметра. Це пришвидшує процес і робить його більш стабільним, особливо під час роботи зі складними даними.

2 РОЗРОБЛЕННЯ МЕТОДУ ПОБУДОВИ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ АВТОМАТИЧНОГО УПРАВЛІННЯ АВТОМОБІЛЕМ ЗАСОБАМИ НЕЙРОННИХ МЕРЕЖ ТА ГЕНЕТИЧНИХ АЛГОРИТМІВ

2.1 Вибір найбільш продуктивної бібліотеки штучного зору для вивчення дорожньої обстановки у реальному часі для створення системи автопілотажу

TensorFlow і OpenCV пропонують потужні інструменти для виявлення об'єктів, однак їхні підходи до вирішення задач і сфери застосування суттєво різняться. TensorFlow вирізняється високим рівнем налаштовуваності та здатністю досягати виняткової точності, підтримуючи складні моделі, як-от згорткові мережі чи трансформери. Проте використання таких моделей пов'язане з високими обчислювальними витратами, що обмежує їх застосування у реальних сценаріях з недостатнім апаратним забезпеченням. OpenCV, у свою чергу, спеціалізується на класичних алгоритмах комп’ютерного зору, забезпечуючи високу оптимізацію для задач попередньої обробки зображень і відеоаналізу. Водночас можливості OpenCV у складних задачах виявлення об'єктів є обмеженими, особливо якщо порівнювати з нейронними мережами, які пропонують сучасні моделі.

YOLOv6 стала ключовим вибором для виявлення об'єктів у реальному часі завдяки своїй здатності забезпечувати баланс між швидкістю, точністю та універсальністю. Вона дозволяє досягати високої продуктивності навіть на обладнанні середньої потужності, що робить її доступною для широкого кола розробників. Гнучкість YOLOv6 у налаштуванні для різних сценаріїв і простота інтеграції на різних платформах виділяють її серед аналогів, як-от TensorFlow чи OpenCV. Порівняння цих рішень наведено в табл. 2.1.

Таблиця 2.1 – Порівняння трьох найпопулярніших рішень

Критерій	TensorFlow	OpenCV	YOLOv6
Призначення	Загальне машинне навчання	Обробка зображень та відео	Виявлення об'єктів у реальному часі

Продовження таблиці 2.1

Гнучкість	Висока	Середня	Низька
Швидкість	Середня (залежить від моделі)	Висока для базових алгоритмів	Дуже висока
Точність	Залежить від архітектури	Обмежена класичними методами	Висока для задач виявлення
Легкість у використанні	Складна для початківців	Відносно проста	Простий інтерфейс, але потребує GPU
Масштабованість	Висока	Обмежена	Середня
Підтримка спільноти	Широка	Велика	Менш поширена

Важливо відзначити відмінності між цими інструментами. TensorFlow, незважаючи на його переваги у складних задачах, має високий поріг входу для нових розробників та потребує значних обчислювальних ресурсів. OpenCV є ідеальним вибором для задач, які не потребують глибоких нейронних мереж, проте не може конкурувати з сучасними моделями у високоточних задачах виявлення. У свою чергу, YOLOv6 орієнтована саме на завдання реального часу, де критичними є швидкість та точність.

Попередні версії YOLO, зокрема YOLOv3 і YOLOv4, свого часу були проривними в об'єктному виявленні. Проте з розвитком технологій їхні архітектури стали застарілими, поступаючись новим версіям за точністю та ефективністю обробки складних даних. Навіть YOLOv5, яка є попередницею YOLOv6, хоч і демонструє схожі підходи до оптимізації, не має таких інноваційних рішень, як модулі BiC чи AAT.

YOLOv6, розроблена компанією Meituan, є результатом постійного вдосконалення одностадійних методів виявлення об'єктів. Її архітектура складається з трьох ключових компонентів: Основа, Шийка та Голова.

Основа використовує модифіковану версію Darknet53, яка забезпечує ефективне вилучення базових характеристик зображень із мінімальними обчислювальними витратами.

Шийка YOLOv6 інтегрує SPPF і RepBi-PAN для покращення аналізу об'єктів різного розміру, а голова моделі забезпечує точність і швидкість.

RepBi-PAN (Reparameterized Bi-directional Path Aggregation Network) — це вдосконалена архітектура "шийки" у YOLOv6, яка забезпечує ефективне об'єднання ознак з різних рівнів для покращення точності виявлення об'єктів.

Багатонаправлене об'єднання ознак: RepBi-PAN використовує двонаправлене об'єднання ознак для покращення сигналів локалізації. Це дозволяє моделі точніше визначати місцезнаходження об'єктів, особливо малих.

Його архітектуру зображено на рис. 2.1

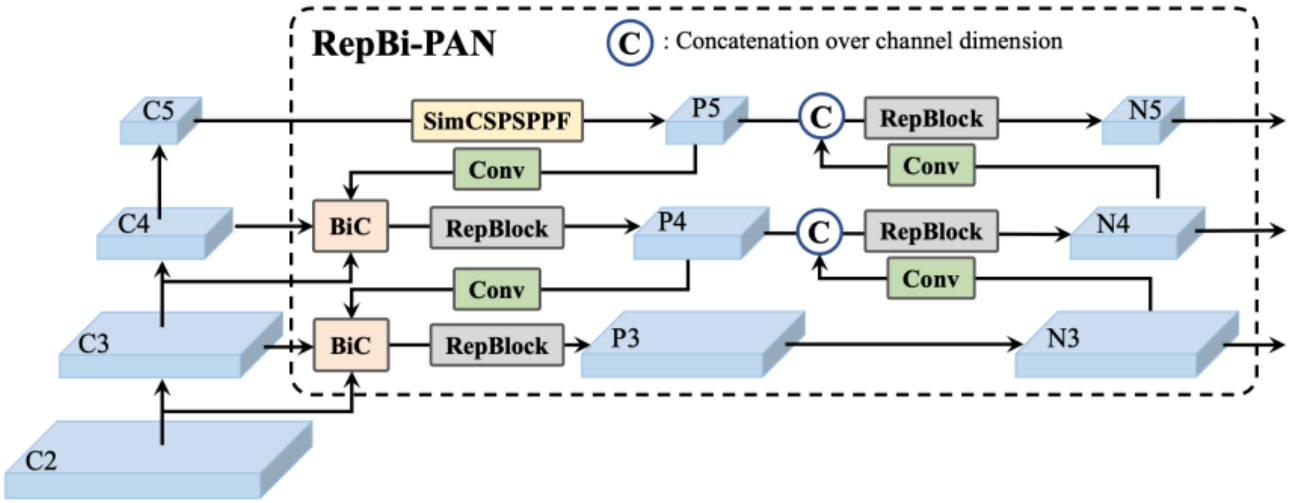


Рисунок 2.1 — Схематичне зображення архітектури RepBi-PAN

Завдяки RepBi-PAN, YOLOv6 досягає високої точності без значного зниження швидкості роботи моделі.

Архітектура RepBi-PAN включає кілька ключових компонентів:

— BiC (Bi-directional Concatenation) модуль: Цей модуль підвищує точність визначення меж об'єктів у складних умовах, об'єднуючи дані з різних рівнів архітектури (рис. 2.2)

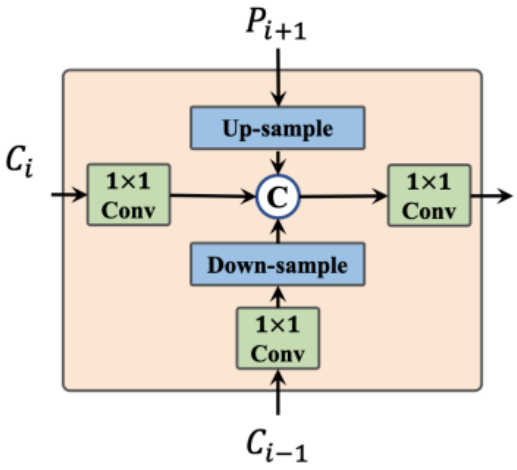


Рисунок 2.2 — Схематичне зображення блоку BiC

— SimCSPSPPF блок: Спрощена версія SPPF блоку, яка підвищує здатність до представлення ознак без значного зниження швидкості (рис. 2.3)

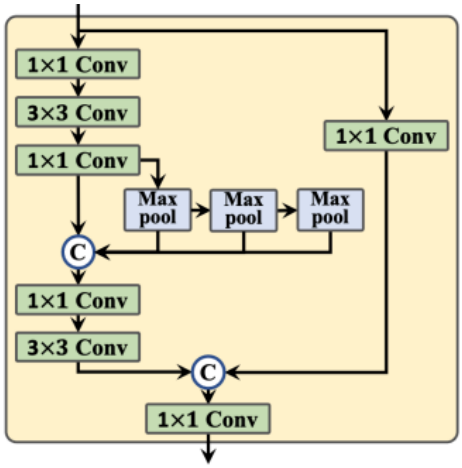


Рисунок 2.3 — Схематичне зображення блоку SimCSPSPPF

Детальний аналіз функціонування та взаємодії модулів у структурі YOLOv6 виходить за рамки цього дослідження. Зазначу лише, що ці компоненти працюють узгоджено, забезпечуючи ефективне вилучення ознак, адаптивну обробку об'єктів різного розміру та точність кінцевих прогнозів.

Головка в YOLOv6 відіграє важливу роль у генеруванні кінцевих результатів моделі. Вона відповідає за формування вихідних даних, які включають координати виявлених об'єктів, їх класи та масштаби. У YOLOv6 головка базується на архітектурі

YOLOv3, але з вдосконаленнями, що дозволяють досягти високої точності та швидкості обробки. Схематичне зображення архітектури головки зображено на рис. 2.4.

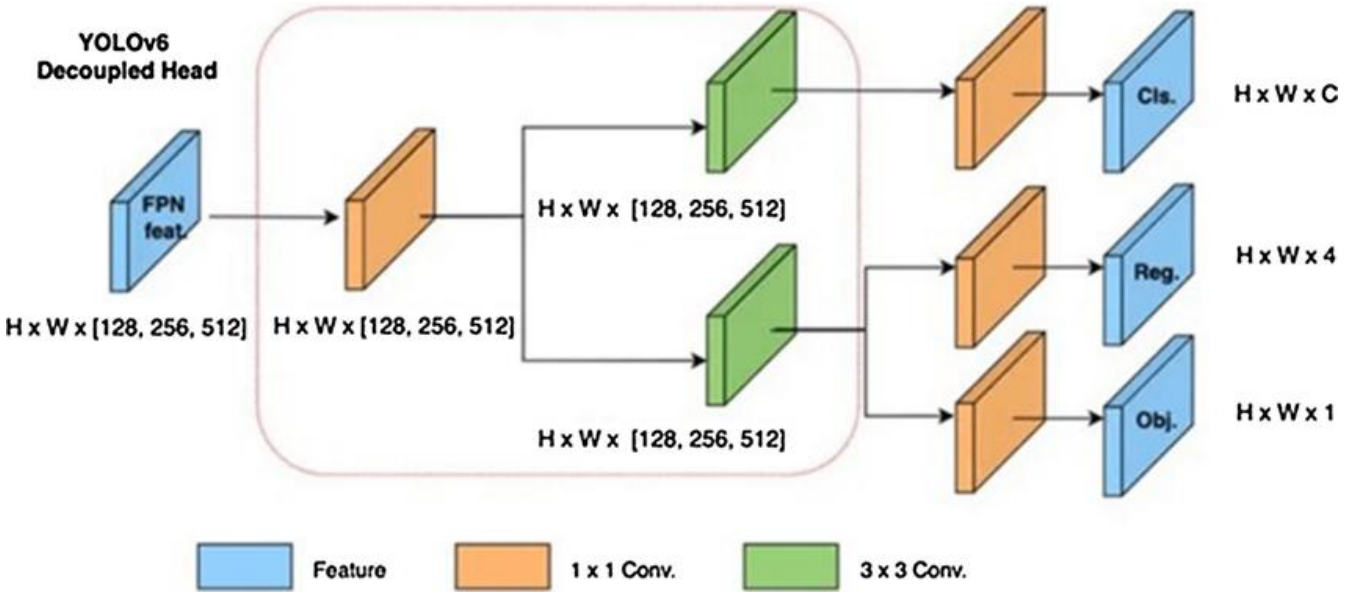


Рисунок 2.4 — Схематичне зображення модифікованої архітектури головки YOLOv6

FPN feat: це функції (features), отримані з Feature Pyramid Network (FPN). Вони передаються на вхід головки, і їх розмірність позначена як $H \times W \times [128,256,512]$

H та W — висота та ширина карти ознак.

[128,256,512] — кількість каналів залежно від рівня карти ознак.

Після обробки карти ознак через блоки 1x1 і 3x3 згорток (Convolutions) розділяється на три окремі гілки:

— Cls (Classification): класифікація об'єктів

— Reg (Regression): регресія для передбачення координат меж об'єктів

— Obj (Objectness): передбачення ймовірності того, що у певній області є об'єкт

1x1 Convolution (помаранчеві блоки) використовуються для зменшення або збільшення кількості каналів без зміни просторових розмірів ($H \times W$). Це зменшує обчислювальну складність та покращує адаптацію моделі до різних типів задач.

3x3 Convolution (зелені блоки) використовуються для вилучення локальних ознак із сусідніх пікселів. Ці згортки забезпечують точність визначення меж об'єктів.

Усі три гілки формують кінцеві передбачення:

Cls (Classification): $H \times W \times C$, де C — кількість класів.

Reg (Regression): $H \times W \times 4$, де 4 означає координати меж прямокутника (x, y, ширина, висота).

Obj (Objectness): $H \times W \times 1$, де 1 відповідає ймовірності наявності об'єкта.

Технологія AAT, що поєднує анкор-орієнтовані та безанкор-орієнтовані підходи, робить модель універсальною та ефективною для різноманітних задач. Self-Distillation підвищує продуктивність менших моделей, дозволяючи досягати високих результатів навіть на обмежених ресурсах.

Завдяки сучасним архітектурним рішенням YOLOv6 стала ідеальним вибором для широкого спектра завдань: від відеоспостереження до автономних систем. Вона забезпечує унікальний баланс між швидкістю та точністю, що робить її незамінним інструментом для задач реального часу, де швидкість прийняття рішень є критично важливою.

2.2 Створення механізму отримання інформації з камер та її передавання до бібліотеки штучного зору в системі автопілотажу

C# WinForms є потужним інструментом для створення інтуїтивних графічних інтерфейсів користувача (GUI), що відзначається рядом суттєвих переваг у контексті сучасного розроблення програмного забезпечення. Завдяки своїй простоті використання та інтеграції з екосистемою Microsoft .NET, ця технологія залишається актуальною у багатьох прикладних задачах, зокрема у сфері створення інтерфейсів, орієнтованих на кінцевого користувача.

Однією з ключових переваг C# WinForms є її висока доступність для розробників. Технологія надає широкий набір вбудованих компонентів, які дозволяють швидко і зручно створювати складні графічні елементи, такі як кнопки,

текстові поля, таблиці, вкладки та інші. Завдяки цьому забезпечується можливість розроблення функціональних прототипів у найкоротші терміни, що є особливо важливим у сучасних умовах стрімкого впровадження технологій.

WinForms використовує подієво-орієнтовану модель програмування, яка дозволяє ефективно обробляти дії користувача, такі як натискання кнопок, введення даних чи зміну стану елементів. Ця модель сприяє чіткому розмежуванню логіки інтерфейсу від основної бізнес-логіки, що забезпечує високий рівень підтримуваності та масштабованості коду. Крім того, доступна можливість створення власних користувацьких елементів управління, що дозволяє адаптувати інтерфейс до унікальних вимог проєкту.

Іншою важливою перевагою WinForms є її інтеграція з Visual Studio – однією з найпотужніших середовищ розробки програмного забезпечення. Завдяки цьому розробник отримує доступ до інструментів візуального проєктування, які значно спрощують процес створення інтерфейсів. Компоненти можна додавати на форму шляхом простого перетягування, що мінімізує кількість ручного коду та зменшує ймовірність помилок. Visual Studio також підтримує широкий спектр функціональних можливостей, таких як інтерактивне налагодження, автодоповнення коду та інтеграція з системами контролю версій, що сприяє підвищенню продуктивності розробки.

Окремо варто відзначити простоту розгортання програм, створених за допомогою WinForms. Зазвичай такі додатки не вимагають складних конфігурацій або залежностей, що значно спрощує процес впровадження програмного забезпечення у корпоративному середовищі. Легкість у розгортанні також позитивно впливає на кінцевого користувача, оскільки зменшується час, необхідний для встановлення та налаштування додатків.

WinForms також підтримує високий рівень кастомізації інтерфейсу. Розробники мають змогу застосовувати різноманітні стилі оформлення, включаючи створення унікальних дизайнів, які відповідають вимогам бренду чи специфіці використання.

Крім того, технологія забезпечує підтримку двовимірної графіки через GDI+ (Graphics Device Interface Plus), що дозволяє створювати складні візуальні ефекти, діаграми та візуалізації даних.

Ще однією значущою перевагою є можливість інтеграції WinForms із зовнішніми бібліотеками та сервісами, що дає змогу розширювати базовий функціонал додатків, додаючи, наприклад, підтримку баз даних, інструменти аналітики чи інтеграцію з хмарними сервісами. WinForms також дозволяє використовувати сторонні компоненти UI.

C# WinForms демонструє високу продуктивність у роботі з додатками, які потребують низького рівня затримок у відгуках інтерфейсу. Завдяки оптимізації виконання програмного коду у середовищі .NET, користувачі отримують швидкі та плавні взаємодії з додатком, що є важливим аспектом при створенні інтуїтивних і зручних GUI. Крім того, низькі вимоги до ресурсів апаратного забезпечення дозволяють запускати WinForms-додатки навіть на пристроях із обмеженими можливостями.

На додаток до вищезазначеного, WinForms забезпечує широку підтримку локалізації, що дозволяє легко адаптувати інтерфейси для користувачів із різних регіонів. Це досягається завдяки вбудованим механізмам обробки ресурсних файлів, які дозволяють створювати багатомовні інтерфейси без значних зусиль. Така гнучкість сприяє розробленню додатків, орієнтованих на міжнародний ринок.

Таким чином, використання C# WinForms для створення інтуїтивного графічного інтерфейсу користувача має низку вагомих переваг, які роблять цю технологію ефективним інструментом у сфері розроблення програмного забезпечення. Її простота, гнучкість, продуктивність і сумісність із сучасними стандартами дозволяють забезпечити високий рівень якості продукту та задоволення кінцевого користувача. Завдяки цьому WinForms залишається одним із популярних виборів для реалізації GUI у різноманітних програмних рішеннях.

DirectX 11, як графічна бібліотека, розроблена компанією Microsoft, демонструє значні переваги в роботі з відеопотоками та обробці зображень, особливо у поєднанні з C# WinForms. Завдяки інтеграції цих технологій розробники можуть створювати високопродуктивні додатки з інтуїтивним інтерфейсом користувача та потужними можливостями обробки графіки в реальному часі.

Однією з ключових переваг DirectX 11 є його архітектура, оптимізована для роботи з багатоядерними процесорами. Це дозволяє розподіляти обчислювальні навантаження між кількома ядрами, забезпечуючи високу продуктивність і плавність відтворення відеопотоків. У контексті C# WinForms, де інтерфейс користувача повинен реагувати на дії користувача без затримок, така продуктивність має вирішальне значення. Розробники можуть використовувати DirectX 11 для виконання складних обчислень у графічному потоці, залишаючи головний потік доступним для обробки взаємодії з користувачем.

DirectX 11 також пропонує широкий набір інструментів для роботи з текстурами та зображеннями, що є критично важливим для обробки відеопотоків. Завдяки підтримці таких функцій, як шейдери пікселів і вершин, розробники можуть реалізовувати складні алгоритми обробки зображень, включаючи фільтрацію, покращення якості та виявлення об'єктів. У поєднанні з C# WinForms це відкриває можливості для створення додатків, які можуть виконувати аналіз і трансформацію відеоданих у реальному часі, залишаючись доступними й простими у використанні для кінцевих користувачів.

Ще однією перевагою DirectX 11 є підтримка апаратного прискорення, що дозволяє ефективно використовувати графічні процесори (GPU) для обробки зображень і відео. Це особливо важливо в задачах, які потребують високої продуктивності, наприклад, обробка потокового відео або виконання алгоритмів машинного зору. Використовуючи DirectX 11 у додатках C# WinForms, розробники можуть перенести обчислення з центрального процесора на графічний, знижуючи загальне навантаження на систему та підвищуючи швидкодію додатка.

Крім того, DirectX 11 підтримує широкий спектр форматів відео та зображень, що спрощує інтеграцію з різними джерелами даних. Це забезпечує гнучкість у роботі з потоками відео з камер, збереженими відеофайлами або іншими джерелами мультимедіа. У середовищі C# WinForms це дозволяє легко реалізовувати інтерфейси для роботи з відеоданими, включаючи перегляд, редагування або аналіз.

Ефективність роботи з великими обсягами даних також є важливою характеристикою DirectX 11. Його оптимізовані механізми роботи з пам’яттю дозволяють швидко завантажувати та обробляти зображення, мінімізуючи затримки. Це особливо важливо для інтерактивних додатків, створених за допомогою C# WinForms, де затримки можуть негативно впливати на досвід користувача. Наприклад, у додатках для відеомоніторингу або візуалізації даних реального часу затримка в обробці кадрів може призвести до втрати критичної інформації.

Також важливо відзначити високу інтеграцію DirectX 11 із Windows, що забезпечує стабільність і надійність роботи додатків. У поєднанні з C# WinForms, розробники отримують доступ до широких можливостей платформи .NET, що включає багатий набір бібліотек і засобів для створення графічного інтерфейсу. Це спрощує процес розробки та знижує витрати на підтримку та оновлення додатків.

Таким чином, використання DirectX 11 у поєднанні з C# WinForms відкриває широкі можливості для створення потужних і зручних додатків. Завдяки своїй продуктивності, гнучкості та інтеграції з сучасними технологіями, ця комбінація є ідеальним вибором для розробників, які працюють над задачами, пов’язаними з обробкою відеопотоків і зображень.

Можливість інтеграції з різноманітними джерелами відео є ключовою перевагою використання DirectX 11 у поєднанні з C# WinForms, що відкриває широкі перспективи для розробників додатків, орієнтованих на обробку відео та взаємодію з графічними даними. Завдяки гнучкості DirectX 11, програми можуть легко підключатися до різних джерел відеопотоків, таких як камери, файли, стримінгові платформи чи навіть інші графічні вікна у системі. Це забезпечується через

низькорівневий доступ до GPU, який дозволяє ефективно захоплювати, обробляти та передавати зображення з мінімальними затримками.

Інтеграція з різними джерелами відео також стає можливою завдяки механізмам захоплення кадрів, доступним у DirectX 11. Розробники можуть створити програму, яка зчитує відеопотоки з різноманітних джерел, таких як IP-камери, веб-камери або зовнішні пристрої захоплення, і обробляє їх для подальшого використання. Ця обробка може включати масштабування, корекцію кольорів, стабілізацію відео, або навіть використання алгоритмів комп’ютерного зору для аналізу зображень.

Крім цього, DirectX 11 надає можливість взаємодії з іншими вікнами, відкритими у системі, що дозволяє створювати рішення для передачі відеоінформації між програмами. Це особливо корисно для завдань моніторингу, знімання екранів або створення складних мультифункціональних систем, які об'єднують дані з різних джерел в одному інтерфейсі. Наприклад, програма може захоплювати графічну інформацію з будь-якого вікна в системі та передавати її до інших модулів для аналізу чи зберігання.

Використання DirectX 11 у тандемі з C# WinForms створює ефективну основу для розробки додатків із широкими можливостями інтеграції відеопотоків. Завдяки високій продуктивності DirectX 11 і простоті створення інтуїтивного інтерфейсу у WinForms, розробники отримують інструменти для реалізації інноваційних рішень, які можуть легко адаптуватися до різних потреб користувачів.

2.3 Створення архітектури багатошарового перцептрону для прийняття релевантного рішення в системі авопілотажу

Застосування бібліотеки YOLO v6 уможливлює розпізнавання об'єктів з ідентифікацією трьох основних параметрів для кожного об’єкта: координати обмежувальної рамки (bounding box), клас об’єкта та рівень впевненості у визначенні (confidence score).

Координати обмежувальної рамки визначають місцезнаходження та розміри об’єкта на зображенні. Це набір значень, що включає x та y координати верхнього лівого кута об’єкта.

Ширина (width) та висота (height) — розміри прямокутної області, яка охоплює об’єкт.

Ці дані дозволяють локалізувати об'єкт у просторі зображення. Наприклад, для автомобіля бібліотека визначає його розташування у пікселях, створюючи прямокутник, що точно відображає його межі. Координати є основою для подальшої обробки, наприклад, для аналізу руху, вимірювання відстані між об’єктами або взаємодії з іншими алгоритмами.

YOLO v6 має можливість класифікувати об’єкти за заздалегідь визначеними класами. До прикладів класів можна віднести:

— Транспортні засоби (автомобіль, вантажівка, автобус);

— Люди (пішоходи, велосипедисти);

— Елементи інфраструктури (дорожні знаки, світлофори).

Confidence score — це показник, що вказує на впевненість системи у правильності свого розпізнавання. Значення варіюється від 0 до 1:

— Близьке до 1 означає високу впевненість;

— Близьке до 0 свідчить про низьку ймовірність правильного визначення.

Цей параметр є особливо важливим у випадках, коли декілька об’єктів можуть бути схожими за формою чи кольором. Наприклад, при виявленні пішохода серед густого натовпу система аналізує отримані результати, щоб уникнути помилкового визначення. У реальному застосуванні цей показник часто використовується як поріг: якщо confidence score перевищує певне значення (наприклад, 0.5), об’єкт вважається розпізнаним.

Як зазначено в пункті 3.2, бібліотека використовує нейронні мережі, які були навчені на великих наборах даних, що включають тисячі зображень із позначеними об’єктами. Завдяки цьому система може точно визначати тип кожного розпізнаного

об’єкта. Для прикладу, камера системи автоматичного водіння може розпізнати пішоходів та дорожні знаки, що дозволяє автопілоту приймати відповідні рішення.

Щоб розрахувати кількість вхідних, прихованих та вихідних нейронів у багатошаровому перцептроні (MLP) для автопілота на основі даних від YOLO v6, скористаємося наданими параметрами та загальними рекомендаціями.

Отже, вхідні дані для MLP складаються з інформації від YOLO v6 і додаткових сенсорних даних від автомобіля:

— загальна кількість параметрів на один об'єкт: 6 (4 координати bounding box + 1 клас об'єкта + 1 ймовірність);

— максимальна кількість об'єктів на кадр: 10.

Максимальна кількість об'єктів на кадр, 10, була визначена як оптимальне значення на основі експериментальних досліджень. Під час експериментів з використанням 5, 10 та 15 об'єктів було виявлено, що найбільш оптимальним є параметр з 10 об'єктами, оскільки він показав найоптимальніші результати як за точністю, так і за продуктивністю.

Середня кількість об'єктів, які одночасно перебувають у полі зору автомобіля під час руху в міському середовищі, інколи може перевищувати 10. Це включає пішоходів, інші транспортні засоби, дорожні знаки, світлофори та інші елементи інфраструктури. Однак, навіть у складних ситуаціях, таких як перехрестя з великою кількістю учасників дорожнього руху, кількість критичних об'єктів, які потребують розпізнавання для прийняття рішень, зазвичай не перевищує порогу в 10 об'єктів.

Результати експериментів підтверджують, що при використанні 10 об'єктів система демонструє оптимальну продуктивність. Це значення забезпечує ефективну роботу з високою точністю та швидкістю обробки. Графік продуктивності, який ілюструє результати експериментів, наведений на рис 2.4.

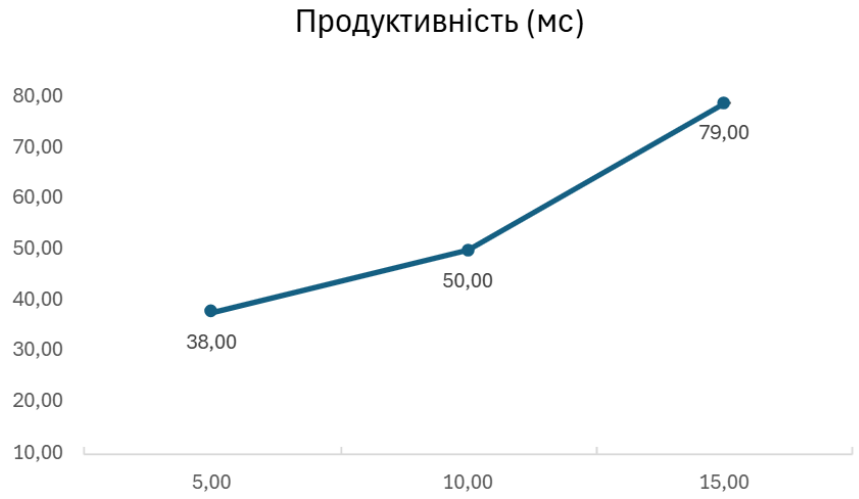


Рисунок 2.4 – Вплив максимальної кількості об'єктів на швидкість обробки

Обмеження кількості об'єктів зменшує ризик перевантаження системи та надмірного використання обчислювальних ресурсів, що є важливим для забезпечення стабільної роботи системи автопілота.

Оскільки багатошаровий перцептрон складається з кількох шарів: вхідного шару, одного або кількох прихованих шарів і вихідного шару та кожен нейрон у шарах з'єднаний з усіма нейронами наступного шару (повнозв'язна архітектура), то:

Розрахуємо кількість вхідних нейронів для MLP від YOLO v6:

$$N_{YOLO} = 6 \cdot 10 = 60.$$

Розглянемо додаткові сенсорні дані:

– поточна швидкість: 1 нейрон;

– поточний кут керма: 1 нейрон;

– лівий поворотник увімкнено: 1 нейрон;

– правий поворотник увімкнено: 1 нейрон;

– нахил дороги: 1 нейрон;

– поточне прискорення/сповільнення: 1 нейрон;

– загальна кількість додаткових вхідних нейронів:

$$N_{extra} = 1 + 1 + 1 + 1 + 1 + 1 = 6.$$

– загальна кількість вхідних нейронів:

$$N_{input} = N_{YOLO} + N_{extra} = 60 + 6 = 66.$$

Для автопілота необхідно приймати рішення щодо управління такими функціями, як:

– рульове управління (поворот керма): 1 нейрон;

– прискорення/гальмування: 1 нейрон;

– лівий поворотник: 1 нейрон;

– правий поворотник: 1 нейрон;

Загальна кількість вихідних нейронів:

$$N_{output} = 4.$$

Вибір кількості нейронів у прихованих шарах може базуватися на емпіричних правилах та експериментальних результатах. Один із поширених підходів — використовувати кількість нейронів у прихованих шарах як середнє арифметичне між кількістю вхідних та вихідних нейронів.

Отже перший прихований шар містить відповідно:

$$N_{hidden1} = 32 \cdot N_{input} + N_{output} \cdot 32 \cdot 66 + 4 \approx 48 \text{ (нейронів)}.$$

Другий прихований шар містить таку кількість нейронів:

$$N_{hidden2} = 32 \cdot N_{hidden1} \approx 32 \cdot 48 \approx 32.$$

Отже, архітектура багатошарового перцептрону описується такими параметрами:

– вхідний шар: 66 нейронів;

– прихований шар 1: 48 нейронів;

– прихований шар 2: 32 нейрони;

– вихідний шар: 4 нейрони.

Для прихованих шарів найпоширенішою функцією активації є ReLU (Rectified Linear Unit). Ця функція визначається як:

$$f(x) = \max(0, x).$$

Ефективне навчання: Вона дозволяє швидше конвергувати алгоритму навчання завдяки простій реалізації. На відміну від сигмоїдної чи гіперболічного тангенса, ReLU не насичується при великих значеннях входу. Завдяки тому, що градієнт є постійним для додатних входів, модель швидше оновлює ваги під час навчання.

Графічний вигляд ReLU зображено на рис 2.5.

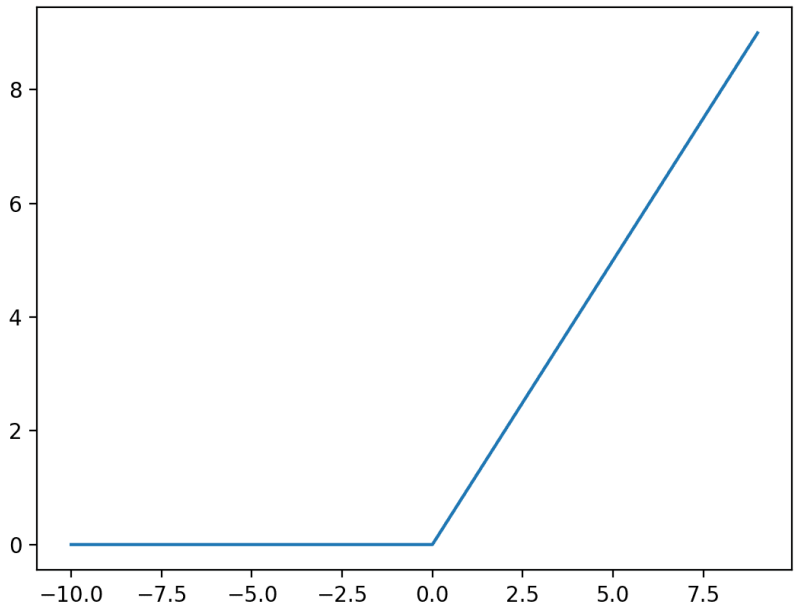


Рисунок 2.5 – Графічний вигляд функції активації ReLU

Для вихідних нейронів, які керують кермом і швидкістю, обирається гіперболічний тангенс. Ця функція має вигляд:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Вихідний діапазон значень [−1,1], що зручно для моделювання як позитивних, так і негативних значень.

Схожа поведінка до сигмоїди, але з більшим градієнтом для середніх значень, що сприяє швидшому навчанню.

Ця функція ідеально підходить для завдань, де потрібна симетрія вихідних значень, наприклад, коли потрібно визначити напрямок (вліво чи вправо) або швидкість (прискорення чи уповільнення).

Графічний вигляд функції гіперболічного тангенса зображено на рис 2.6.

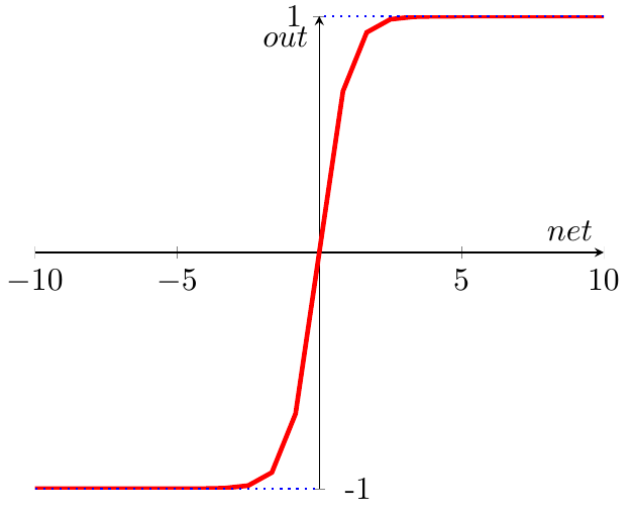


Рисунок 2.6 – Графічний вигляд функції активації гіперболічний тангенс

Для вихідних нейронів, що відповідають за керування поворотниками, доцільно використовувати сигмоїдну функцію. Вона визначається як:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Діапазон значень [0,1] що дозволяє інтерпретувати вихід як ймовірність.

Проста інтерпретація: значення ближче до 1 сигналізує про необхідність увімкнення поворотника, тоді як значення ближче до 0 — про його вимкнення.

Сигмоїда використовується для бінарних класифікаційних завдань, таких як прийняття рішення "увімкнути/вимкнути поворотник".

Графічний вигляд сигмоїдної функції зображено на рис. 2.7.

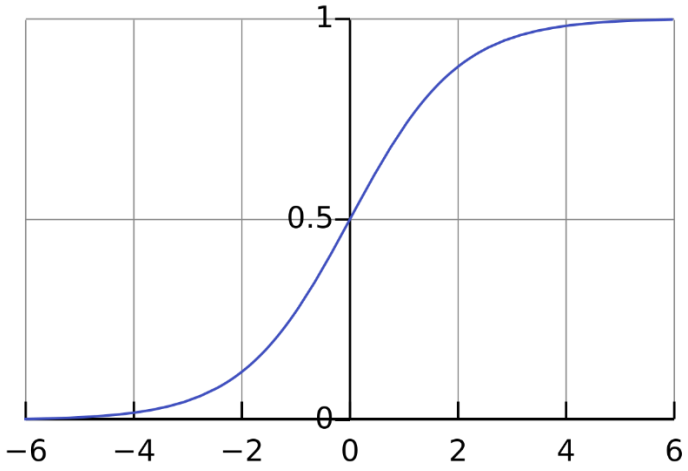


Рисунок 2.7 – Графічний вигляд функції активації сигмоїдної функції

Загальну кількість параметрів моделі можна розрахувати як суму вагових коефіцієнтів та зміщень (bias):

— між вхідним шаром і першим прихованим шаром:

$$P_{input\text{-}hidden1} = N_{input} \cdot N_{hidden1} + N_{hidden1} = 66 \cdot 48 + 48 = 3216;$$

— між першим і другим прихованими шарами:

$$P_{hidden1\text{-}hidden2} = N_{hidden1} \cdot N_{hidden2} + N_{hidden2} = 48 \cdot 32 + 32 = 1568;$$

— між другим прихованим шаром і вихідним шаром:

$$P_{hidden2\text{-}output} = N_{hidden2} \cdot N_{output} + N_{output} = 32 \cdot 4 + 4 = 132;$$

— загальна кількість параметрів:

$$P_{total} = 3216 + 1568 + 132 = 4916.$$

Отже, модель багатошарового перцептрону має таку архітектуру, що представлено на рисунку 2.8.

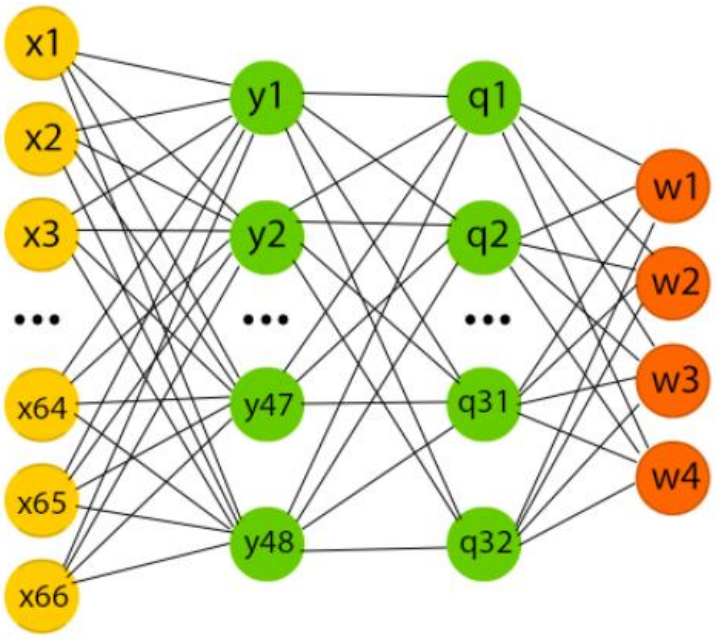


Рисунок 2.8 – Багатошаровий перцептрон для ПР щодо керування авомобілем

2.4 Підвищення продуктивності системи автопілотажу засобами застосування навченого генетичними алгоритмами багатошарового перцептрону

Backpropagation, або зворотне поширення помилки, — це один із ключових алгоритмів навчання багатошарових нейронних мереж, який використовується для налаштування ваг, щоб зменшити похибку між прогнозованим результатом і бажаним виходом. Він є основою більшості сучасних підходів до навчання моделей машинного навчання, особливо тих, що працюють із глибоким навчанням. Алгоритм працює на основі правила градієнтного спуску, забезпечуючи оптимізацію функції втрат, що описує похибку моделі.

Робота Backpropagation починається з прямого проходу через нейронну мережу, коли дані вводяться до вхідного шару, передаються через приховані шари, а на виході отримується результат. Цей етап називається feedforward. На основі результату обчислюється функція втрат — показник того, наскільки отриманий результат відрізняється від очікуваного. Типовими функціями втрат можуть бути

середньоквадратична похибка або крос-ентропія. Величина втрати є ключовою метою мінімізації в процесі навчання моделі.

Коли похибка визначена, Backpropagation починає працювати у зворотному напрямку. Алгоритм обчислює, наскільки кожен нейрон у мережі сприяв загальній похибці. Цей процес називається зворотне поширення помилки. Для цього використовується похідна функції втрат за кожною вагою, що дозволяє оцінити, наскільки зміна конкретної ваги вплине на загальну похибку. Ідея базується на правилі ланцюга, яке в математиці дозволяє обчислювати похідні складних функцій через похідні їх складових. У нейронній мережі функція, яку потрібно оптимізувати, є комбінацією активацій нейронів, зважених сум і нелінійних функцій активації.

У зворотному проході для кожного нейрона обчислюється його локальна похибка, яка залежить від його внеску в похибку наступного шару. Для вихідного шару похибка легко обчислюється як різниця між фактичним виходом і очікуваним значенням. Для прихованих шарів похибка визначається через вагові коефіцієнти і похибки наступного шару. Важливим етапом є врахування похідної функції активації кожного нейрона, яка модулює вплив ваги на результат. Наприклад, у функції ReLU (Rectified Linear Unit) похідна дорівнює 1, коли значення більше нуля, і 0 в іншому випадку. Для сигмоїдальної функції активації похідна залежить від самого значення виходу нейрона.

Після визначення похибок алгоритм оновлює ваги з використанням градієнтного спуску. Кожна вага коригується на величину, пропорційну градієнту функції втрат за цією вагою. Величина зміни ваги залежить також від параметра, який називається швидкістю навчання. Цей параметр контролює, наскільки великі кроки робляться під час коригування ваг, і допомагає запобігти як надто повільному навчанню, так і нестабільності алгоритму.

Демонстрації роботи алгоритму наведено на рисунку 2.9.

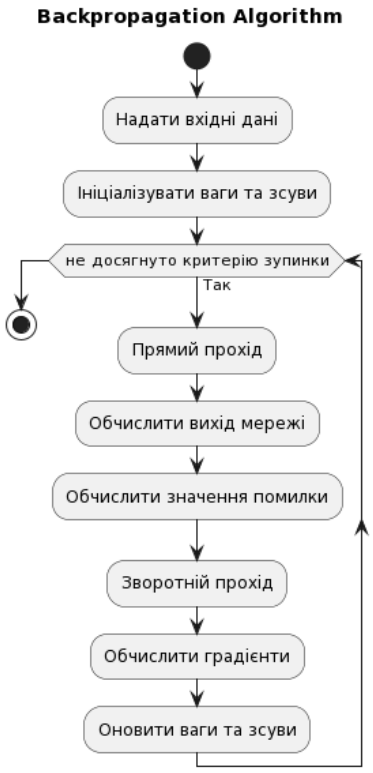


Рисунок 2.9 — Схема алгоритму зворотного поширення

Backpropagation є ефективним завдяки здатності розподіляти похибку через всю мережу. Це дозволяє одночасно оптимізувати всі ваги і налаштовувати навіть складні багатошарові структури. Однак він має кілька недоліків, таких як проблема зникнення градієнтів у дуже глибоких мережах, де похідні на початкових шарах стають надто малими, щоб забезпечити суттєві зміни ваг. Цю проблему частково розв’язали сучасні архітектури мереж та функції активації, такі як ReLU або Leaky ReLU, а також методи оптимізації, такі як Adam чи RMSProp.

Під час навчання Backpropagation зазвичай виконується на багатьох прикладах одночасно, що називається пакетною обробкою (mini-batch). Це дозволяє стабілізувати процес оптимізації та ефективно використовувати обчислювальні ресурси. У великих нейронних мережах, таких як згорткові чи рекурентні, алгоритм часто використовується разом із вдосконаленими техніками, такими як регуляризація чи нормалізація шарів, що покращує загальну продуктивність моделі.

Завдяки Backpropagation нейронні мережі стали основою сучасного машинного навчання, забезпечивши прориви в обробці зображень, текстів, аудіо та багатьох інших завдань. Алгоритм залишається одним із найпотужніших інструментів для оптимізації моделей, дозволяючи їм досягати високої точності та узагальнювати дані на реальних прикладах.

Об'єднання генетичного алгоритму та методу зворотного поширення помилки дозволяє створити потужний підхід до навчання нейронних мереж, у якому поєднуються переваги обох методів. Генетичний алгоритм виконує глобальний пошук, створюючи велику кількість популяцій моделей із різними параметрами. Завдяки цьому він швидко знаходить області в просторі ваг, які забезпечують відносно хороші результати. Цей етап допомагає уникнути застрягання в локальних мінімумах, що є типовою проблемою для Backpropagation, особливо у задачах із великою кількістю параметрів.

Після завершення глобального пошуку генетичний алгоритм передає результати Backpropagation, який використовується для точного налаштування ваг. Це дозволяє покращити якість моделі, досягти високої точності та забезпечити кращу здатність до узагальнення даних. Генетичний алгоритм слугує основою для швидкого пошуку ефективних параметрів, а Backpropagation виконує детальне доопрацювання, налаштовуючи модель на основі похибок, що виникають у процесі навчання.

Таке поєднання допомагає подолати недоліки обох методів. Генетичний алгоритм не обмежується лише градієнтною інформацією, яка може бути неточною або неефективною, а Backpropagation уникає потреби в надмірних обчислювальних ресурсах для глобального пошуку. У результаті цей підхід дозволяє значно пришвидшити процес навчання нейронної мережі, підвищуючи її ефективність і забезпечуючи високу якість моделі. Це робить його ідеальним для розв'язання складних задач, де традиційні методи могли бути неефективними.

2.5 Побудова апаратної частини системи автопілотажу

Апаратна частина системи автопілотажу є важливим елементом, оскільки вона забезпечує отримання, обробку та передачу даних, необхідних для роботи програмного забезпечення, що реалізує функціонал автопілота. У цій розробці апаратна частина представлена мінімальним набором обладнання, що складається з IP-камери або іншого джерела відео, яке слугує головним сенсором системи, а також комп’ютера, на якому працює програмне забезпечення серверної частини. Такий підхід дозволяє побудувати систему, яка є відносно простою, гнучкою та масштабованою.

Основною функцією IP-камери є зйомка відео в режимі реального часу та передача його на серверну програму для подальшої обробки. IP-камера забезпечує високу роздільну здатність відео, яка критично важлива для коректної роботи системи комп’ютерного зору. Чіткість і деталізація зображення дозволяють покращити розпізнавання об’єктів, дорожньої розмітки, світлофорів та інших елементів оточення. Вибір IP-камери залежить від вимог системи та умов експлуатації. Наприклад, для міського середовища може знадобитися камера з широким кутом огляду та адаптацією до різних рівнів освітлення, тоді як для шосе або сільської місцевості більше значення матиме дальність зйомки.

Як альтернатива IP-камері, система також може використовувати інші джерела відео, зокрема відеопотоки з камер спостереження, дронів або мобільних пристроїв. Це робить систему більш універсальною та дає змогу адаптувати її до різних сценаріїв використання. Наприклад, у випадках, коли необхідно аналізувати відео, зняте попередньо, система може працювати з записаними файлами, зберігаючи алгоритми обробки незмінними.

Відео, отримане з камери, передається на серверну програму, яка розташована на комп’ютері. Цей комп’ютер є обчислювальним центром системи автопілотажу. Серверна частина виконує складну обробку відеопотоку, використовуючи алгоритми

комп’ютерного зору, такі як YOLO v6, які дозволяють виділяти та класифікувати об’єкти, а також відстежувати їх положення в кадрі. Для забезпечення стабільної роботи системи обчислювальна потужність комп’ютера повинна відповідати вимогам до швидкості обробки відео в реальному часі. Зазвичай це досягається використанням сучасних процесорів та графічних карт, які підтримують апаратне прискорення для задач машинного навчання.

Передача відео з IP-камери на комп’ютер виконується за допомогою мережевих протоколів, таких як RTSP або HTTP. Ці протоколи забезпечують стабільність передачі даних навіть у випадках з низькою пропускною здатністю мережі. Серверна програма приймає відеопотік, розбиває його на окремі кадри та передає їх на аналіз нейронною мережею. У свою чергу, оброблена інформація використовується для ухвалення рішень, таких як керування швидкістю транспортного засобу, поворотами чи гальмуванням.

Така структура апаратної частини системи дозволяє забезпечити її високу адаптивність. Заміна камери чи джерела відео на інше не потребує серйозних змін у програмному забезпеченні, що робить систему універсальною для використання в різних сценаріях. Цей підхід також дозволяє легко інтегрувати систему у вже існуючу інфраструктуру транспортних засобів чи систем моніторингу, що значно розширює її можливості та сферу застосування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОПІЛОТАЖУ

3.1 Встановлення та налаштування YOLO v6

Розглянемо процес налаштування YOLOv6 для створення середовища, що дозволяє використовувати можливості YOLOv6 для різних завдань, таких як аналіз зображень або відео, створення автоматизованих систем і багато іншого.

Для роботи YOLOv6 спочатку я встановив Python останньої версії. Python є мовою програмування, яка широко використовується для розробки нейронних мереж і роботи з бібліотеками машинного навчання. Наявність останньої версії гарантує сумісність із сучасними інструментами та бібліотеками.

Після цього я встановив pip – інструмент для керування пакетами Python. Pip дозволяє легко завантажувати та встановлювати необхідні бібліотеки, а також забезпечує автоматичне встановлення залежностей для таких великих проєктів, як YOLOv6.

Для автоматично встановлення залежностей та потрібних бібліотек використав команду: pip install yolov6

Це дозволило автоматично встановити всі необхідні бібліотеки та залежності, які потрібні для коректної роботи YOLOv6. Список встановлених компонентів включає:

— tensorboard-data-server: забезпечує візуалізацію та аналіз роботи нейронних мереж, особливо під час тренування моделей;

— six: бібліотека для підтримки сумісності між Python 2 і Python 3;

— setuptools: інструмент для встановлення і розповсюдження Python-пакетів;

— protobuf: формат обміну даними, який широко використовується у машинному навчанні для збереження та обміну моделями;

— packaging: бібліотека для аналізу та створення залежностей між пакетами;

— numpy: одна з найважливіших бібліотек для роботи з масивами даних і математичними обчисленнями;

— MarkupSafe: забезпечує захист від ін'єкцій шкідливого коду у веб-додатках;

— markdown: використовується для роботи з текстом у форматі Markdown;

— grpcio: фреймворк для міжпроцесної взаємодії та комунікації між компонентами системи;

— absl-py: допоміжна бібліотека для створення проєктів на Python;

— werkzeug: утиліта для розробки веб-додатків;

— tensorboard: інструмент для візуалізації роботи TensorFlow-моделей.

Після успішного виконання цієї команди всі ці пакети були встановлені, що підтверджується виводом в консоль який зображено на рис 3.1.

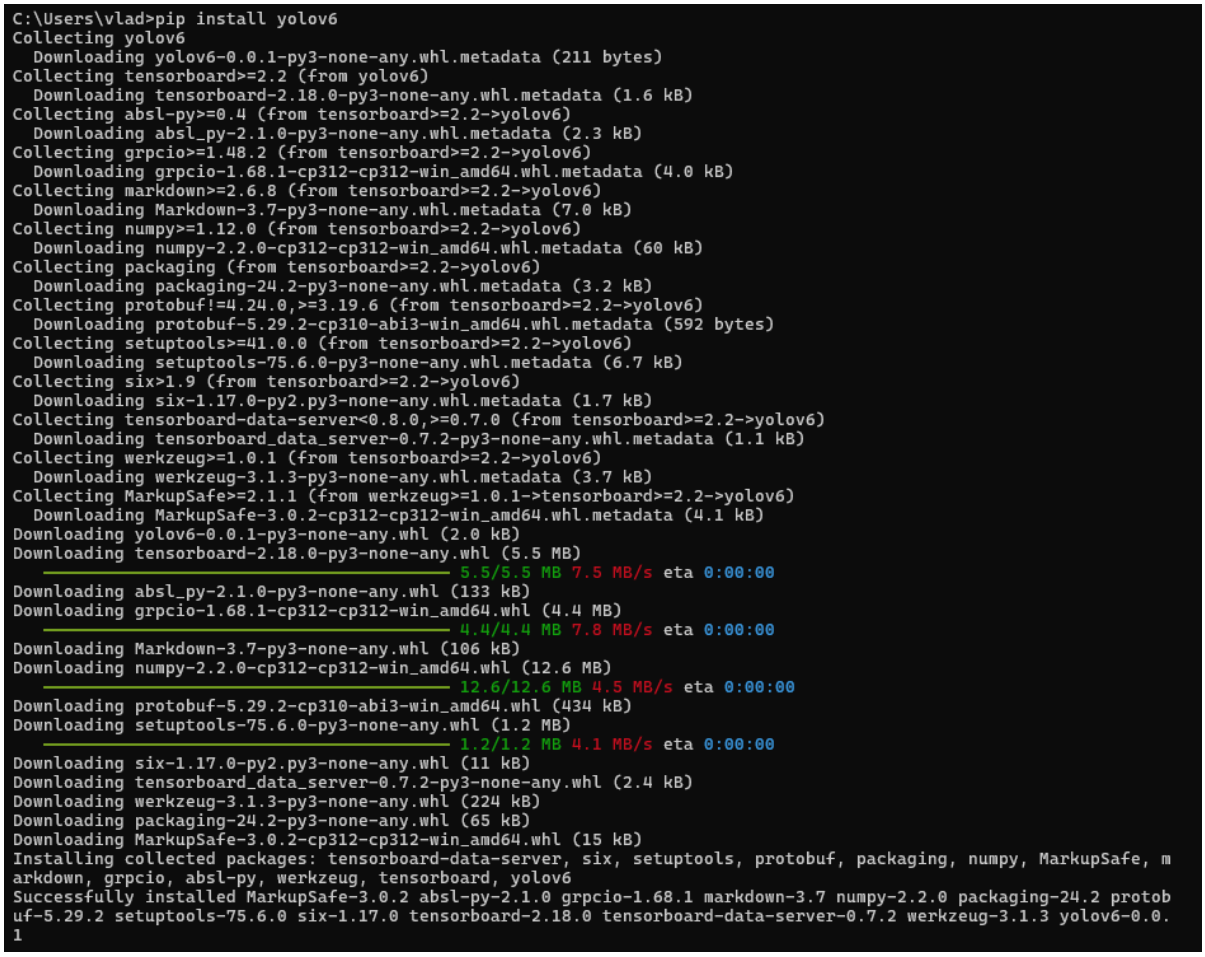


Рисунок 3.1 — Встановлення залежностей YOLO v6

Наступним кроком було завантаження Git – інструменту для роботи з системою контролю версій. Git дозволяє клонувати репозиторії з віддалених джерел, наприклад, із GitHub, що є зручним для завантаження великих проєктів, як-от YOLOv6.

Я виконав наступні команди для завантаження архітектури YOLOv6 з офіційного репозиторію:

git clone https://github.com/meituan/YOLOv6;

cd YOLOv6.

Ці команди клонували репозиторій, що містить повний код бібліотеки, документацію, приклади використання та готову до роботи модель YOLOv6. Після переходу в каталог YOLOv6 я отримав доступ до всіх файлів і налаштувань, необхідних для подальшої роботи з моделлю.

3.2 Формування тренувальних даних для YOLO v6

Для успішного тренування YOLOv6 потрібно підготувати датасет із чітко розміченими зображеннями.

Можна скористатися готовими наборами даних, доступними на популярних платформах, таких як COCO Dataset, Open Images Dataset та Kaggle Datasets. У цьому проєкті було обрано Open Images Dataset, оскільки він надає можливість вибору конкретних класів зображень, таких як Car, Person, Traffic Light.

Для початку роботи необхідно встановити інструменти, що забезпечують завантаження та обробку датасету. У цьому проєкті було обрано бібліотеку FiftyOne, яка є зручним і функціональним інструментом для роботи з набором даних Open Images. FiftyOne надає можливість ефективного завантаження, фільтрації й обробки даних, що робить її ідеальним вибором для виконання поставлених задач. Для встановлення бібліотеки достатньо виконати команду pip install fiftyone у середовищі Python.

Процес завантаження датасету передбачає підключення FiftyOne у Python для подальшої роботи. На першому етапі визначаються класи об'єктів, які становлять

інтерес для аналізу. Наприклад, у цьому випадку було обрано класи "Car", "Person" і "Traffic Light". Відповідно, у коді формується список таких класів, щоб зосередитися лише на потрібних категоріях зображень із великого набору даних Open Images.

Далі використовується функція load_zoo_dataset з бібліотеки FiftyOne для завантаження необхідних зображень разом із відповідними координатами обмежувальних рамок (Bounding Boxes). Параметри виклику функції налаштовуються залежно від вимог до вибірки. Наприклад, параметр split="train" визначає, що буде завантажено дані для тренувальної вибірки. Якщо потрібні дані для валідації моделі, цей параметр можна змінити на split="validation". Додатково задається максимальна кількість зображень за допомогою параметра max_samples, що дозволяє обмежити обсяг завантажених даних до необхідної кількості.

Отримані дані зберігаються в локальній директорії, яка задається через параметр dataset_dir. У цьому випадку вона була позначена як "open_images_dataset". Такий підхід дозволяє завантажити лише необхідні класи зображень і спростити їх подальшу обробку для навчання або тестування моделі. Таким чином, бібліотека FiftyOne забезпечує ефективний і структурований процес роботи з Open Images Dataset, значно полегшуючи підготовку даних для вирішення завдань комп'ютерного зору.

Для використання моделі YOLO необхідно підготувати дані у специфічному форматі розмітки, який відрізняється від формату, що використовується в Open Images Dataset. Кожне зображення повинно супроводжуватися текстовим файлом (.txt), який містить інформацію про всі Bound Boxes (обмежувальні рамки) об'єктів на зображенні. Кожен рядок у цьому файлі описує один Bound Box і має наступний формат: <Class ID> <x_center> <y_center> <width> <height>.

Параметр <Class ID> є числовим індексом, що відповідає певному класу об’єкта (наприклад, 0 — "Car", 1 — "Person", 2 — "Traffic Light"). Координати x_center, y_center, а також розміри рамки width і height повинні бути нормалізовані, тобто представлені у значеннях від 0 до 1 відносно ширини та висоти зображення.

Для виконання цієї конвертації було використано спеціальний скрипт на Python. Спочатку завантажуються анотації, які містять Bound Boxes з Open Images Dataset, та виконується відповідність між назвами класів і їх індексами. Потім створюється окрема папка для збереження файлів розмітки у форматі, що відповідає YOLO. Код продемонтстровано в лістингу 3.1.

Лістинг 3.1 — Python-скрипт для конвертації

```
import os
import pandas as pd
data_dir = "open_images_dataset"
label_file = os.path.join(data_dir, "labels/detections.csv")
annotations = pd.read_csv(label_file)
class_mapping = {"Car": 0, "Person": 1, "Traffic Light": 2}
labels_dir = os.path.join(data_dir, "labels/yolo")
os.makedirs(labels_dir, exist_ok=True)
for _, row in annotations.iterrows():
    image_id = row["ImageID"]
    class_name = row["LabelName"]
    x_min, x_max = row["XMin"], row["XMax"]
    y_min, y_max = row["YMin"], row["YMax"]
    if class_name not in class_mapping:
        continue
    x_center = (x_min + x_max) / 2
```

Продовження лістингу 3.1

```
    y_center = (y_min + y_max) / 2
    width = x_max - x_min
    height = y_max - y_min
    class_id = class_mapping[class_name]
    line = f"{class_id} {x_center} {y_center} {width} {height}\n"
```

```
    label_path = os.path.join(labels_dir, f"{image_id}.txt")
    with open(label_path, "a") as f:
        f.write(line)
```

Кожен рядок Bound Box обробляється таким чином: координати x_min, x_max, y_min, y_max, отримані з Open Images, перетворюються на нормалізовані значення центру рамки (x_center, y_center), а також її ширини і висоти. На основі цієї інформації формується рядок у форматі YOLO, де вказується відповідний <Class ID> і нормалізовані координати. Після цього створений рядок додається у відповідний .txt файл, ім'я якого відповідає ідентифікатору зображення.

Цей підхід автоматизує процес перетворення даних із Open Images Dataset у формат, сумісний із YOLO, та дозволяє швидко генерувати необхідні текстові файли для навчання моделі. Така методика спрощує інтеграцію різних наборів даних у робочі процеси комп'ютерного зору, забезпечуючи правильне налаштування та точність моделі.

3.3 Написання програми для передачі даних до YOLO v6

Для вирішення завдання розробки серверного додатка, здатного в реальному часі отримувати дані з будь-якого джерела, було реалізовано програму, яка обробляє відеопотік, розбиває його на окремі кадри та передає ці дані в модель YOLO v6 для аналізу. Для створення цього додатка було обрано платформу WPF (Windows Presentation Foundation) у поєднанні з бібліотекою SharpDX, яка забезпечує ефективну взаємодію з графічними API, такими як DirectX.

Основна функціональність програми полягає в обробці відеопотоку. Спочатку сервер отримує відео в реальному часі, яке може надходити з різних джерел, наприклад, з камер спостереження, відеофайлів або потокових платформ. Завдяки використанню бібліотеки SharpDX забезпечується доступ до низькорівневих функцій обробки графіки, що дозволяє виконувати операції з високою продуктивністю та мінімальними затримками.

Після отримання відеопотоку сервер розбиває його на окремі кадри. Це дозволяє працювати з кожним зображенням окремо, що є необхідним для подальшого аналізу за допомогою YOLO v6. Кадри передаються в модель YOLO для ідентифікації об'єктів і аналізу сцени. Для передачі даних у YOLO кадри перетворюються у формат, який підтримується моделлю, що забезпечує коректність вхідних даних і точність результатів.

Розробка на базі WPF дозволяє створити зручний і зрозумілий інтерфейс користувача, який може бути використаний для моніторингу роботи програми, відображення результатів аналізу або налаштування параметрів сервера. Використання SharpDX, у свою чергу, гарантує ефективну обробку графічних даних і забезпечує плавність роботи навіть при високих навантаженнях.

Після створення базового додатку за допомогою шаблону було розроблено клас DxWindow, який реалізує створення та відображення вікна, використовуючи бібліотеку SharpDX. SharpDX є обгорткою для DirectX 11, що забезпечує ефективний доступ до низькорівневих можливостей графічної обробки в Windows. Клас DxWindow відповідає за ініціалізацію графічного контексту та управління відображенням вікна, в якому буде виконуватися рендеринг зображень і кадрів відео.

В конструкторі класу DxWindow ініціалізуються два параметри: title — заголовок вікна та _captureMethod — інтерфейс для методів захоплення зображень (лістинг 3.2).

Лістинг 3.2 — Конструктор класу DxWindow

```
public DxWindow(string title, ICaptureMethod captureMethod)
{
    _title = title;
    _captureMethod = captureMethod;
}
```

Клас DxWindow реалізує інтерфейс IDisposable, що дозволяє коректно звільняти ресурси при закритті вікна, викликаючи методи Dispose для захоплення та інших ресурсів.

Використовується клас RenderForm для створення вікна з вказаним заголовком.

Встановлюється розмір клієнтської області вікна та налаштовується опис обміну (swap chain) для забезпечення подвійного буферизації, що дозволяє уникнути мерехтіння зображення при відображенні.

Далі відбувається створення пристрою та обміну. Створюється пристрій Direct3D 11 (Device) та обмін (SwapChain), який відповідає за управління буферами та відображенням на екрані.

За допомогою Device.CreateWithSwapChain створюється графічний пристрій, що взаємодіє з вікном.

Для рендерингу використовуються два шейдера: вершинний та піксельний. Вони компілюються з файлу Shader.fx який буде наведено в **додатку**.

Шейдери відповідають за трансформацію вершин (Vertex Shader) і розрахунок кольору пікселів (Pixel Shader).

Створюється буфер, що містить чотири вершини, які визначають координати та текстурні координати для простого квадрата (прямокутника), що буде відображатися на екрані (лістинг 3.3).

Лістинг 3.3 — Створення буфера для вершин

```
using var vertexes = Buffer.Create(device, BindFlags.VertexBuffer, new[]
{
    new Vertex { Position = new RawVector3(-1.0f, 1.0f, 0.5f), TexCoord = new RawVector2(0.0f, 0.0f) },
    new Vertex { Position = new RawVector3(1.0f, 1.0f, 0.5f), TexCoord = new RawVector2(1.0f, 0.0f) },
    new Vertex { Position = new RawVector3(-1.0f, -1.0f, 0.5f), TexCoord = new RawVector2(0.0f, 1.0f) },
    new Vertex { Position = new RawVector3(1.0f, -1.0f, 0.5f), TexCoord = new RawVector2(1.0f, 1.0f) }
});
```

Далі створюється опис стану семплера, який визначає як текстури будуть оброблятися при рендерингу (лістинг 3.4).

Лістинг 3.3 — Налаштування стану семплера

```
var samplerStateDescription = new SamplerStateDescription
{
   AddressU = TextureAddressMode.Wrap,
   AddressV = TextureAddressMode.Wrap,
   AddressW = TextureAddressMode.Wrap,
   Filter = Filter.MinMagMipLinear
};
```

Вікно має цикл рендерингу, що виконується, поки вікно не закрите. У цьому циклі відбуваються всі операції рендерингу, включаючи очистку буфера, завантаження текстури та рендеринг.

Для кожного кадру відбувається:

— очищення екрану;

— захоплення наступного кадру за допомогою методу захоплення;

— виведення на екран текстури або зображення.

Код який реалізує це наведено на лістингу 3.4.

Лістинг 3.4 — Рендеринг в циклі

```
RenderLoop.Run(form, () =>
{
   if (!_captureMethod.IsCapturing)
      _captureMethod.StartCapture(form.Handle, device, factory);
   device.ImmediateContext.ClearRenderTargetView(renderView, new
RawColor4(1.0f, 1.0f, 1.0f, 1.0f));
   using var texture2d = _captureMethod.TryGetNextFrameAsTexture2D(device);
   if (texture2d != null)
   {
      using var shaderResourceView = new ShaderResourceView(device, texture2d);
      device.ImmediateContext.PixelShader.SetShaderResource(0,
shaderResourceView);
   }
```

```
    device.ImmediateContext.Draw(4, 0);
    swapChain1.Present(1, PresentFlags.None, new PresentParameters());
});
```

Далі розроблено клас GraphicsCapture який реалізує метод захоплення графічного виведення екрану для програми Windows, використовуючи API Windows Runtime (WinRT).

Метод StartCapture наведений в лістингу 3.5.

Лістинг 3.4 — Метод StartCapture

```
public void StartCapture(IntPtr hWnd, Device device, Factory factory)
    {
    var capturePicker = new WindowPicker();
    var captureHandle = capturePicker.PickCaptureTarget(hWnd);
    if (captureHandle == IntPtr.Zero)
        return;
    _captureItem = CreateItemForWindow(captureHandle);
    if (_captureItem == null)
        return;
    _captureItem.Closed += CaptureItemOnClosed;
    var hr = NativeMethods.CreateDirect3D11DeviceFromDXGIDevice(device.NativePointer, out var pUnknown);
    if (hr != 0)
    {
        StopCapture();
        return;
    }
    var winrtDevice = (IDirect3DDevice)Marshal.GetObjectForIUnknown(pUnknown);
    Marshal.Release(pUnknown);
```

Продовження лістинг 3.5

```
    _captureFramePool = Direct3D11CaptureFramePool.Create(winrtDevice, DirectXPixelFormat.B8G8R8A8UIntNormalized, 2, _captureItem.Size);
    _captureSession = _captureFramePool.CreateCaptureSession(_captureItem);
```

Продовження лістингу 3.5

```
    Cursor.Hide();
```

```
    _captureSession.StartCapture();
    IsCapturing = true;
}
```

Використовується WindowPicker, щоб вибрати вікно для захоплення, засноване на переданому дескрипторі вікна hWnd.

Створюється об'єкт GraphicsCaptureItem для вибраного вікна, що містить інформацію про вікно, яке буде захоплюватися.

Для захоплення графіки створюється сесія захоплення за допомогою Direct3D11CaptureFramePool і починається захоплення через метод StartCapture.

Після цього можна спробувати запустити програму, демонстрація роботи зображено на рисунках 3.2 та 3.3.

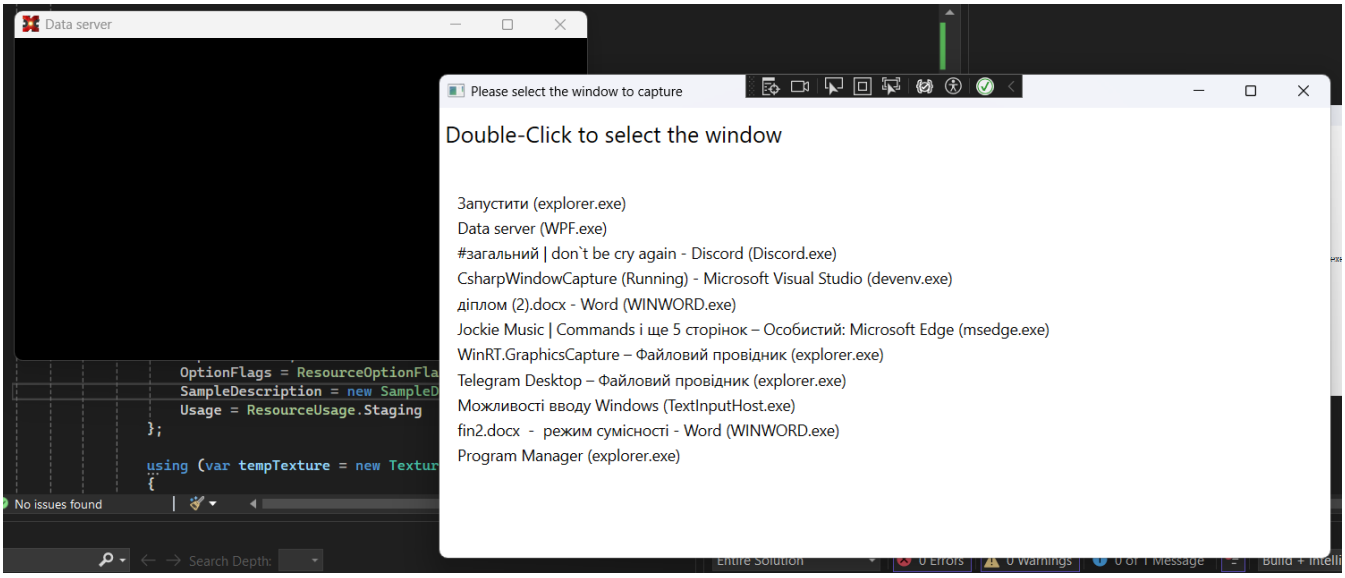


Рисунок 3.2— Вибір вікна для захоплення

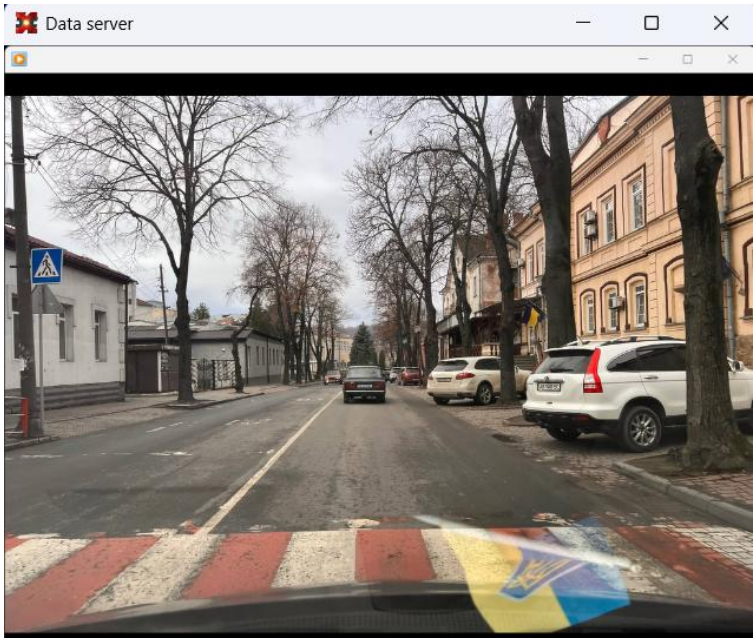


Рисунок 3.3— Програма працює

Метод SaveData який наведено на лістингу 3.6 створює тимчасовий текстурний об'єкт Texture2D з доступом до пікселів, копіюючи в нього дані з оригінальної текстури. Потім ці дані перетворюються в зображення за допомогою класу Bitmap і зменшуються до бажаного розміру, після чого готове зображення передається сокет в нейронну мережу.

Лістинг 3.6 Метод SaveDate

```
public void SaveDate(Socket socket, Device device, Texture2D texture, int targetWidth, int targetHeight) {
    if (texture != null){
        var textureDesc = new Texture2DDescription{
            ArraySize = 1,
            BindFlags = BindFlags.None,
            CpuAccessFlags = CpuAccessFlags.Read,
            Format = Format.B8G8R8A8_UNorm,
            Height = texture.Description.Height,
```

Продовження лістингу 3.6

```
Width = texture.Description.Width,
            MipLevels = 1,
            OptionFlags = ResourceOptionFlags.None,
```

```
                SampleDescription = new SampleDescription(1, 0),
                Usage = ResourceUsage.Staging
            };
            using (var tempTexture = new Texture2D(device, textureDesc)){
                device.ImmediateContext.CopyResource(texture, tempTexture);
                var dataBox = device.ImmediateContext.MapSubresource(tempTexture, 0,
MapMode.Read, SharpDX.Direct3D11.MapFlags.None);
                var width = tempTexture.Description.Width;
                var height = tempTexture.Description.Height;
                var bitmap = new Bitmap(width, height, stride,
PixelFormat.Format32bppPArgb, dataBox.DataPointer);
                var newBitmap = new Bitmap(targetWidth, targetHeight);
                using (var g = Graphics.FromImage(newBitmap)){
                    g.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
                    g.DrawImage(bitmap, 0, 0, targetWidth, targetHeight);
                }
                    using (var ms = new MemoryStream()){
                    newBitmap.Save(ms, ImageFormat.Png);
                    byte[] imageBytes = ms.ToArray();
                socket.Send(imageBytes);
                }
```

Продовження лістингу 3.6

```
            device.ImmediateContext.UnmapSubresource(tempTexture, 0);
            }
        }
    }
```

Сервер слухає на порту 12345. Як тільки він отримує з'єднання, він читає розмір зображення і потім отримує саме зображення.

Таким чином, створений серверний додаток є універсальним рішенням для отримання, обробки та аналізу відеопотоку в реальному часі. Завдяки використанню WPF і SharpDX вдалося досягти високої продуктивності та забезпечити інтеграцію з моделлю YOLO v6, що робить цей інструмент ефективним для задач комп’ютерного зору в реальному часі.

3.4 Інтеграція зворотного поширення помилки та генетичних алгоритмів

Коли я почав розробляти свій нейронний мережевий алгоритм, я вирішив реалізувати два основні компоненти: прямий прохід (Feedforward) і зворотне поширення помилки (Backpropagate). Обидва ці методи є ключовими для функціонування багатошарових перцептронів.

Мій метод Feedforward відповідає за передачу вхідних даних через нейронну мережу та обчислення результатів для кожного шару (лістинг 3.1).

Лістинг 3.1 — Передача вхідних значень

```
for (int i = 0; i < inputs.Length; i++)
{
   neurons[0][i] = inputs[i];
}
```

Тут inputs — це масив вхідних значень, який ініціалізує перший шар нейронів neurons[0].

Далі я створив цикл, який проходить через усі шари мережі, починаючи з другого (лістинг 3.2):

У кожному шарі я обчислюю значення кожного нейрона як зважену суму виходів із попереднього шару плюс зміщення (лістинг 3.2).

Далі додємо зміщення biases[i - 1][j] до кожного нейрона, а також множу ваги (weights) на значення виходів із попереднього шару (neurons[i - 1][k]).

Лістинг 3.2 — Обчислення значень для прихованих шарів

```
for (int i = 1; i < neurons.Length; i++)
{
float value = biases[i - 1][j] + BIAS;
for (int k = 0; k < neurons[i - 1].Length; k++)
{
   value += weights[weightLayerIndex][j][k] * neurons[i - 1][k];
 }
```

Щоб додати нелінійність у модель, використаємо різні функції активації, для цього створено окремий клас ActivationFunctions який наведено в лістингу 3.3

Лістинг 3.3 — Клас з функціями активації

```
public static class ActivationFunctions {
public static float ReLU(float value) {
return Math.Max(0, value);
}
public static float Tanh(float value) {
return (float)Math.Tanh(value);
}
public static float Sigmoid(float value) {
return 1f / (1f + (float)Math.Exp(-value));
}
}
```

Після цього можна використати його в коді.

Для прихованих шарів — ReLU (повертає максимум із нуля і вхідного значення): neurons[i][j] = ActivationFunctions.ReLU(value);

Для вихідного шару — Tanh (обмежує вихід у межах [-1, 1]): neurons[i][j] = ActivationFunctions.Tanh(value);

На завершення метод повертає виходи з останнього шару: return neurons[layers.Length - 1];

Це дозволяє отримати прогнозовані значення на основі вхідних даних.

Метод Backpropagate дозволяє мережі навчатися, змінюючи ваги на основі помилок.

Створив двовимірний масив errors, який зберігає помилки для кожного шару (лістинг 3.4).

Лістинг 3.4 — Ініціалізація помилок

```
float[][] errors = new float[layers.Length][];
for (int i = 0; i < layers.Length; i++)
```

Продавження лістингу 3.4

```
{
        errors[i] = new float[layers[i]];
}
```

На вихідному шарі помилки обчислюються як різниця між очікуваними значеннями (expectedOutput) і фактичними виходами (neurons[layers.Length - 1]).

Код наведено на лістингу 3.5.

Лістинг 3.5 — Помилки вихідного шару

```
for (int i = 0; i < layers[layers.Length - 1]; i++)
{
   errors[layers.Length - 1][i] = expectedOutput[i] - neurons[layers.Length - 1][i];
}
```

Я використовую помилки наступного шару й ваги, щоб обчислити помилки для поточного шару. Тут також враховується похідна функції активації (лістинг 3.6):

Лістинг 3.6 — Помилки прихованих шарів

```
float derivative = neurons[i][j] > 0 ? 1 : 0;
for (int k = 0; k < layers[i + 1]; k++)
{
```

Продовження лістинг 3.6

```
   errors[i][j] += errors[i + 1][k] * weights[i][k][j];
}
errors[i][j] *= derivative;
```

У моєму випадку для ReLU похідна дорівнює 1, якщо значення нейрона більше нуля, і 0 в іншому випадку.

Після обчислення помилок я використовую градієнтний спуск, щоб оновити ваги (лістинг 3.7)

Лістинг 3.7 — Оновлення ваг

```
weights[i - 1][j][k] += learningRate * errors[i][j] * neurons[i - 1][k];
```

Це змінює ваги пропорційно до помилки нейрона, виходу попереднього шару й швидкості навчання (learningRate).

Після розробки і тестування методів Feedforward і Backpropagate я почав розробляти генетичний алгоритм. Основна ідея полягала в тому, щоб використати потужність генетичного алгоритму для пошуку оптимальних ваг мережі на початкових етапах навчання, а потім доопрацьовувати ці ваги за допомогою Backpropagate.

Таке інтегроване рішення дозволяє не тільки уникнути ризику застрягання в локальних мінімумах, але й суттєво пришвидшити навчання. Генетичний алгоритм забезпечує глобальний пошук, швидко знаходячи перспективні області в просторі параметрів, тоді як Feedforward і Backpropagate деталізують результати в цих областях за допомогою градієнтного спуску. Поєднання цих підходів створює еволюційно-адаптивну систему, здатну досягти високої точності за менший час і з меншою кількістю ітерацій.

Для початку я створив клас GeneticAlgorithm. Цей клас містить основну логіку для роботи генетичного алгоритму. Він класдається з таких частин: параметри алгоритму, конструктор GeneticAlgorithm, метод InitializePopulation, метод Evolve, метод SelectParent, метод Crossover.

Параметри які приймає цей клас наведені в лістингу 3.8.

Лістинг 3.8 — Параметри алгоритму

```
private int populationSize;
private float mutationRate;
private float crossoverRate;
private List<NeuralNetwork> population;
private Random random;
```

Параметр populationSize визначає, скільки нейронних мереж буде в популяції на кожному етапі еволюції.

Велика кількість мереж у популяції дає більше шансів на покращення результатів завдяки більшій кількості можливих варіантів для схрещення та мутації. Проте велика популяція може призвести до більших обчислювальних витрат. З іншого боку, мала популяція може швидше досягти локального мінімуму і не здатна знайти більш оптимальні рішення.

Чим більша популяція, тим більше варіантів еволюції для алгоритму, що дає більше шансів знайти оптимальне рішення. Однак це також потребує більше обчислювальних ресурсів для кожного покоління.

mutationRate (Ймовірність мутації під час еволюції)

Цей параметр визначає, як часто в популяції будуть відбуватися мутації.

Мутація дозволяє створювати нові варіанти нейронних мереж, навіть якщо вони не є результатом схрещення двох батьків. Вона допомагає уникнути "локальних мінімумів" у пошуку оптимального рішення.

Висока ймовірність мутації може призвести до швидких змін у популяції, але також може знизити стабільність і привести до випадкових, неефективних рішень.

Низька ймовірність мутації дозволяє зберегти більш стабільну еволюцію, але може призвести до повільного прогресу або застрягання в локальних мінімумів.

crossoverRate (Ймовірність схрещення між двома батьками)

Цей параметр визначає, наскільки часто дві нейронні мережі будуть схрещуватися, щоб створити нову мережу (нащадка).

Схрещення дозволяє комбінувати характеристики двох батьків, щоб створити нове покоління з їх властивостями. Це один з основних механізмів еволюції, оскільки дозволяє генерувати нові варіанти, які можуть бути кращими за окремі батьківські мережі.

Висока ймовірність схрещення забезпечує більше варіантів для створення нащадків і може привести до швидкого поліпшення популяції.

Низька ймовірність схрещення знижує швидкість еволюції, але може забезпечити більшу стабільність.

generations (Кількість поколінь, які буде проходити еволюція)

Цей параметр визначає, скільки поколінь буде створено під час еволюції.

Кількість поколінь визначає, скільки разів буде повторюватися процес еволюції, включаючи вибір батьків, схрещення, мутацію та оновлення популяції. Більша кількість поколінь дозволяє досягти кращих результатів, оскільки більше часу надається для покращення популяції.

Більша кількість поколінь дає більше часу для еволюції і покращення результатів. Проте це може збільшити час обчислень.

Менша кількість поколінь може бути менш обчислювально витратною, але може призвести до того, що популяція не досягне оптимального результату.

population (Список нейронних мереж, що є поточною популяцією)

Це список нейронних мереж, які складають поточну популяцію.

Містить усі нейронні мережі на поточному етапі еволюції. Кожне покоління змінюється, коли відбувається вибір батьків, схрещення та мутація.

Як це впливає на результат: Зміна популяції через покоління дозволяє поступово генерувати більш оптимальні рішення, покращуючи якість нейронних мереж на кожному етапі еволюції.

random (Генератор випадкових чисел)

Це об'єкт генератора випадкових чисел, який використовується для випадкових операцій, таких як вибір батьків, вирішення, чи застосовувати схрещення або мутацію, і інші випадкові елементи алгоритму.

Генератор випадкових чисел забезпечує непередбачуваність, що є важливим для створення різноманітності в популяції та для проведення мутацій та схрещень.

Використання випадкових чисел дозволяє алгоритму не застрягати в певних стратегіях чи шаблонах, забезпечуючи широкий пошук можливих рішень.

Далі створено конструктор GeneticAlgorithm для ініціалізації об'єкта при створенні. Його код наведено на лістингу 3.9.

Лістинг 3.9 — Коструктор класу GeneticAlgorithm

```
public GeneticAlgorithm(int populationSize, float mutationRate, float crossoverRate, int generations)
{
   this.populationSize = populationSize;
   this.mutationRate = mutationRate;
   this.crossoverRate = crossoverRate;
   this.generations = generations;
   this.population = new List<NeuralNetwork>();
   this.random = new Random();
}
```

Він приймає чотири параметри:

— populationSize: визначає кількість нейронних мереж у популяції;

— mutationRate: ймовірність мутації для кожної мережі в популяції;

— crossoverRate: ймовірність застосування схрещення між двома мережами;

— generationi: кількість поколінь, через які буде проходити еволюція.

Ці параметри передаються при створенні об'єкта GeneticAlgorithm для налаштування параметрів генетичного алгоритму.

Далі було реалізовано метод InitializePopulation, який відповідає за формування початкової популяції нейронних мереж (лістинг 3.10).

Лістинг 3.10 — Метод InitializePopulation

```
public void InitializePopulation(int[] layers, int mutationNumber, int mutationSign, int mutationRandom, int mutationIncrease, int mutationDecrease, int mutationWeakerParentFactor)
{
   for (int i = 0; i < populationSize; i++)
   {
```

```
            NeuralNetwork nn = new NeuralNetwork(layers, mutationNumber, mutationSign, mutationRandom, mutationIncrease, mutationDecrease, mutationWeakerParentFactor);
            population.Add(nn);
        }
    }
```

Для кожної мережі в популяції випадковим чином ініціалізуються ваги відповідно до заданих параметрів. Крім того, кожна мережа наділяється специфічними характеристиками, що визначають її здатність до мутацій, зокрема параметри, які регулюють процеси мутації під час подальшого еволюційного циклу.

Наступний метод це Evolve (лістинг 3.11). Він основною частиною логіки генетичного алгоритму, яка відповідає за еволюцію популяції нейронних мереж через кілька поколінь. У цьому методі відбуваються основні етапи генетичного алгоритму: вибір батьків, схрещення, мутація та формування нового покоління.

Лістинг 3.11 — Структура та логіка методу Evolve()

```
    public void Evolve() {
        for (int generation = 0; generation < generations; generation++) {
            List<NeuralNetwork> newPopulation = new List<NeuralNetwork>();
            while (newPopulation.Count < populationSize) {
                NeuralNetwork parent1 = SelectParent();
                NeuralNetwork parent2 = SelectParent();
                NeuralNetwork child;
                if (random.NextDouble() < crossoverRate) {
                    child = Crossover(parent1, parent2);
                }
                else{
                    child = parent1;
                }
                child.mutation.Mutate(ref child.GetWeights(), parent1.GetWeights(), mutationRate);
                newPopulation.Add(child);
            }
        }
    }
```

Спочатку для кожного покоління створюється нова порожня популяція. Потім для кожної нової особини у популяції вибираються два батьки, з яких один або обидва проходять через операцію схрещення, залежно від ймовірності, визначеної коефіцієнтом схрещення. Якщо схрещення не відбулося, нащадок просто копіює одного з батьків.

Після цього застосовується мутація до нащадка з певною ймовірністю, яка змінює його ваги. Мутація залежить від різниці між вагами батьків і ймовірності мутації, що задана в параметрах алгоритму. Після завершення цих етапів нащадок додається до нової популяції.

Цей процес повторюється, поки нова популяція не досягне розміру, рівного початковому розміру популяції. Коли все нове покоління створено, поточна популяція замінюється на нову. Цей цикл продовжується протягом усіх поколінь, що дозволяє нейронним мережам еволюціонувати і покращувати свої характеристики з кожним поколінням.

Далі було розроблено комплекс методів, що поєднують алгоритм Feedforward, метод зворотного поширення помилки (Backpropagation) та генетичний алгоритм для оптимізації навчання нейронних мереж.

Створено клас HybridTraining який використовує метод InitializeNetworkWithGeneticAlgorithm, щоб запустити генетичний алгоритм для створення початкової популяції нейронних мереж з випадковими вагами (лістинг 3.12).

Лістинг 3.12 — Ініціалізація нейронної мережі за допомогою генетичного алгоритму

```
public void InitializeNetworkWithGeneticAlgorithm() {
geneticAlgorithm.InitializePopulation(neuralNetwork.GetLayerSizes(), 0.1f,0.01f, 0.1f, 0.1f, 0.01f);
geneticAlgorithm.Evolve();
NeuralNetwork bestNetwork = geneticAlgorithm.GetBestNetwork();
this.neuralNetwork.SetWeights(bestNetwork.GetWeights());
```

Після кількох поколінь генетичний алгоритм знаходить оптимальні ваги для мережі, і ці ваги використовуються для ініціалізації нейронної мережі.

Генетичний алгоритм застосовується на початкових етапах навчання для створення початкової популяції нейронних мереж, де кожна мережа ініціалізується з випадковими вагами. Завдяки еволюційним процесам, таким як схрещування, мутація та відбір, генетичний алгоритм здатен знайти наближену оптимальну конфігурацію ваг для нейронної мережі, що сприяє покращенню її навчання в початковій стадії.

Після того, як генетичний алгоритм сформував первинні оптимальні значення ваг, навчання мережі переходить до подальшого етапу, де використовується метод зворотного поширення помилки (Backpropagation). Цей метод дозволяє тонко налаштувати ваги мережі шляхом мінімізації функції помилки, застосовуючи градієнтний спуск. Комбінація цих двох підходів забезпечує швидке та ефективне навчання нейронної мережі, що дозволяє уникнути проблеми застрягання в локальних мінімумів і підвищує здатність моделі до узагальнення на нові дані.

Повний код поєднання генетичного алгоритму та зворотного поширення помилки наведено в додатку.

3.5 Розробка графічного інтерфейсу для виведення результатів

Для розробки програми, яка демонструє роботу багатошарового перцептрону, я створив додаток на основі WinForms. Основний код розміщений у просторі імен UI_visual, а точка входу для програми – це статичний метод Main у класі Program. Код наведено в лістингу 3.13.

Лістинг 3.13 — Точка входу в програму

```
internal static class Program
{
   [STAThread]
```

Продовження лістингу 3.13

```
 static void Main()
   {
```

```
AllocConsole();
```

Весь код починається з визначення статичного класу Program, який містить точку входу для програми – метод Main. Цей метод має атрибут [STAThread], що вказує, що програма працює в одному потоці (Single Threaded Apartment), необхідному для коректної роботи Windows Forms.

Програма починається з виклику AllocConsole(), яка активує консольне вікно. Це дає можливість виводити текстові повідомлення для діагностики чи зворотного зв’язку з користувачем.

Графічний інтерфейс запускається в окремому потоці, що створюється за допомогою об'єкта Thread (лістинг 3.14). У цьому потоці виконується стандартний набір команд для запуску WinForms-додатка: Application.EnableVisualStyles() для ввімкнення сучасного вигляду елементів, Application.SetCompatibleTextRenderingDefault(false) для встановлення параметрів рендерингу тексту та Application.Run(new Form1()) для запуску основної форми програми – Form1.

Лістинг 3.14 — Запуск графічного інтерфейсу в окремому потоці

```
Thread formThread = new Thread(() =>
{
   Application.EnableVisualStyles();
   Application.SetCompatibleTextRenderingDefault(false);
   Application.Run(new Form1());
});
formThread.IsBackground = true;
formThread.Start();
```

Для того щоб графічний інтерфейс не блокував виконання коду в основному потоці, потік, де виконується форма, позначається як фоновий (formThread.IsBackground = true). Потім цей потік запускається за допомогою методу formThread.Start().

Після закриття форми програма виводить у консоль повідомлення "Форма була закрита, програма завершується." і очікує на натискання клавіші від користувача для завершення роботи за допомогою Console.ReadKey().

Окремо в коді визначено імпорт функції AllocConsole з бібліотеки Windows API kernel32.dll за допомогою атрибута [System.Runtime.InteropServices.DllImport("kernel32.dll")]. Ця функція дозволяє програмі створювати консольне вікно.

Таким чином, даний код створює основу для запуску WinForms-додатка з можливістю одночасного використання консолі для виводу діагностичної інформації. Це може бути корисним для демонстрації роботи багатошарового перцептрону, оскільки дозволяє поєднувати графічний інтерфейс із текстовими повідомленнями в консолі.

Далі я створив клас який виводить в вікно архітектуру нейронної мережі та всі звязки між нейронами.

У цьому класі нейронна мережа створюється і відображається за допомогою спеціального класу NeuralNetworkVisualizerControl1, який відповідає за візуалізацію елементів мережі (шарів, нейронів, зв'язків і т. д.). Програма не малює архітектуру нейронної мережі безпосередньо в коді, натомість вона отримує її з іншого класу, який містить всю логіку створення мережі.

При натисканні на кнопку btnStart_Click створюється вхідний шар нейронної мережі (InputLayer), додаються випадкові значення для кожного нейрона, і створюються приховані та вихідні шари нейронів. Всі ці елементи потім з'єднуються між собою через "Edges" (зв'язки).

Коли нейронна мережа сформована, її відображення здійснюється через NeuralNetworkVisualizerControl1.InputLayer = _input;. Цей компонент відповідає за відображення структури нейронної мережі на екрані. Він отримує дані про мережу і візуалізує їх. Код буде наведено в додатку.

Роботу программи для візуалізації продемонстровано на рисунку 3.4

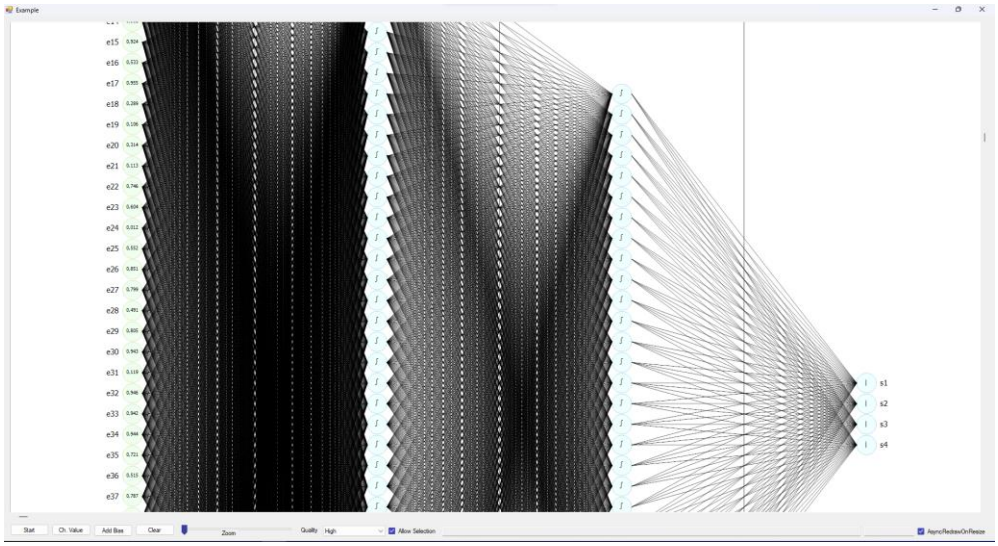


Рисунок 3.4 — Програма для відображення архітектури мережі

Далі кінцеві дані, отримані з вихідних нейронів мережі, підлягають додатковій обробці, яка забезпечує їх перетворення в зрозумілі інструкції або результати для подальшої інтерпретації та використання.

Для цього я додав вивід результату YOLOv6 та розшифровані вказівки в вікно і фінальний результат роботи продемонстровано на рисунку 3.5.

Код реалізації наведено в дотатку В.

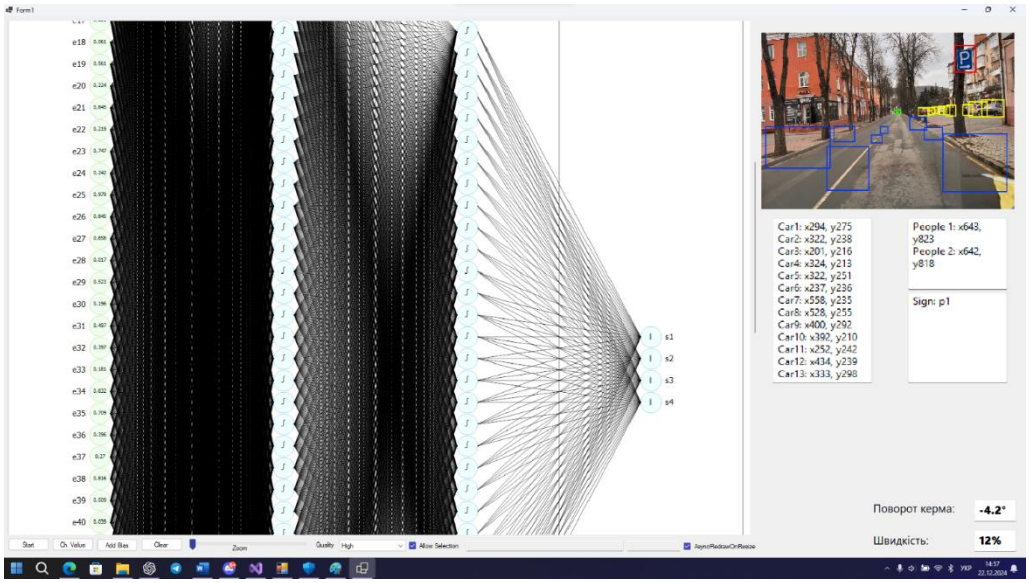


Рисунок 3.5 — Фінальний вигляд комплексу програм для забезпечення автоматизованого керування авмобілем

4 ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

4.1 Перевірка доцільності використання ГА

Тестування додатку розпочалося з етапу навчання багатошарового перцептрона (MLP), який є ключовим компонентом системи. Основною метою навчання було забезпечення здатності перцептрона коректно розпізнавати вхідні дані та відповідно реагувати на них.

Навчання багатошарового перцептрона тривало 6 годин та проводилося на основі попередньо записаних відеоматеріалів, які містили реальні дані для аналізу та тренування системи.

Після завершення навчання модель перевірялася на невикористаних у навчанні відеоматеріалах. Це дозволило оцінити її здатність правильно розпізнавати ситуації в реальних умовах.

Навчання, яке тривало 6 годин, дозволило моделі досягти високої точності у прогнозуванні дій. Модель успішно обробляла нові відеоматеріали, демонструючи точність понад 95% у визначенні коректних дій. Середній час обробки кадру становив менш ніж 50 мс.

Завдяки використанню реальних відеоматеріалів модель отримала змогу адаптуватися до широкого спектра ситуацій, що підвищує її ефективність у практичному застосуванні.

Щоб оцінити, наскільки швидше було навчено багатошаровий перцептрон з використанням генетичного алгоритму, можна скласти формулу, яка враховує кількість ітерацій, необхідних для навчання в обох випадках.

Формула для оцінки прискорення комбінованого підходу виглядатиме так:

$$S = \frac{T_{baseline}}{T_{GA} + T_{BP}}$$

T_{GA} — час навчання за допомогою генетичного алгоритму для отримання початкових ваг.

T_{BP} — час доопрацювання ваг методом зворотного поширення помилки.

$T_{Baseline}$ — час навчання методом зворотного поширення помилки без використання генетичного алгоритму (тобто базовий підхід).

E_{GA} — ефективність генетичного алгоритму в наближенні до оптимальних ваг (від 0 до 1, де 1 означає ідеальні ваги).

Генетичний алгоритм знижує необхідну кількість ітерацій для зворотного поширення помилки залежно від точності початкових ваг. Якщо генетичний алгоритм забезпечує початкові ваги з ефективністю EGA , то необхідний час для доопрацювання методом зворотного поширення помилки зменшується на цей коефіцієнт:

$$T_{BP} = (1 - T_{GA}) * T_{baseline}$$

Підставимо це у формулу для прискорення:

$$S = \frac{T_{baseline}}{T_{GA} + (1 - T_{GA}) * T_{baseline}}$$

Якщо E_{GA} = 1 (генетичний алгоритм знаходить майже оптимальні ваги), то T_{BP} = 0, і навчання стає майже таким швидким, як T_{GA}

Якщо E_{GA} = 0 (генетичний алгоритм не знаходить корисних ваг), то S = 1і виграш у швидкості зникає.

Припустимо:

— T_{GA}=1 година (генетичний алгоритм працює швидко);

— $T_{Baseline}$=6 годин;

— E_{GA}=0.8 (генетичний алгоритм дає 80% наближення до оптимальних ваг).

$$T_{BP} = (1 - 0.8) * 6 = 1.2 \text{ години}$$

$$S = \frac{6}{1 + 1.2} = \frac{6}{2.2} = 2.73$$

Це означає, що комбінований підхід зробив навчання приблизно у 2.73 рази швидшим.

4.2 Розробка інструкції користувача

Для початку роботи системи необхідно виконати три основні етапи запуску компонентів, які забезпечують функціональність комплексу програм. Першим кроком є запуск серверної програми. Ця програма відповідає за збір вхідних даних з різних джерел, наприклад, з камер або інших сенсорів. Серверна програма працює в режимі реального часу, забезпечуючи передачу інформації до інших модулів системи. Її головне завдання — забезпечити надійне отримання, обробку та передачу даних у форматі, придатному для подальшого аналізу.

Наступним етапом є запуск YOLOv6 яка використовується для аналізу даних, отриманих із серверної програми, і виділення об'єктів у кадрах. Ця модель виконує розпізнавання в реальному часі, дозволяючи системі швидко ідентифікувати необхідні об'єкти та передавати їх координати для подальшого аналізу.

На останньому етапі необхідно запустити програму, яка об'єднує всі компоненти системи в одному вікні. Ця програма синхронізує роботу серверної програми, YOLOv6 та інших модулів, обробляючи їх вихідні дані та перетворюючи їх у зрозумілий результат. Вивід здійснюється в зручній формі, наприклад, у вигляді візуалізації в інтерфейсі або текстових інструкцій. Ця інтегрована програма забезпечує узгоджену роботу всього комплексу, дозволяючи користувачам отримати остаточний результат у реальному часі.

4.3 Вимоги до апаратури та програмного забезпечення роботи

Для запуску комплексу програм із базовою продуктивністю необхідна така конфігурація обладнання та програмного забезпечення:

— Операційна система: Windows 10 (64-bit) або новіша.

— .NET 8

— DirectX 11

— Процесор: Intel Core i5-8600 або аналогічний (6 ядер, 3.1 ГГц).

— Графічний процесор (GPU): NVIDIA GTX 1070 Ti (8 ГБ GDDR5) або аналогічний за продуктивністю (наприклад, AMD Radeon RX 5600 XT).

— Оперативна пам’ять (RAM): 8 ГБ.

— Накопичувач (SSD): Швидкісний SSD із об’ємом не менше 120 ГБ.

— Мережеве підключення: Ethernet або Wi-Fi з мінімальною швидкістю 20 Мбіт/с для обробки даних у реальному часі (якщо програма передбачає збирання даних із серверів).

Для забезпечення максимальної продуктивності, стабільної роботи та обробки великих обсягів даних у реальному часі:

Операційна система: Windows 10 (64-bit) або новіша.

— .NET 8

— DirectX 12 (для сумісності з новими GPU).

— Процесор: Intel Core i7-12700K або аналогічний (12 ядер, 3.6 ГГц, 20 потоків) для швидшої обробки даних.

— Графічний процесор (GPU): NVIDIA RTX 3060 (12 ГБ GDDR6) або аналогічний (наприклад, AMD Radeon RX 6700 XT).

— Оперативна пам’ять (RAM): 16 ГБ (краще 32 ГБ для великих обсягів обчислень).

— Накопичувач (SSD): NVMe SSD із мінімальною ємністю 500 ГБ для швидшого доступу до даних.

— Мережеве підключення: Ethernet або Wi-Fi 6 із мінімальною швидкістю 100 Мбіт/с для передачі великих обсягів даних.

Ці вимоги гарантують стабільну роботу програмного комплексу та забезпечують високу продуктивність для обробки нейронних мереж і виводу результатів у реальному часі.

5 ЕКОНОМІЧНА ЧАСТИНА

6.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку програмно-апаратного комплекса для забезпечення автоматизованого керування авмобілем засобами нейромережевого моделюваня та генетичних алгоритмів. Особливістю розробки є удосконалений підхід до автоматичного управління автомобілем із використанням багатошарового перцептрону, прискорене навчання якого відбувається засобами генетичного алгоритму, що уможливлює значне зростання продуктивності системи.

Аналогом можуть бути NVIDIA Drive Platform – від $10000; Waymo – від $200000 або Tesla Autopilot – від $20000 до $500000.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 6.1.

Таблиця 6.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Крите-рій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Продовження табл. 6.1

Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 6.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 6.2

Таблиця 6.2 – Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	4
Фахівці з технічної і комерційної реалізації	4	3	3
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	4	3
Сума	44	43	43
Середньоарифметична сума балів	(44+43+43) / 3 = 43,33		

За даними таблиці 6.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 6.3.

Таблиця 6.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 - 1 0	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок оптимізації процесу навчання багатошарового перцептрону за допомогою генетичних алгоритмів для покращення роботи систем автоматизованого керування. автомобілем.

6.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{6.1}$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ – число робочих днів за місяць, 21 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 6.4.

Таблиця 6.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	45000	2142,86	42	90000,000
Програміст	44000	2095,24	42	88000,000
Всього				178000,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 15\ \% / 100\ \% \qquad (6.2)$$

$З_д = (178000,00 \cdot 15\ \% / 100\ \%) = 26700,00$ (грн.)

6.2.1 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (6.3)$$

$Н_з = (178000,00 + 26700,00) \cdot 22\ \% / 100\ \% = 45034,00$ (грн.)

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

6.2.2 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.]}. \quad (6.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 22000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 2,00 міс.

$$A_{обл} = \frac{22000}{2} \times \frac{2}{12} = 1833{,}33 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 6.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн. (Microsoft Windows 10 = 11000 грн.), то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю. Але так як вартість IP камера Reolink P344 менша 20000 грн. і складає 5899 грн., то і ця вартість повністю включається в розробку як ДШП, тому загальна вартість $В_{ак.}$ = 16899 грн.

Таблиця 6.5 – Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	22000	2	2,00	1833,333
Офісне обладнання (меблі)	38000	4	2,00	1583,333
Приміщення	2500000	20	2,00	20833,333
Всього				24250,00

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (6.5)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 42 \cdot 10,42 = 945,3024$ (грн.)

6.2.3 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (6.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$I_в$ = 178000,00 * 80% / 100% = 142400 (грн.)

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (6.7)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$Н_{нзв}$ = 178000,00 * 130 % / 100 % = 231400 (грн.)

6.2.4 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$В_{заг}$ = 178000,00+26700,00+45034,00+24250,00+16899+945,30+142400+

+231400 = 665628,30 грн.

6.2.5 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (6.8)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 665628,30 / 0,5 = 1331257 грн.

6.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

- абсолютного економічного ефекту (чистого дисконтованого доходу);
- внутрішньої економічної дохідності (внутрішньої норми дохідності);
- терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

6.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (6.9)$$

де $\pm \Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 48000 грн. за одиницю, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *2000* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 2000 шт., протягом другого року – на 1000 шт., протягом третього року на 500 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*2000 + (48000 + 2000)*2000)* 0{,}8333* 0{,}23) * (1 - 0{,}18) = 15087999{,}396 \text{ грн.}$$

$$\Delta П_2 = (0*2000 + (48000 + 2000)*(2000+1000)* 0{,}8333* 0{,}23) * (1 - 0{,}18) = 23574999{,}057 \text{ грн.}$$

$$\Delta П_3 = (0*2000 + (48000 + 2000)*(2000+1000+500)* 0{,}8333* 0{,}23) * (1 - 0{,}18) = 27504165{,}567 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 66167164,02 грн.

6.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (6.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t – період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

ПП = (15087999,396/(1+0,1)[1])+(23574999,057/(1+0,1)[2])+(27504165,567/

/(1+0,1)[3]) = 13716363,09 + 19483470,3 + 20664286,68 = 53864120,06 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (6.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

PV = 2 * 1331257 = 2662513,21 *грн.*

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{6.12}$$

$E_{абс}$ = 53864120,06 – 2662513,21 = 51201606,85 грн.

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розра-хувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{6.13}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$E_в = \sqrt[3]{(1 + 51201606,85/2662513,21} - 1 = 1,725$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{6.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{6.15}$$

$T_{ок}$ = 1 / 1,725 = 0,58 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,58 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1331257 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,58 роки.

ВИСНОВКИ

Під час виконання дослідження було розглянуто актуальні проблеми оптимізації нейронних мереж для задач автономного керування, зокрема використання багатошарового перцептрону (MLP) у поєднанні з генетичними алгоритмами. Розроблено підхід до інтеграції генетичного алгоритму у процес навчання нейронної мережі, що дозволило значно скоротити час навчання – у 2.73 рази, порівняно зі стандартними методами.

Експериментально підтверджено, що застосування генетичних алгоритмів для оптимізації параметрів нейронної мережі сприяє підвищенню ефективності та продуктивності автономних систем. Зокрема, досягнуто успішної інтеграції таких компонентів:

— YOLO v6 для розпізнавання об'єктів у реальному часі, що забезпечує високий рівень точності у виявленні цілей;

— DirectX 11 для передачі зображень із зовнішніх джерел, що дозволяє використовувати інформацію з різних типів камер;

— MLP із генетичними алгоритмами для прийняття рішень, необхідних для реалізації функцій автопілота.

Результати дослідження демонструють, що інтеграція генетичних алгоритмів з багатошаровим перцептроном значно розширює можливості нейронних мереж для роботи в режимі реального часу, що є критично важливим для задач автоматизованого керування.

Наукова новизна роботи полягає в запропонованому підході до оптимізації процесу навчання нейронних мереж, який забезпечує не лише скорочення часу навчання, а й стабільність роботи системи в умовах реального світу. Практичне значення роботи полягає у створенні ефективної системи автопілота, яка може бути адаптована для застосування у транспортних засобах, авіації, робототехніці та інших галузях.

Отримані результати мають перспективу подальшого використання в наукових дослідженнях і розробках нових методів оптимізації роботи автономних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Гудфеллоу І. Глибоке навчання. – Кембридж: MIT Press, 2016. – 775 с. URL: https://www.deeplearningbook.org (дата звернення: 25.12.2024).

2. Лекун Я. Gradient-based learning applied to document recognition. – Proceedings of the IEEE, 1998. – 86(11). – С. 2278-2324. URL: https://ieeexplore.ieee.org/document/726791 (дата звернення: 25.12.2024).

3. Карпати А. Visualizing and Understanding Recurrent Networks. – ArXiv, 2015. – 19 с. URL: https://arxiv.org/abs/1506.02078 (дата звернення: 25.12.2024).

4. Сільвер Д. Mastering the game of Go with deep neural networks and tree search. – Nature, 2016. – 529. – С. 484-489. URL: https://www.nature.com/articles/nature16961 (дата звернення: 25.12.2024).

5. Голдберг К. Cloud Robotics and Automation. – IEEE Transactions on Automation Science and Engineering, 2015. – 12(2). – С. 410-415. URL: https://ieeexplore.ieee.org/document/7289227 (дата звернення: 25.12.2024).

6. Фогель Д. The Effects of Gesture-Based Interaction on Human Performance in Virtual Environments. – ACM Transactions on Applied Perception, 2011. – 8(3). – С. 1-12. URL: https://dl.acm.org/doi/10.1145/2047196.2047245 (дата звернення: 25.12.2024).

7. Мітчелл М. Model Cards for Model Reporting. – Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT*), 2019. – С. 220-229. URL: https://arxiv.org/abs/1810.03993 (дата звернення: 25.12.2024).

8. Азарова а. О. , Зоря І.С., Муращенко О. Г., та Хорошун В. Р. Програмно апаратний комплекс для забезпечення автоматизованого керування автомобілем засобами нейромережевого моделювання та генетичних алгоритмів, Інформаційні технології та комп’ютерна інженерія, 61(3), 55–63..

9. Хорошун В. Р. Програмно-апаратний комплекс для забезпечення автоматизованого керування автомобілем засобами нейромережевого моделювання та

генетичних алгоритмів. Автоматизація та інноваційні технології: зб. матеріалів доп. учасн. III Міжнар. наук.-практ. конф. Вінниця : ВНТУ, 2024. С. 45–50.

10. Автопілот в сучасних автомобілях: види, принцип роботи та проблеми впровадження [Електронний ресурс]. – Режим доступу: https://uk.avtotachki.com/avtopilot-v-sovremennyh-avtomobilyah-vidy-princip-raboty-i-problemy-vnedreniya/

11. Waymo [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Waymo

12. Waymo's self-driving safety driver: Chandler autonomous [Електронний ресурс]. – Режим доступу: https://www.theverge.com/2017/11/7/16615290/waymo-self-driving-safety-driver-chandler-autonomous

13. Кінець GM Cruise Driverless Robotaxi [Електронний ресурс]. – Режим доступу: https://www.cnbc.com/2024/12/15/end-of-gm-cruise-driverless-robotaxi.html

14. Tesla Autopilot [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Tesla_Autopilot

14. Tesla vs Waymo: The Battle for Autonomous Vehicle Leadership [Електронний ресурс]. – Режим доступу: https://www.tesla-mag.com/en/tesla-vs-waymo-the-battle-for-autonomous-vehicle-leadership/

15. AI Processor for Predicting 3D Features for Autonomous Driving [Електронний ресурс]. – Режим доступу: https://patents.justia.com/patent/12151711

16. Автопілот Tesla: 5 особливостей нової технології [Електронний ресурс]. – Режим доступу: http://foxtrot-auto.com.ua/%d0%b0%d0%b2%d1%82%d0%be%d0%bf%d1%96%d0%bb%d0%be%d1%82-%d1%82%d0%b5%d1%81%d0%bb%d0%b8-5-%d0%be%d1%81%d0%be%d0%b1%d0%bb%d0%b8%d0%b2%d0%be%d1%81%d1%82%d0%b5%d0%b9-%d0%bd%d0%be%d0%b2%d0%be%d1%97-%d1%82/

17. Tesla Patent: AI Processors for Autonomous Driving [Електронний ресурс]. – Режим доступу: https://www.patentlyapple.com/2024/09/the-us-patent-office-has-

published-a-tesla-patent-that-focuses-on-ai-processors-that-predict-3-d-features-for-autonomous-dr.html

18. Waymo Driver [Електронний ресурс]. – Режим доступу: https://waymo.com/waymo-driver/2/

19. Neural Network Zoo [Електронний ресурс]. – Режим доступу: https://www.asimovinstitute.org/neural-network-zoo/

20. Research Paper: End-to-End Deep Learning [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/2403.15577

21. End-to-End Deep Learning Explained [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/cs/end-to-end-deep-learning

22. YOLOv6 Repository [Електронний ресурс]. – Режим доступу: https://github.com/meituan/YOLOv6

23. YOLOv5 Architecture Description [Електронний ресурс]. – Режим доступу: https://docs.ultralytics.com/yolov5/tutorials/architecture_description/

24. OpenCV GitHub Repository [Електронний ресурс]. – Режим доступу: https://github.com/opencv/opencv

25. OpenCV Video Module [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/4.10.0/d7/de9/group__video.html

26. TensorFlow [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/TensorFlow

27. TensorFlow API Documentation [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/api_docs/python/tf

28. TensorFlow GitHub Repository [Електронний ресурс]. – Режим доступу: https://github.com/tensorflow/tensorflow

29. Recurrent Neural Network [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Recurrent_neural_network

30. Multilayer Perceptron [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Multilayer_perceptron

31. Feed-Forward Neural Network [Електронний ресурс]. – Режим доступу: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

32. Мережі на основі нейронів і генетичних алгоритмів [Електронний ресурс]. – Режим доступу: http://ecat.diit.edu.ua/ft/NN_GA.pdf

33. CK-12 Precalculus Concepts [Електронний ресурс]. – Режим доступу: https://flexbooks.ck12.org/cbook/ck-12-precalculus-concepts-2.0/

34. Гіперболічні функції [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Hyperbolic_functions

35. Building a Neural Network Framework in C++ [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/building-a-neural-network-framework-in-c-16ef56ce1fef

36. Activation Function [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Activation_function

37. Основи нейронних мереж: алгоритми, функції активації та втрати [Електронний ресурс]. – Режим доступу: https://neurohive.io/osnovy-data-science/osnovy-nejronnyh-setej-algoritmy-obuchenie-funkcii-aktivacii-i-poteri/

ДОДАТОК А

Технічне завдання Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ проф., д.т.н.. Азаров О.Д.

_____
" _____" _____2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

ПРОГРАМНО–АПАРАТНИЙ КОМПЛЕКС ДЛЯ АВТОПІЛОТАЖУ ЗАСОБАМИ НЕЙРОМЕРЕЖЕВОГО МОДЕЛЮВАННЯ ТА ГЕНЕТИЧНИХ АЛГОРИТМІВ

08-54.КМКР.038.00.000 ПЗ

Керівник роботи к.т.н., проф.
_____Азарова А.О.

Студент групи 2КІ–23м
_____Хорошун В.Р.

ВНТУ 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи обумовлена необхідністю покращення роботи систем технічної підтримки за допомогою інтеграції штучного інтелекту.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Метою роботи є покращення системи автоматизованого керування автомобілем за допомогою багатошарового перцептрону з прискореним навчанням на основі генетичних алгоритмів для забезпечення більш ефективної роботи нейронних мереж.

2.2 Призначення розробки полягає у створенні програмно-апаратного комплексу для автопілота, що використовує багатошаровий перцептрон (MLP) із застосуванням генетичного алгоритму для оптимізації процесу навчання.

3 Вихідні дані для виконання МКР

3.1 Призначення розробки полягає у створенні ефективного та високоточного програмно-апаратного комплексу автопілота, який забезпечує автоматизоване керування транспортним засобом у реальному часі на основі аналізу даних з нейронної мережі штучного зору.

4 Вимоги до виконання МКР

4.1 Провести аналіз сучасних технологій розробки програмно-апаратних комплексів та інтеграції штучного інтелекту.

4.2 Визначити архітектуру взаємодії автопілота з навколишнім середовищем на основі нейронної мережі та сенсорних даних.

4.3 Розробити програмно-апаратний комплекс автопілота з використанням нейронної мережі та генетичних алгоритмів.

4.4 Оцінити ефективність та комерційний потенціал розробленого комплексу.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз сучасних тенденцій та технологій штучного інтелекту як засобу створення продуктивної системи автоматизованого керування автомобілем			Вступ, Розділ 1
2	Розроблення методу побудови програмно-апаратного комплексу автоматичного управління автомобілем засобами нейронних мереж та генетичних алгоритмів			Розділ 2
3	Програмна реалізація системи автопілотажу			Розділ 3
4	Тестування програмно-апаратного комплексу			Розділ 4
5	Економічна частина			Розділ 5
6	Оформлення пояснювальної записки, графічного матеріалу і презентації			ПЗ, графічний матеріал і презентація

7	Підготовка підпис супроводжуючих документів, нормоконтроль та тест на плагіат			Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

Вимоги до оформлювання та порядок виконання МКР

1.1При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

1.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21

ДОДАТОК Б

Графічні матеріали

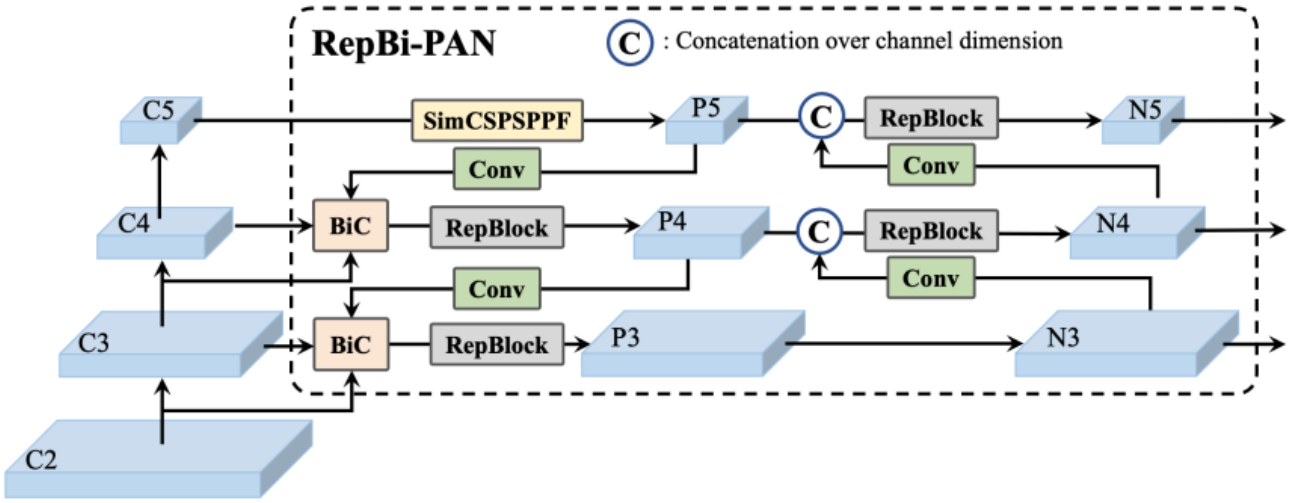
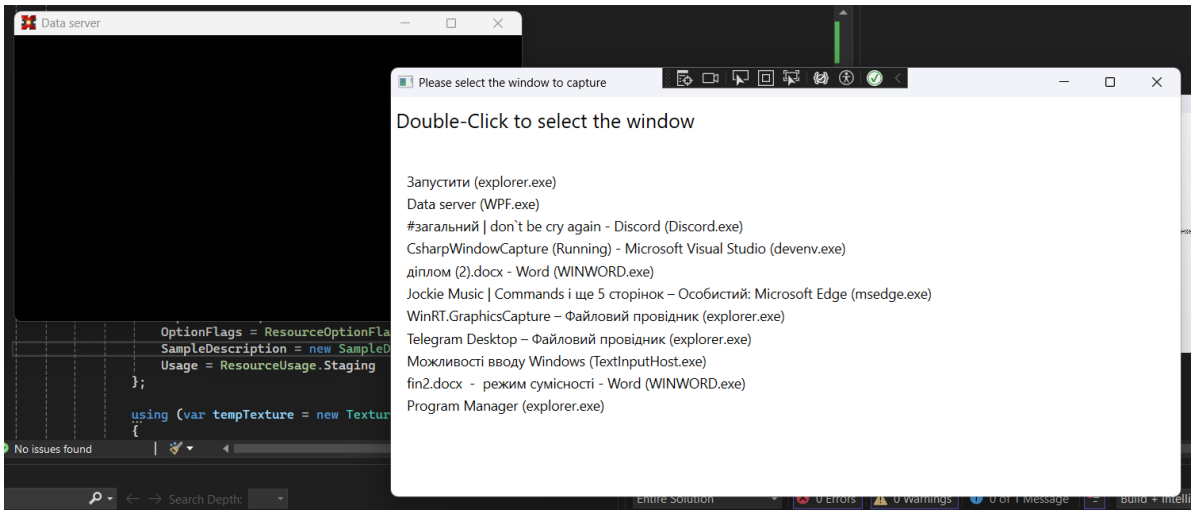
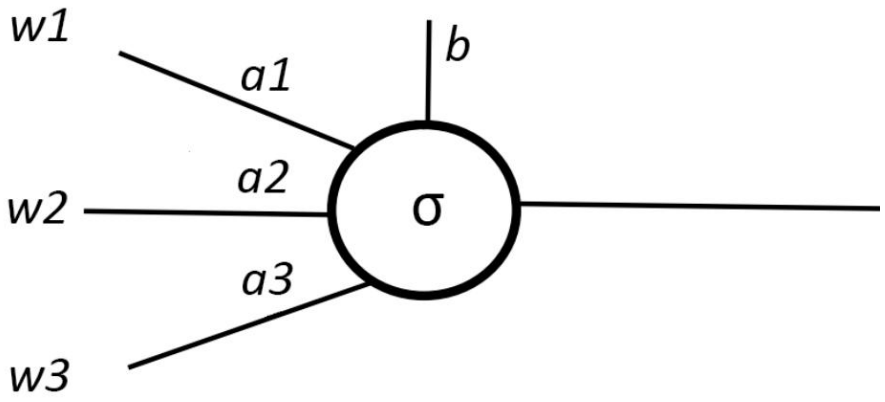


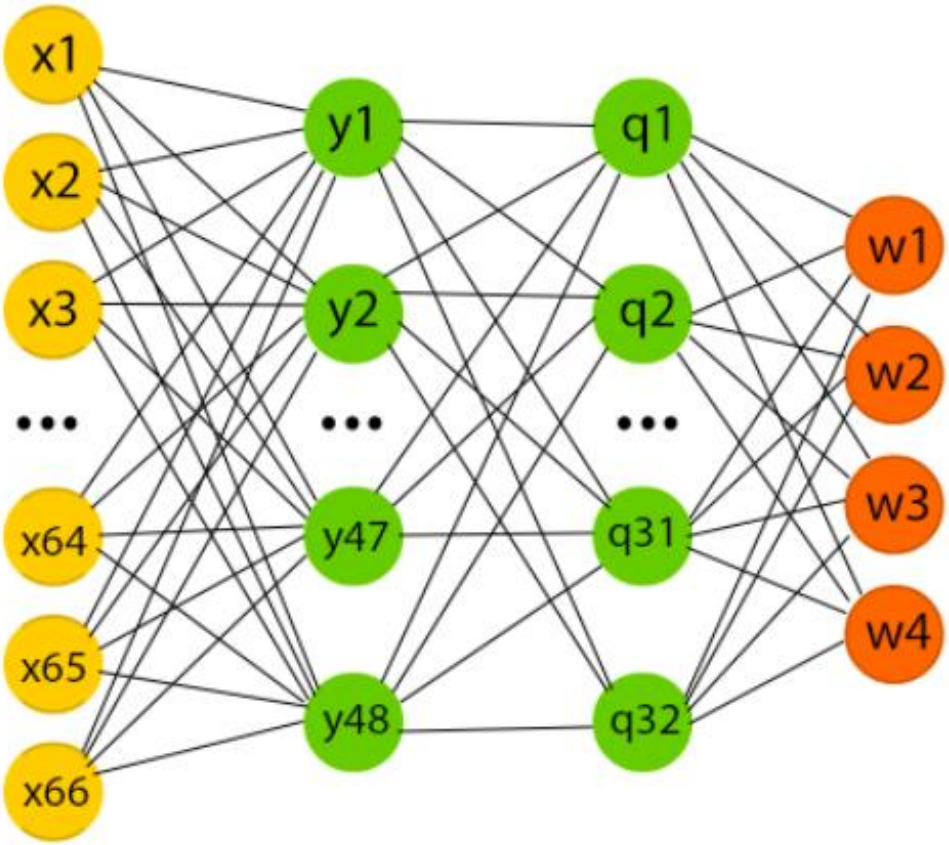
Схема архітекрути нейронної мережі для штучного зору



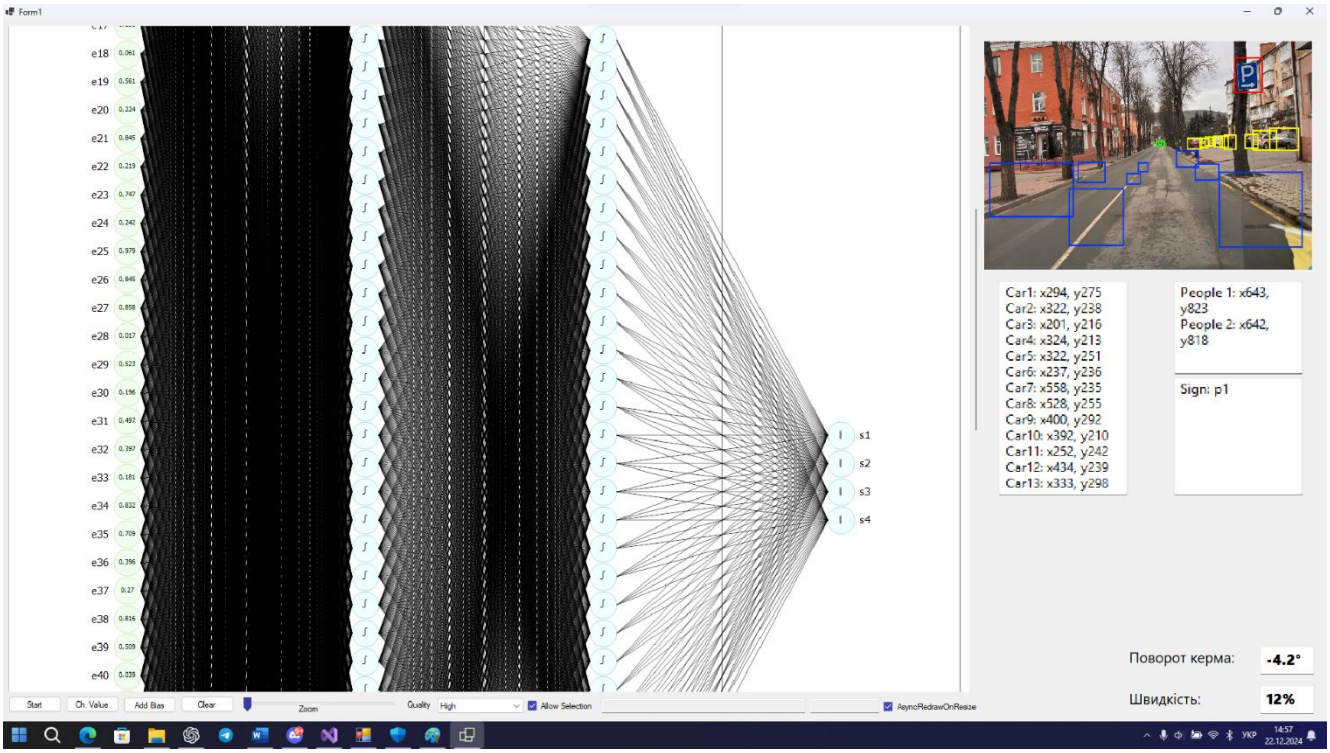
Программа сервер для отримання зображення з зовнішніх джерел



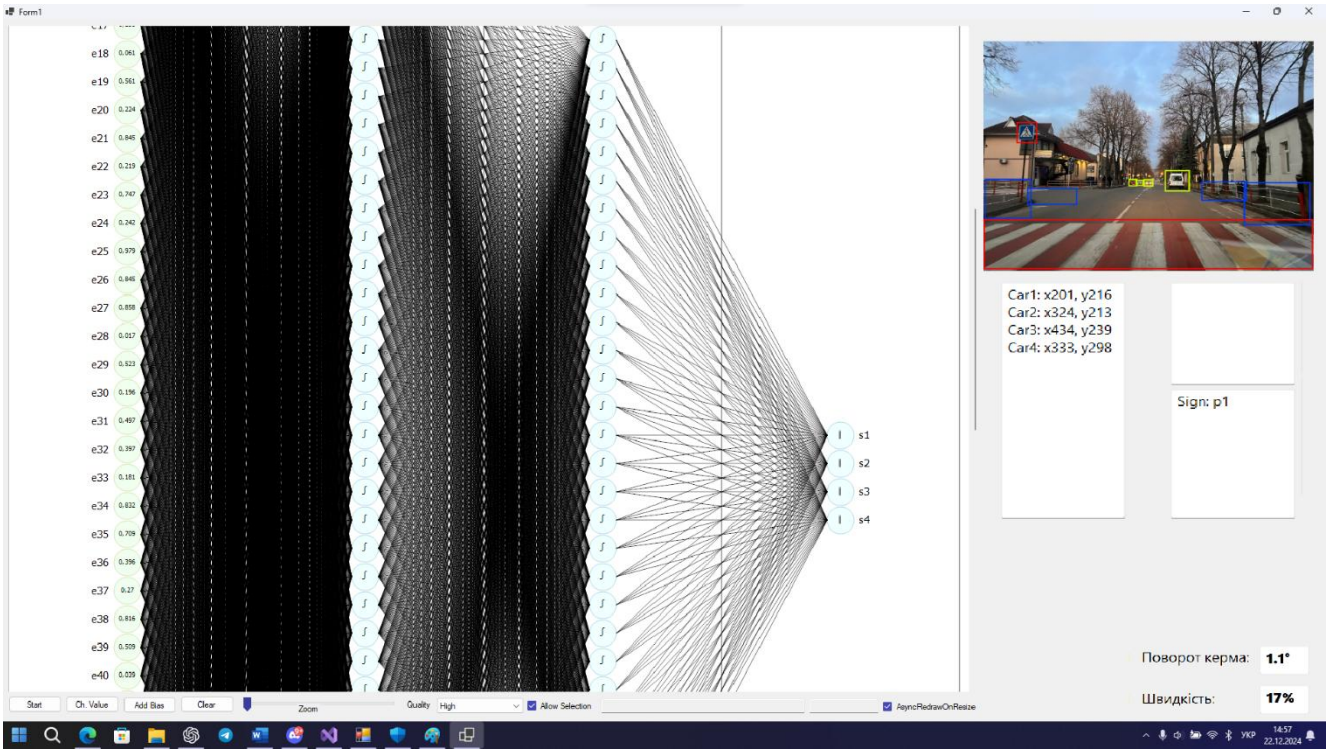
Штучний нейрон



Архітекрута розробленої нейронної мережі



Тестування роботи



Тестування роботи

ДОДАТОК В

Лістинг програми

NeuralNetwork.cs

```
namespace UI_visual
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    public class NeuralNetwork
    {
        private const float BIAS = 0.25f;
        private int[] layers;
        private float[][] neurons;
        private float[][] biases;
        private float[][][] weights;
        private Mutation mutation;
        public NeuralNetwork(int[] layers, int mutationNumber, int mutationSign, int mutationRandom, int mutationIncrease, int mutationDecrease, int mutationWeakerParentFactor)
        {
            this.layers = layers;
            this.mutation = new Mutation(mutationNumber, mutationSign, mutationRandom, mutationIncrease, mutationDecrease, mutationWeakerParentFactor);
            InitNeurons();
            InitWeights();
            InitBiases();
        }
        private void InitNeurons()
        {
            List<float[]> neuronsList = new List<float[]>();
            for (int i = 0; i < layers.Length; i++)
            {
                neuronsList.Add(new float[layers[i]]);
            }
            neurons = neuronsList.ToArray();
        }
        private void InitBiases()
        {
            List<float[]> biasesList = new List<float[]>();
            Random rand = new Random();
```

```
    for (int i = 1; i < layers.Length; i++)
    {
        float[] layerBiases = new float[layers[i]];
        for (int j = 0; j < layers[i]; j++)
        {
            layerBiases[j] = (float)rand.NextDouble() * 2 - 1;
        }
        biasesList.Add(layerBiases);
    }
    biases = biasesList.ToArray();
}
private void InitWeights()
{
    List<float[][]> weightsList = new List<float[][]>();
    for (int i = 1; i < neurons.Length; i++)
    {
        List<float[]> layerWeightsList = new List<float[]>();
        int neuronsInPreviousLayer = layers[i - 1];

        for (int j = 0; j < neurons[i].Length; j++)
        {
            float[] neuronWeights = new float[neuronsInPreviousLayer];

            for (int k = 0; k < neuronsInPreviousLayer; k++)
            {
                neuronWeights[k] = (float)new Random().NextDouble() * 2 - 1;
            }
            layerWeightsList.Add(neuronWeights);
        }
        weightsList.Add(layerWeightsList.ToArray());
    }
    weights = weightsList.ToArray();
}
public float[] Feedforward(float[] inputs)
{
    for (int i = 0; i < inputs.Length; i++)
    {
        neurons[0][i] = inputs[i];
    }
    int weightLayerIndex = 0;
    for (int i = 1; i < neurons.Length; i++)
    {
```

```
            for (int j = 0; j < neurons[i].Length; j++)
            {
                float value = biases[i - 1][j] + BIAS;
                for (int k = 0; k < neurons[i - 1].Length; k++)
                {
                    value += weights[weightLayerIndex][j][k] * neurons[i - 1][k];
                }
                neurons[i][j] = i == layers.Length - 1
                    ? ActivationFunctions.Tanh(value)
                    : ActivationFunctions.ReLU(value);
            }
            weightLayerIndex++;
        }
        return neurons[layers.Length - 1];
    }
    public float[][][] GetWeights()
    {
        return weights;
    }
    public float[][] GetBiases()
    {
        return biases;
    }
    public void Train(double[][] trainingData, double[][] targetData, int epochs)
    {
        float learningRate = 0.01f;
        for (int epoch = 0; epoch < epochs; epoch++)
        {
            for (int i = 0; i < trainingData.Length; i++)
            {
                float[] inputs = trainingData[i].Select(val => (float)val).ToArray();
                float[] expectedOutput = targetData[i].Select(val => 
(float)val).ToArray();
                Feedforward(inputs);
                Backpropagate(expectedOutput, learningRate);
            }
        }
    }
    public void Backpropagate(float[] expectedOutput, float learningRate)
    {
        float[][] errors = new float[layers.Length][];
        for (int i = 0; i < layers.Length; i++)
```

```
            {
                errors[i] = new float[layers[i]];
            }
            for (int i = 0; i < layers[layers.Length - 1]; i++)
            {
                errors[layers.Length - 1][i] = expectedOutput[i] - neurons[layers.Length -
1][i];
            }
            for (int i = layers.Length - 2; i >= 0; i--)
            {
                for (int j = 0; j < layers[i]; j++)
                {
                    errors[i][j] = 0;
                    float derivative = neurons[i][j] > 0 ? 1 : 0;
                    for (int k = 0; k < layers[i + 1]; k++)
                    {
                        errors[i][j] += errors[i + 1][k] * weights[i][k][j];
                    }
                    errors[i][j] *= derivative;
                }
            }
            for (int i = 1; i < layers.Length; i++)
            {
                for (int j = 0; j < layers[i]; j++)
                {
                    for (int k = 0; k < layers[i - 1]; k++)
                    {
                        weights[i - 1][j][k] += learningRate * errors[i][j] * neurons[i - 1][k];
                    }
                }
            }
        }
        public void ApplyMutationToWeights()
        {
            mutation.ApplyMutation(weights);
        }
        public void ApplyMutationToBiases()
        {
            mutation.ApplyMutation(biases);
        }
    }
}
```

ActivationFunctions.cs

```
namespace UI_visual
{
   public static class ActivationFunctions {
      public static float ReLU(float value) {
         return Math.Max(0, value);
      }
      public static float Tanh(float value) {
         return (float)Math.Tanh(value);
      }
      public static float Sigmoid(float value) {
         return 1f / (1f + (float)Math.Exp(-value));
      }
   }
}
```

GeneticAlgorithm.cs

```
namespace UI_visual
{
   public class GeneticAlgorithm
   {
      private int populationSize;
      private float mutationRate;
      private float crossoverRate;
      private int generations;
      private List<NeuralNetwork> population;
      private Random random;
      public GeneticAlgorithm(int populationSize, float mutationRate, float crossoverRate, int generations)
      {
         this.populationSize = populationSize;
         this.mutationRate = mutationRate;
         this.crossoverRate = crossoverRate;
         this.generations = generations;
         this.population = new List<NeuralNetwork>();
         this.random = new Random();
      }
      public void InitializePopulation(int[] layers, int mutationNumber, int mutationSign, int mutationRandom, int mutationIncrease, int mutationDecrease, int mutationWeakerParentFactor)
      {
         for (int i = 0; i < populationSize; i++)
         {
```

```
                NeuralNetwork nn = new NeuralNetwork(layers, mutationNumber, mutationSign, mutationRandom, mutationIncrease, mutationDecrease, mutationWeakerParentFactor);
                population.Add(nn);
            }
        }
        public void Evolve()
        {
            for (int generation = 0; generation < generations; generation++)
            {
                List<NeuralNetwork> newPopulation = new List<NeuralNetwork>();
                while (newPopulation.Count < populationSize)
                {
                    NeuralNetwork parent1 = SelectParent();
                    NeuralNetwork parent2 = SelectParent();
                    NeuralNetwork child;
                    if (random.NextDouble() < crossoverRate)
                    {
                        child = Crossover(parent1, parent2);
                    }
                    else
                    {
                        child = parent1;
                    }
                    child.mutation.Mutate(ref child.GetWeights(), parent1.GetWeights(), mutationRate);
                    newPopulation.Add(child);
                }
                population = newPopulation;
            }
        }
        private NeuralNetwork SelectParent()
        {
            return population[random.Next(populationSize)];
        }
        private NeuralNetwork Crossover(NeuralNetwork parent1, NeuralNetwork parent2)
        {
            return parent1;
        }
    }
}
```

Mutation.cs

```
namespace UI_visual
{
    using System;
    using System.Linq;
    public class Mutation
    {
        private int mutationNumber;
        private int mutationSign;
        private int mutationRandom;
        private int mutationIncrease;
        private int mutationDecrease;
        private int mutationWeakerParentFactor;
        public Mutation(int mutationNumber, int mutationSign, int mutationRandom, int mutationIncrease, int mutationDecrease, int mutationWeakerParentFactor)
        {
            this.mutationNumber = mutationNumber;
            this.mutationSign = mutationSign;
            this.mutationRandom = mutationRandom;
            this.mutationIncrease = mutationIncrease;
            this.mutationDecrease = mutationDecrease;
            this.mutationWeakerParentFactor = mutationWeakerParentFactor;
        }
        public void Mutate(ref float[][][] weights, float[][][] parentWeights, float mutationRatio)
        {
            int mutationValue = (int)(mutationRatio * 1000 * (1f / (float)mutationWeakerParentFactor));
            for (int i = 0; i < weights.Length; i++)
            {
                for (int j = 0; j < weights[i].Length; j++)
                {
                    for (int k = 0; k < weights[i][j].Length; k++)
                    {
                        int randomNumber = new Random().Next(0, 1000);
                        if (randomNumber < mutationValue)
                        {
                            weights[i][j][k] = parentWeights[i][j][k];
                        }
                    }
                }
            }
```

```
            Mutate(ref weights);
        }
        public void ApplyMutation(ref float[][][] weights)
        {
            Random rand = new Random();
            for (int i = 0; i < weights.Length; i++)
            {
                for (int j = 0; j < weights[i].Length; j++)
                {
                    for (int k = 0; k < weights[i][j].Length; k++)
                    {
                        if (rand.NextDouble() < 0.1) // Mutation rate (10%)
                        {
                            weights[i][j][k] += (float)(rand.NextDouble() * mutationRandom -
mutationRandom / 2.0);
                        }
                    }
                }
            }
        }
        public void ApplyMutation(ref float[][] biases)
        {
            Random rand = new Random();
            for (int i = 0; i < biases.Length; i++)
            {
                for (int j = 0; j < biases[i].Length; j++)
                {
                    if (rand.NextDouble() < 0.1) // Mutation rate (10%)
                    {
                        biases[i][j] += (float)(rand.NextDouble() * mutationRandom -
mutationRandom / 2.0);
                    }
                }
            }
        }
        private void Mutate(ref float[][][] weights)
        {
            for (int i = 0; i < weights.Length; i++)
            {
                for (int j = 0; j < weights[i].Length; j++)
                {
                    for (int k = 0; k < weights[i][j].Length; k++)
```

```
                    {
                        float weight = weights[i][j][k];
                        int randomNumber1 = new Random().Next(1, mutationNumber);
                        if (randomNumber1 <= mutationSign)
                        {
                            weight *= -1f;
                        }
                        else if (randomNumber1 <= mutationSign + mutationRandom)
                        {
                            weight = (float)new Random().NextDouble() * 2 - 1;
                        }
                        else if (randomNumber1 <= mutationSign + mutationRandom +
mutationIncrease)
                        {
                            float factor = (float)new Random().NextDouble() + 1f;
                            weight *= factor;
                        }
                        else if (randomNumber1 <= mutationSign + mutationRandom +
mutationIncrease + mutationDecrease)
                        {
                            float factor = (float)new Random().NextDouble();
                            weight *= factor;
                        }
                        weights[i][j][k] = weight;
                    }
                }
            }
        }
    }
}
```

GraphicsCapture.cs

```
using System;
using System.Runtime.InteropServices;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Graphics.Capture;
using Windows.Graphics.DirectX;
using Windows.Graphics.DirectX.Direct3D11;
using SharpDX.Direct3D11;
using SharpDX.DXGI;
using Win32.Shared;
using Win32.Shared.Interfaces;
using WinRT.GraphicsCapture.Interop;
```

```
using Device = SharpDX.Direct3D11.Device;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Drawing;
namespace WinRT.GraphicsCapture
{
   internal class GraphicsCapture : ICaptureMethod
   {
      private Direct3D11CaptureFramePool _captureFramePool;
      private GraphicsCaptureItem _captureItem;
      private GraphicsCaptureSession _captureSession;
      private const int TargetFrameRate = 25;
      private TimeSpan _targetFrameInterval = TimeSpan.FromMilliseconds(1000.0 /
TargetFrameRate);
      private DateTime _lastFrameTime = DateTime.MinValue;
      public GraphicsCapture()
      {
         IsCapturing = false;
      }
      public bool IsCapturing { get; private set; }

      public void Dispose()
      {
         StopCapture();
      }
      public void SaveScreenshot(string filePath, Device device, Texture2D texture,
int targetWidth, int targetHeight)
      {
         if (texture != null)
         {
            var textureDesc = new Texture2DDescription
            {
               ArraySize = 1,
               BindFlags = BindFlags.None,
               CpuAccessFlags = CpuAccessFlags.Read,
               Format = Format.B8G8R8A8_UNorm,
               Height = texture.Description.Height,
               Width = texture.Description.Width,
               MipLevels = 1,
               OptionFlags = ResourceOptionFlags.None,
               SampleDescription = new SampleDescription(1, 0),
               Usage = ResourceUsage.Staging
```

```
                };
                using (var tempTexture = new Texture2D(device, textureDesc))
                {
                    device.ImmediateContext.CopyResource(texture, tempTexture);
                    var dataBox = device.ImmediateContext.MapSubresource(tempTexture,
0, MapMode.Read, SharpDX.Direct3D11.MapFlags.None);
                    var width = tempTexture.Description.Width;
                    var height = tempTexture.Description.Height;
                    var stride = dataBox.RowPitch;
                    var bitmap = new Bitmap(width, height, stride,
PixelFormat.Format32bppPArgb, dataBox.DataPointer);
                    var newBitmap = new Bitmap(targetWidth, targetHeight);
                    using (var g = Graphics.FromImage(newBitmap))
                    {
                        g.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
                        g.DrawImage(bitmap, 0, 0, targetWidth, targetHeight);
                    }
                    try
                    {
                        newBitmap.Save(filePath, ImageFormat.Png);
                    }
                    catch (System.Runtime.InteropServices.ExternalException ex)
                    {
                        Console.WriteLine($"Error saving image: {ex.Message}");
                        Console.WriteLine($"Stack Trace: {ex.StackTrace}");
                    }
                    device.ImmediateContext.UnmapSubresource(tempTexture, 0);
                }
            }
        }
        public void StartCapture(IntPtr hWnd, Device device, Factory factory)
        {
            var capturePicker = new WindowPicker();
            var captureHandle = capturePicker.PickCaptureTarget(hWnd);
            if (captureHandle == IntPtr.Zero)
                return;
            _captureItem = CreateItemForWindow(captureHandle);
            if (_captureItem == null)
                return;
            _captureItem.Closed += CaptureItemOnClosed;
```

```
            var hr = NativeMethods.CreateDirect3D11DeviceFromDXGIDevice(device.NativePointer, out var pUnknown);
            if (hr != 0)
            {
                StopCapture();
                return;
            }
            var winrtDevice = (IDirect3DDevice)Marshal.GetObjectForIUnknown(pUnknown);
            Marshal.Release(pUnknown);
            _captureFramePool = Direct3D11CaptureFramePool.Create(winrtDevice, DirectXPixelFormat.B8G8R8A8UIntNormalized, 2, _captureItem.Size);
            _captureSession = _captureFramePool.CreateCaptureSession(_captureItem);
            Cursor.Hide();
            _captureSession.StartCapture();
            IsCapturing = true;
        }
        public Texture2D TryGetNextFrameAsTexture2D(Device device)
        {
            using var frame = _captureFramePool?.TryGetNextFrame();
            if (frame == null)
                return null;
            var currentTime = DateTime.Now;
            var elapsed = currentTime - _lastFrameTime;
            if (elapsed < _targetFrameInterval)
            {
                frame.Dispose();
                return null;
            }
            _lastFrameTime = currentTime;
            var surfaceDxgiInterfaceAccess = (IDirect3DDxgiInterfaceAccess)frame.Surface;
            var pResource = surfaceDxgiInterfaceAccess.GetInterface(new Guid("dc8e63f3-d12b-4952-b47b-5e45026a862d"));
            using var surfaceTexture = new Texture2D(pResource);
            var texture2dDescription = new Texture2DDescription
            {
                ArraySize = 1,
                BindFlags = BindFlags.ShaderResource | BindFlags.RenderTarget,
                CpuAccessFlags = CpuAccessFlags.None,
                Format = Format.B8G8R8A8_UNorm,
```

```
                Height = surfaceTexture.Description.Height,
                MipLevels = 1,
                SampleDescription = new SampleDescription(1, 0),
                Usage = ResourceUsage.Default,
                Width = surfaceTexture.Description.Width
            };
            var texture2d = new Texture2D(device, texture2dDescription);

SaveScreenshot($"C:\\1\\screenshot_{currentTime:yyyyMMddHHmmssfff}.png",  device,
surfaceTexture, 854, 480);
            device.ImmediateContext.CopyResource(surfaceTexture, texture2d);
            return texture2d;
        }
        public void StopCapture()
        {
            _captureSession?.Dispose();
            _captureFramePool?.Dispose();
            _captureSession = null;
            _captureFramePool = null;
            _captureItem = null;
            IsCapturing = false;
            Cursor.Show();
        }
        private static GraphicsCaptureItem CreateItemForWindow(IntPtr hWnd)
        {
            var                                    factory                                    =
WindowsRuntimeMarshal.GetActivationFactory(typeof(GraphicsCaptureItem));
            var interop = (IGraphicsCaptureItemInterop)factory;
            var          pointer          =          interop.CreateForWindow(hWnd,
typeof(GraphicsCaptureItem).GetInterface("IGraphicsCaptureItem").GUID);
            var        capture        =        Marshal.GetObjectForIUnknown(pointer)        as
GraphicsCaptureItem;
            Marshal.Release(pointer);
            return capture;
        }
        private void CaptureItemOnClosed(GraphicsCaptureItem sender, object args)
        {
            StopCapture();
        }
    }
}
```

Form1.cs

```
using NeuralNetwork.Model;
using NeuralNetwork.Model.Layers;
using NeuralNetwork.Model.Nodes;
using NeuralNetwork.Visualizer.Contracts.Drawing.Core.Brushes;
using NeuralNetwork.Visualizer.Contracts.Drawing.Core.Pens;
using NeuralNetwork.Visualizer.Contracts.Drawing.Core.Primitives;
using NeuralNetwork.Visualizer.Contracts.Drawing.Core.Text;
using NeuralNetwork.Visualizer.Contracts.Preferences;
using NeuralNetwork.Visualizer.Contracts.Selection;
using NeuralNetwork.Visualizer.Winform.Drawing.Canvas.GdiMapping;
using NeuralNetwork.Visualizer.Preferences.Formatting;
using System;
using System.Linq;
using System.Windows.Forms;
using System.Drawing;
using SolidBrush = NeuralNetwork.Visualizer.Contracts.Drawing.Core.Brushes.SolidBrush;
using Color = NeuralNetwork.Visualizer.Contracts.Drawing.Core.Primitives.Color;
using Pen = NeuralNetwork.Visualizer.Contracts.Drawing.Core.Pens.Pen;
namespace WindowsFormsApp1
{
  public partial class Form1 : Form
  {
    private InputLayer _input;
     public Form1()
     {
        InitializeComponent();
        NeuralNetworkVisualizerControl1.SelectBias += NeuralNetworkVisualizerControl1_SelectBias;
        NeuralNetworkVisualizerControl1.SelectEdge += NeuralNetworkVisualizerControl1_SelectEdge;
        NeuralNetworkVisualizerControl1.SelectInput += NeuralNetworkVisualizerControl1_SelectInput;
        NeuralNetworkVisualizerControl1.SelectInputLayer += NeuralNetworkVisualizerControl1_SelectInputLayer;
        NeuralNetworkVisualizerControl1.SelectNeuron += NeuralNetworkVisualizerControl1_SelectNeuron;
        NeuralNetworkVisualizerControl1.SelectNeuronLayer += NeuralNetworkVisualizerControl1_SelectNeuronLayer;
     }
     private void NeuralNetworkVisualizerControl1_SelectNeuronLayer(object sender, SelectionEventArgs<NeuronLayer> e)
```

```
        {
            ShowSelectedElements();
            ShowLastSelection($"Neuron Layer: {e.Element.Id}", e.IsSelected);
        }

        private void NeuralNetworkVisualizerControl1_SelectNeuron(object sender,
SelectionEventArgs<Neuron> e)
        {
            ShowSelectedElements();
            ShowLastSelection($"Neuron: {e.Element.Id}", e.IsSelected);
        }
        private void NeuralNetworkVisualizerControl1_SelectInputLayer(object sender,
SelectionEventArgs<InputLayer> e)
        {
            ShowSelectedElements();
            ShowLastSelection($"Input Layer: {e.Element.Id}", e.IsSelected);
        }
        private void NeuralNetworkVisualizerControl1_SelectInput(object sender,
SelectionEventArgs<Input> e)
        {
            ShowSelectedElements();
            ShowLastSelection($"Input: {e.Element.Id}", e.IsSelected);
        }
        private void NeuralNetworkVisualizerControl1_SelectEdge(object sender,
SelectionEventArgs<Edge> e)
        {
            ShowSelectedElements();
            ShowLastSelection($"Edge: {e.Element.Id}", e.IsSelected);
        }
        private void NeuralNetworkVisualizerControl1_SelectBias(object sender,
SelectionEventArgs<Bias> e)
        {
            ShowSelectedElements();
            ShowLastSelection($"Bias: {e.Element.Id}", e.IsSelected);
        }
        private void ShowSelectedElements()
        {
            txtSelectedElements.Text = string.Join(", ",
NeuralNetworkVisualizerControl1.SelectedElements.Select(se => se.Id));
        }
        private void ShowLastSelection(string text, bool isSelected)
        {
```

```
            txtLastSelected.BackColor = isSelected ? Color.LightGreen.ToGdi() :
Color.LightPink.ToGdi();
            txtLastSelected.Text = text;
          }
          private void Form1_Load(object sender, EventArgs e)
          {
            NeuralNetworkVisualizerControl1.Preferences.AutoRedrawMode =
AutoRedrawMode.AutoRedrawAsync;
              NeuralNetworkVisualizerControl1.Preferences.Quality =
(NeuralNetwork.Visualizer.Contracts.Preferences.RenderQuality)RenderQuality.High;
            cboQuality.Items.Add(RenderQuality.Low);
            cboQuality.Items.Add(RenderQuality.Medium);
            cboQuality.Items.Add(RenderQuality.High);
            cboQuality.SelectedItem =
NeuralNetworkVisualizerControl1.Preferences.Quality;
            NeuralNetworkVisualizerControl1.Preferences.Inputs.OutputValueFormatter =
new ByValueSignFormatter<FontLabel>(
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Red)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Gray)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black))
            );
            NeuralNetworkVisualizerControl1.Preferences.Neurons.OutputValueFormatter
= new ByValueSignFormatter<FontLabel>(
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Red)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Gray)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black))
            );
            NeuralNetworkVisualizerControl1.Preferences.Edges.WeightFormatter = new
ByValueSignFormatter<FontLabel>(
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Red)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Gray)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black)),
              new FontLabel(FontLabel.Default, new SolidBrush(Color.Black))
            );
            NeuralNetworkVisualizerControl1.Preferences.Edges.ConnectorFormatter =
new CustomFormatter<Pen>((v) => v == 0.0 ? Pen.BasicFromColor(Color.LightGray) :
Pen.BasicFromColor(Color.Black));
            NeuralNetworkVisualizerControl1.Preferences.AsyncRedrawOnResize =
chAsyncRedrawOnResize.Checked;
            NeuralNetworkVisualizerControl1.Preferences.Layers.Title = null;
```

```
        }
         private void btnStart_Click(object sender, EventArgs e)
         {
            _input = new InputLayer("Input")
            {
               Bias = new Bias("bias") { OutputValue = 1.234 }
            };
            Random random = new Random();
            for (int i = 1; i <= 66; i++)
            {
               double outputValue = random.NextDouble();
               _input.AddNode(new Input($"e{i}") { OutputValue = outputValue });
            }
            var hidden1 = new NeuronLayer("Hidden1");
            for (int i = 1; i <= 48; i++)
            {
               hidden1.AddNode(new Neuron($"h1_{i}") { ActivationFunction =
ActivationFunction.Tanh });
            }
            _input.Connect(hidden1);

            var hidden2 = new NeuronLayer("Hidden2");
            for (int i = 1; i <= 32; i++)
            {
               hidden2.AddNode(new Neuron($"h2_{i}") { ActivationFunction =
ActivationFunction.Tanh });
            }
            hidden1.Connect(hidden2);
            var output = new NeuronLayer("Output");
            for (int i = 1; i <= 4; i++)
            {
               output.AddNode(new Neuron($"s{i}") { ActivationFunction =
ActivationFunction.Softmax });
            }
            hidden2.Connect(output);
            foreach (var p in hidden1.Nodes)
            {
               foreach (var edge in p.Edges)
               {
                  edge.Weight = (random.NextDouble() - 0.5) * 2;
               }
            }
```

```
            foreach (var p in hidden2.Nodes)
            {
               foreach (var edge in p.Edges)
               {
                  edge.Weight = (random.NextDouble() - 0.5) * 2;
               }
            }
            foreach (var p in output.Nodes)
            {
               foreach (var edge in p.Edges)
               {
                  edge.Weight = (random.NextDouble() - 0.5) * 2;
               }
            }
            NeuralNetworkVisualizerControl1.InputLayer = _input;
            btnChangeValue.Enabled = btnAddBias.Enabled = btnClear.Enabled =
trackZoom.Enabled = cboQuality.Enabled = true;
         }
         private async void btnChangeValue_Click(object sender, EventArgs e)
        {
          NeuralNetworkVisualizerControl1.SuspendAutoRedraw();
          var edge = _input.Find<Edge>("Input.bias - Hidden.o1");
          edge.Weight = 0.123;
          var node = _input.Nodes.Single(n => n.Id == "e3");
          node.OutputValue = 1.44444;
          await NeuralNetworkVisualizerControl1.ResumeAutoRedraw();
        }
        private void btnAddBias_Click(object sender, EventArgs e)
        {
          AddHiddenBias();
        }
        private async void AddHiddenBias()
        {
          NeuralNetworkVisualizerControl1.SuspendAutoRedraw();
          var newbias = new Bias("HiddenBias") { OutputValue = 0.777 };
          _input.Next.Bias = newbias;
          var outputs = _input.Next.Next.Nodes;
          var edges = outputs.SelectMany(o => o.Edges.Where(e => e.Source ==
newbias));
          double weight = 1.99;
          foreach (var edge in edges)
```

```
            {
              edge.Weight = weight;
              weight++;
            }
            await NeuralNetworkVisualizerControl1.ResumeAutoRedraw();
          }
          private void trackZoom_Scroll(object sender, EventArgs e)
          {
            NeuralNetworkVisualizerControl1.Zoom = trackZoom.Value / 10f;
          }
          private void btnClear_Click(object sender, EventArgs e)
          {
            NeuralNetworkVisualizerControl1.InputLayer = null;
            btnChangeValue.Enabled = btnAddBias.Enabled = btnClear.Enabled =
trackZoom.Enabled = cboQuality.Enabled = false;
          }

           private void cboQuality_SelectedIndexChanged(object sender, EventArgs e)
           {
              NeuralNetworkVisualizerControl1.Preferences.Quality =
(NeuralNetwork.Visualizer.Contracts.Preferences.RenderQuality)cboQuality.SelectedItem;
              NeuralNetworkVisualizerControl1.Redraw();
           }
           private void chSelectable_CheckedChanged(object sender, EventArgs e)
          {
            NeuralNetworkVisualizerControl1.Preferences.Selectable =
chSelectable.Checked;
          }
          private void chAsyncRedrawOnResize_CheckedChanged(object sender,
EventArgs e)
          {
            NeuralNetworkVisualizerControl1.Preferences.AsyncRedrawOnResize =
chAsyncRedrawOnResize.Checked;
          }
        }
      }
```

ДОДАТОК Г

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_____Програмно–апаратний комплекс для автопілотажу засобами нейромережевого моделювання та генетичних алгоритмів_____
Тип роботи:_____Магістерська кваліфікаційна робота_____
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))
Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет (інститут), навчальна група)

Керівник_____к.т.н., проф. Азарова А.О._____
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	86
Загальнасхожість	14

Аналіз звіту подібності (відмітитипотрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор_____Хорошун В.Р._____
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____

Особа, відповідальна за перевірку_____ _Захарченко С.М__
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)