МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

МАМЧУР Сергій Віталійович

Програмний модуль класифікації зображень на основі згорткової нейронної мережі на вбудованій обчислювальній платформі NVIDIA Jetson / Software Module for Image Classification Based on a Convolutional Neural Network on an Embedded NVIDIA Jetson Computing Platform

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
Мамчур С.В.

_____

Науковий керівник
к.т.н., доцент Д.І. Загородня

_____

Кваліфікаційну роботу допущено до
захисту
«___»_____ 20___ р.
Завідувач кафедри
_____ Н. В. Дзюбановська

Тернопіль – 2026

Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп’ютерні науки
освітньо-професійна програма – Комп’ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
МАМЧУР Сергій Віталійович
(прізвище, ім’я, по батькові)

1. Програмний модуль класифікації зображень на основі згорткової нейронної мережі на вбудованій обчислювальній платформі NVIDIA Jetson / Software Module for Image Classification Based on a Convolutional Neural Network on an Embedded NVIDIA Jetson Computing Platform
керівник роботи Д. І. Загородня к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 01 грудня 2025 р. № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові публікації з комп'ютерного зору та глибинного навчання, технічна документація NVIDIA Jetson, програмні засоби TensorFlow, PyTorch, TensorRT та OpenCV.
4. Основні питання, які потрібно розробити:
- аналіз сучасних методів класифікації зображень у системах комп’ютерного зору;
- дослідження архітектур згорткових нейронних мереж та методів їх оптимізації для вбудованих систем;
- аналіз архітектурних особливостей та можливостей платформи NVIDIA J□ts□n N□n□;
- розробка архітектури програмного модуля класифікації зображень на основі згорткової нейронної мережі;
- програмна реалізація та оптимізація моделі з використанням TensorRT і FP16-квантування;
- експериментальна перевірка точності класифікації, швидкодії та ресурсоємності програмного модуля.
5. Перелік графічного матеріалу в роботі:
- узагальнена схема процесу класифікації зображень;
- логічна архітектура цільової платформи NVIDIA J□ts□n N□n□;
- архітектурна модель програмного модуля та схема потоків даних.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 1 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студент_____ С.В. Мамчур_____
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Д. І. Загородня
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль класифікації зображень на основі згорткової нейронної мережі на вбудованій обчислювальній платформі NVIDIA Jetson» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 76 сторінок і містить 19 ілюстрацій, 8 таблиць, 4 додатка та 35 використаних джерела.

Метою роботи є підвищення ефективності класифікації зображень у вбудованих системах штучного інтелекту шляхом розробки та оптимізації програмного модуля на основі згорткової нейронної мережі для обчислювальної платформи NVIDIA Jetson.

Методами дослідження обрано методи системного аналізу для дослідження сучасних підходів до класифікації зображень та технологій комп'ютерного зору, методи об'єктно-орієнтованого проєктування для побудови архітектури програмного модуля, методи машинного навчання та глибинного навчання для створення і навчання згорткових нейронних мереж, методи оптимізації моделей для граничних обчислень, а також експериментальні методи оцінювання ефективності за показниками точності, швидкодії та використання обчислювальних ресурсів.

У результаті виконання роботи проведено аналіз сучасних архітектур згорткових нейронних мереж та методів їх адаптації до вбудованих систем штучного інтелекту, досліджено особливості обчислювальної платформи NVIDIA Jetson, розроблено архітектуру програмного модуля класифікації зображень, реалізовано та оптимізовано модель нейронної мережі для виконання інференсу в режимі реального часу. Розроблений програмний модуль забезпечує ефективну класифікацію зображень на вбудованих пристроях із досягненням оптимального балансу між точністю розпізнавання та швидкодією роботи системи.

Ключові слова: КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, КОМП'ЮТЕРНИЙ ЗІР, ШТУЧНИЙ ІНТЕЛЕКТ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, NVIDIA JETSON, EDGE AI, ГЛИБИННЕ НАВЧАННЯ, ВБУДОВАНІ СИСТЕМИ.

ANNOTATION

The qualification thesis entitled “Software Module for Image Classification Based on a Convolutional Neural Network on the NVIDIA Jetson Embedded Computing Platform” submitted for the Bachelor's degree in specialty 122 “Computer Science” under the educational program “Computer Science” consists of 76 pages and includes 19 figures, 8 tables, 4 appendices, and 35 references.

The purpose of the study is to improve the efficiency of image classification in embedded artificial intelligence systems through the development and optimization of a software module based on a convolutional neural network for the NVIDIA Jetson computing platform.

Research methods: system analysis methods were used to investigate modern approaches to image classification and computer vision technologies; object-oriented design methods were applied to develop the architecture of the software module; machine learning and deep learning methods were employed to create and train convolutional neural networks; model optimization techniques for edge computing were utilized; and experimental evaluation methods were applied to assess performance in terms of classification accuracy, inference speed, and computational resource utilization.

As a result of the study, modern convolutional neural network architectures and methods for their adaptation to embedded artificial intelligence systems were analyzed. The architectural features of the NVIDIA Jetson computing platform were investigated, the architecture of the image classification software module was developed, and the neural network model was implemented and optimized for real-time inference. The developed software module provides efficient image classification on embedded devices while achieving an optimal balance between recognition accuracy and system performance.

Keywords: IMAGE CLASSIFICATION, COMPUTER VISION, ARTIFICIAL INTELLIGENCE, CONVOLUTIONAL NEURAL NETWORKS, NVIDIA JETSON, EDGE AI, DEEP LEARNING, EMBEDDED SYSTEMS.

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким поширенням систем штучного інтелекту та переходом від хмарних обчислень до обробки даних безпосередньо на кінцевих (вбудованих) пристроях — у парадигмі граничних обчислень (Edge AI). Ключову роль у цих процесах відіграють технології комп’ютерного зору, що дозволяють автономним системам, робототехніці, безпілотним апаратам та розумним камерам сприймати й інтерпретувати візуальне середовище в режимі реального часу.

Незважаючи на високу точність сучасних згорткових нейронних мереж, їх застосування на мобільних та вбудованих пристроях суттєво обмежується через високу обчислювальну складність, значні вимоги до обсягу пам'яті та енергоспоживання. Передача візуальних даних у хмарні сервіси для обробки часто є неприйнятною через затримки в мережі, ризики порушення конфіденційності та залежність від стабільного інтернет-з'єднання. Це зумовлює необхідність перенесення обчислювального навантаження безпосередньо на апаратні вузли збору даних.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення ефективних програмних рішень, здатних забезпечити високу швидкість та точність класифікації зображень безпосередньо на вбудованих платформах, таких як сімейство NVIDIA Jetson. Це вимагає використання не лише оптимізованих легковагових архітектур нейронних мереж, але й специфічних методів апаратного прискорення, оптимізації графів обчислень та квантування ваг моделі для максимального використання можливостей графічних ядер пристрою.

Додатковим фактором, що підсилює актуальність дослідження, є зростаюча потреба індустрії у розгортанні автономних систем, здатних працювати в умовах жорстких обмежень на енергоспоживання і тепловіддачу. Забезпечення оптимального балансу між точністю класифікації та продуктивністю системи на апаратному рівні є складним інженерно-науковим завданням.

У зв’язку з цим виникає потреба у розробці та дослідженні програмних модулів класифікації зображень, адаптованих до архітектурних особливостей

платформи NVIDIA Jetson, з урахуванням оптимізації процесу логічного висновку (інференсу) нейронної мережі.

Метою кваліфікаційної роботи є підвищення ефективності класифікації зображень у вбудованих системах штучного інтелекту шляхом розробки та оптимізації програмного модуля на основі згорткової нейронної мережі для обчислювальної платформи NVIDIA Jetson.

Для досягнення поставленої мети в роботі необхідно розв'язати такі основні завдання:

- провести аналіз сучасних архітектур згорткових нейронних мереж та методів їх оптимізації для граничних обчислень;
- дослідити апаратні особливості та обмеження обчислювальної платформи NVIDIA Jetson;
- розробити архітектуру програмного модуля класифікації зображень та алгоритми підготовки даних;
- здійснити навчання обраної моделі нейронної мережі та її конвертацію у формат, оптимізований для апаратного прискорення;
- реалізувати програмний модуль та інтегрувати його у цільове середовище;
- провести експериментальну оцінку ефективності роботи модуля за показниками точності, швидкодії та ресурсоємності.

Об'єктом дослідження є процеси автоматизованої класифікації зображень у системах комп'ютерного зору.

Предметом дослідження є методи та алгоритми оптимізації згорткових нейронних мереж для роботи на вбудованій обчислювальній платформі NVIDIA Jetson.

Особливість отриманих результатів полягає в удосконаленні процесу адаптації згорткових нейронних мереж до обмежених обчислювальних ресурсів вбудованих систем за рахунок комплексного застосування методів оптимізації та апаратного прискорення. Запропонований підхід дозволяє досягти оптимального компромісу між точністю розпізнавання та швидкодією системи без суттєвих втрат інформативності ознак зображення.

Практична значимість роботи полягає у створенні готового до використання програмного модуля, який забезпечує класифікацію зображень у режимі реального часу. Результати дослідження можуть бути безпосередньо інтегровані в системи інтелектуального відеоспостереження, безпілотні апарати, системи контролю якості на виробництві або автономну робототехніку, де потрібна обробка візуальних даних без залучення хмарних обчислень.

Результати кваліфікаційної роботи були апробовані на IV Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі», яка відбулася 20 травня 2026р. (м. Тернопіль, Україна) та опубліковані у збірнику матеріалів конференції в доповіді «Програмний модуль класифікації зображень на основі згорткової нейронної мережі для платформи NVIDIA Jetson». Текст публікації наведено у додатку А.

1 ТЕОРЕТИЧНІ ОСНОВИ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ ТА РОБОТИ ВБУДОВАНИХ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Задачі класифікації зображень у сучасних системах комп'ютерного зору

У сучасному інформаційному суспільстві технології комп’ютерного зору відіграють ключову роль у процесах автоматизації аналізу візуальних даних. Завдання класифікації зображень є однією з фундаментальних проблем комп’ютерного зору, що полягає у віднесенні вхідного зображення до однієї з наперед визначених категорій (класів) на основі його візуальних ознак [1]. Здатність алгоритмічно визначати семантичний вміст зображень є базою для вирішення більш складних задач, таких як детекція об'єктів, семантична сегментація та трекінг об'єктів у відеопотоці.

Математично задачу класифікації зображень можна сформулювати як пошук оптимальної функції відображення f, яка перетворює вхідний тензор пікселів X у вектор ймовірностей належності до заданих класів Y [2]:

$$Y = f(X, W) \tag{1.1}$$

де X — вхідне зображення у вигляді багатовимірного масиву (наприклад, розмірності H x W x C, де H — висота, W — ширина, C — кількість кольорових каналів);

W — матриця параметрів (вагових коефіцієнтів) моделі класифікатора, що підлягає оптимізації під час навчання; Y — вихідний вектор розмірності N (кількість класів), елементи якого відображають ймовірність належності зображення X до відповідної категорії.

Процес класифікації візуальної інформації зазвичай складається з декількох послідовних етапів. На рисунку 1.1 подано узагальнену схему цього конвеєра (pipeline) обробки даних.

Історично підходи до розв'язання задачі класифікації зображень поділяються на класичні методи комп'ютерного зору та сучасні методи на основі глибокого

машинного навчання. Класичні алгоритми базуються на ручному конструюванні екстракторів ознак (наприклад, дескрипторів SIFT, HOG, SURF), після чого сформований вектор ознак подається на вхід традиційного класифікатора, такого як метод опорних векторів (SVM) або алгоритм випадкового лісу (Random Forest) [3].

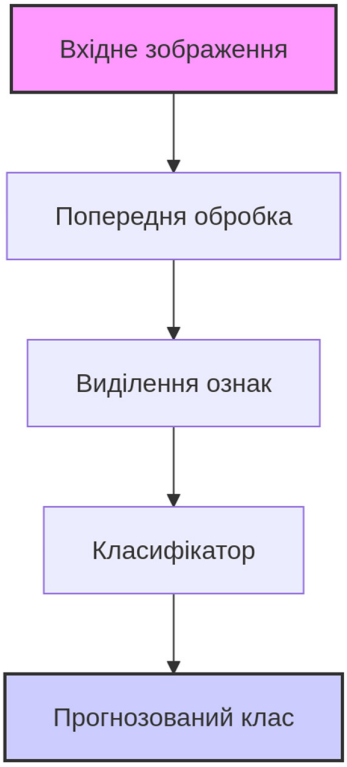

Рисунок 1.1 – Узагальнена схема процесу класифікації зображень

Натомість, методи глибокого навчання, зокрема згорткові нейронні мережі (CNN), здатні самостійно виділяти ієрархічні візуальні ознаки безпосередньо з «сирих» піксельних даних у процесі навчання [4]. Порівняльну характеристику цих двох парадигм наведено у таблиці 1.1.

Незважаючи на домінування нейромережевих підходів, сучасні системи штучного інтелекту продовжують стикатися з низкою практичних проблем розпізнавання. Серед основних викликів виділяють варіативність освітлення сцен, зміну ракурсу зйомки, часткове перекриття (оклюзію) цільових об'єктів, наявність фонового шуму, а також внутрішньокласову варіативність [5].

Особливо гостро ці проблеми постають під час розгортання моделей у вбудованих та мобільних системах (Edge AI). Забезпечення високої точності

розпізнавання за умов жорстких апаратних обмежень (лімітована обчислювальна потужність, обмежена пропускна здатність оперативної пам'яті та ліміти енергоспоживання) вимагає впровадження спеціалізованих технік оптимізації програмно-апаратного комплексу, що і є предметом розгляду в наступних розділах даної роботи.

Таблиця 1.1 — Порівняльна характеристика підходів до класифікації зображень

Характеристика	Класичні методи комп'ютерного зору	Методи глибокого навчання (CNN)
Процес виділення ознак	Ручне математичне конструювання (HOG, SIFT)	Автоматичне (ієрархічне) під час навчання
Вимоги до обсягу даних	Низькі або середні (сотні зображень)	Високі (потребують тисячі/мільйони зразків)
Обчислювальна складність	Середня (допустимо використання CPU)	Висока (потребує GPU/TPU для навчання)
Схильність до перенавчання	Переважно низька	Висока (вимагає регуляризації та аугментації)
Точність розпізнавання	Обмежена у складних умовах	state-of-the-art на великомасштабних датасетах

Натомість, застосування глибоких згорткових нейронних мереж (CNN) дозволяє вирішити цю проблему завдяки здатності до автоматичного формування багаторівневої ієрархії ознак безпосередньо з «сирих» піксельних даних. У процесі навчання мережа самостійно адаптує просторові фільтри, виокремлюючи інваріантні до масштабу та освітлення семантичні дескриптори цільових об'єктів. Це обґрунтовує доцільність використання легковагових нейромережевих архітектур як математичного ядра розроблюваного програмного модуля для платформи NVIDIA Jetson.

1.2 Огляд і аналіз існуючих рішень для обробки візуальних даних

У сучасних системах комп’ютерного зору задача класифікації зображень вирішується за допомогою широкого спектра методів, що пройшли еволюцію від класичних алгоритмів обробки зображень до складних нейромережевих моделей і спеціалізованих рішень для вбудованих обчислювальних платформ. Аналіз існуючих підходів дозволяє визначити їх переваги, обмеження та доцільність застосування в умовах обмежених ресурсів Edge AI.

На ранніх етапах розвитку комп’ютерного зору домінували класичні методи, які базуються на ручному виділенні ознак. Зокрема, широко застосовуються дескриптори SIFT, SURF, HOG та ORB [1–3], що дозволяють формувати інформативні представлення зображень у вигляді векторів ознак. Отримані ознаки передаються до традиційних класифікаторів, таких як метод опорних векторів або випадковий ліс. Незважаючи на відносно низькі вимоги до обчислювальних ресурсів, ці підходи характеризуються обмеженою точністю та недостатньою стійкістю до змін умов зйомки, а також потребують ручного проєктування ознак, що ускладнює їх адаптацію до нових задач.

Подальший розвиток галузі пов’язаний із впровадженням методів глибокого навчання, зокрема згорткових нейронних мереж (Convolutional Neural Networks, CNN), які є основним інструментом сучасного комп’ютерного зору [4–6]. Їхня архітектура базується на використанні операції математичної згортки для виділення локальних просторових ознак зображення та натхненна принципами роботи зорової кори біологічних організмів. На відміну від повнозв’язних мереж, CNN дозволяють суттєво зменшити кількість параметрів завдяки використанню локальних полів сприйняття та механізму спільного використання ваг [7].

Еволюція архітектур згорткових мереж пройшла шлях від класичних послідовних моделей (наприклад, AlexNet, VGG) до значно глибших структур із залишковими (residual) зв'язками, таких як ResNet [8]. Проте, оскільки цільовою платформою в даному дослідженні є вбудована обчислювальна система NVIDIA Jetson, ключовим критерієм вибору базової моделі є баланс між точністю класифікації та обчислювальною складністю, яка вимірюється кількістю операцій

множення-додавання з плаваючою комою (FLOPs).

Для зменшення обчислювального навантаження в архітектурах, орієнтованих на мобільні та вбудовані платформи, було розроблено низку структурних оптимізацій. Зокрема, в архітектурах сімейства MobileNet використовується метод глибинно-сепарабельної згортки (Depthwise Separable Convolution). Цей метод розбиває стандартну згортку на два етапи: просторову згортку для кожного вхідного каналу окремо (Depthwise Convolution) та точкову згортку 1 x 1 для лінійного комбінування отриманих каналів (Pointwise Convolution). На рисунках 1.2 та 1.3 подано логічні схеми стандартної та глибинно-сепарабельної згорток відповідно.

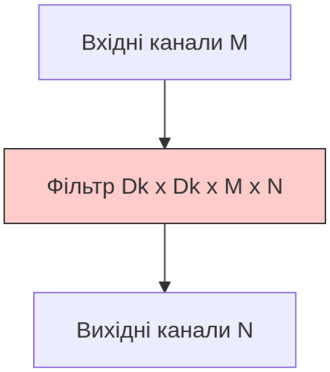

Рисунок 1.2 – Логічна схема стандартної згортки

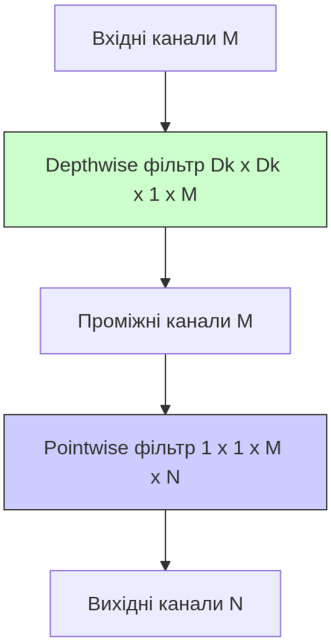

Рисунок 1.3 – Логічна схема глибинно-сепарабельної згортки

Завдяки застосуванню глибинно-сепарабельних згорток досягається скорочення обчислювальних витрат у N разів (де N — кількість вихідних каналів) порівняно зі стандартними підходами, що робить їх ідеальними для платформ класу Edge AI. Крім MobileNet, високу ефективність демонструє сімейство моделей EfficientNet, у яких застосовано метод комплексного масштабування (compound scaling) для одночасної оптимізації ширини, глибини та роздільної здатності мережі [10].

Для обґрунтованого вибору базової моделі нейронної мережі для програмного модуля доцільно провести порівняльний аналіз сучасних легковагових архітектур. У таблиці 1.2 наведено їхні ключові технічні характеристики.

Таблиця 1.2 — Порівняльний аналіз архітектур CNN для вбудованих систем

Архітектура	Кількість параметрів (млн)	Кількість операцій (MFLOPs)	Точність Top-1 (ImageNet)	Цільове застосування
ResNet-18	11.7	~1800	69.8%	Базова серверна обробка / Edge GPU
MobileNetV2	3.4	~300	72.0%	Мобільні пристрої, IoT, Edge AI
MobileNetV3-Small	2.5	~56	67.4%	Мікроконтролери, малопотужний Edge
EfficientNet-B0	5.3	~390	77.1%	Високоточний Edge AI (NVIDIA Jetson)

Аналіз даних таблиці свідчить, що для реалізації на платформі NVIDIA Jetson найбільш доцільним є використання архітектур класу MobileNetV2 або EfficientNet-B0. Вони забезпечують високу точність при відносно невеликій кількості параметрів, що дозволить ефективно застосувати інструменти апаратного прискорення (такі як NVIDIA TensorRT) для забезпечення обробки даних у режимі реального часу.

Для забезпечення роботи нейромережевих моделей у режимі реального часу на платформі NVIDIA Jetson ключову роль відіграють методи компресії та апаратної

оптимізації графа обчислень. Найбільш ефективними серед них є квантування (quantization) та відсікання нейронних зв'язків (pruning).

Квантування — це процес зниження розрядності представлення числових даних моделі. Стандартні архітектури навчаються з використанням 32-бітної арифметики з плаваючою комою (FP32), що забезпечує високу точність градієнтного спуску. Проте для етапу логічного висновку (інференсу) така точність часто є надлишковою. Процес квантування полягає у відображенні (маппінгу) неперервного або високоточного діапазону значень у дискретний простір нижчої розрядності, наприклад, у 16-бітний (FP16) або цілочисельний 8-бітний (INT8) формати.

Математично процес лінійного (афінного) квантування для переходу в INT8 описується наступною формулою:

$$q = \text{round}\left(\frac{r}{S} + Z\right) \tag{1.2}$$

де r — дійсне значення параметру моделі (ваги або активації) у форматі FP32;

S (Scale) — масштабний множник, що визначає крок квантування;

Z (Zero-point) — цілочисельне значення, яке відповідає дійсному нулю;

q — результуюче квантоване значення.

У випадку переходу від FP32 до FP16 (який нативно підтримується архітектурою GPU NVIDIA Maxwell на платі Jetson Nano), оптимізація відбувається шляхом усічення кількості біт мантиси (з 23 до 10) та експоненти (з 8 до 5). Такий підхід дозволяє вдвічі зменшити обсяг оперативної пам'яті, необхідний для зберігання ваг моделі, та пропорційно знизити навантаження на спільну шину передачі даних, практично не втрачаючи вихідної точності класифікації (падіння Accuracy зазвичай не перевищує 0.5-1%).

Альтернативним або додатковим методом компресії є прунінг — метод розрідження нейронної мережі шляхом видалення малозначущих вагових коефіцієнтів. Математична гіпотеза методу полягає в тому, що значення ваг, близькі до нуля (які задовольняють умову |w| < \tau, де \tau — заданий поріг відсікання),

мають мінімальний вплив на фінальний результат функції активації. Шляхом примусового обнулення таких параметрів формуються розріджені матриці (sparse matrices). Хоча прунінг дозволяє суттєво зменшити теоретичну кількість макрооперацій множення-додавання (FLOPs), його практична швидкодія на стандартних графічних процесорах (без спеціалізованих тензорних ядер для розріджених обчислень) може бути обмеженою. З огляду на архітектурні особливості цільової платформи, у даній роботі основним методом підвищення обчислювальної ефективності обрано квантування у поєднанні зі структурним злиттям шарів за допомогою TensorRT.

Окрім вибору архітектури, важливим є використання програмних засобів реалізації, серед яких найбільш поширеними є TensorFlow, PyTorch, OpenCV та ONNX. Проте їх безпосереднє застосування на вбудованих платформах є неефективним через відсутність оптимізації під конкретну апаратну архітектуру.

Для підвищення продуктивності систем на платформі NVIDIA Jetson застосовуються методи апаратної оптимізації, зокрема оптимізація графа обчислень, злиття шарів та квантування вагових коефіцієнтів. Ключовим інструментом є NVIDIA TensorRT, який дозволяє значно підвищити швидкодію інференсу. Використання таких підходів, зокрема FP16-квантування, дозволяє зменшити навантаження на пам’ять і забезпечити роботу системи у режимі реального часу. Для узагальнення розглянутих підходів у таблиці 1.3 наведено їх порівняльну характеристику.

Проведений аналіз свідчить, що класичні методи комп’ютерного зору не забезпечують необхідного рівня точності для сучасних задач, тоді як повноцінні згорткові нейронні мережі є надмірно ресурсоємними для використання на вбудованих платформах. Найбільш ефективним підходом є використання легковагових архітектур у поєднанні з методами апаратної оптимізації. Це, у свою чергу, визначає доцільність розробки спеціалізованого програмного модуля класифікації зображень, адаптованого до умов граничних обчислень та орієнтованого на використання платформи NVIDIA Jetson.

Таблиця 1.3 – Порівняльний аналіз відомих рішень

Підхід	Точність	Швидкодія	Вимоги до ресурсів	Придатність для Jetson
Класичні методи (SIFT, HOG)	Низька–середня	Висока	Низькі	Висока
CNN (VGG, ResNet)	Висока	Низька	Дуже високі	Низька
EfficientNet	Висока	Середня	Високі	Середня
MobileNetV2	Висока	Висока	Низькі	Висока
TensorRT-оптимізовані моделі	Висока	Дуже висока	Оптимізовані	Дуже висока

Реалізація систем комп'ютерного зору на кінцевих пристроях (Edge AI) кардинально відрізняється від хмарних обчислень через наявність жорстких апаратних обмежень. Основною парадигмою проєктування таких систем є мінімізація показників SWaP (Size, Weight, and Power — розмір, вага та енергоспоживання) при збереженні прийнятного рівня обчислювальної продуктивності [11].

Апаратний комплекс фізично складається з двох ключових компонентів: обчислювального модуля (System-on-Module, SoM) моделі P3450 та базової плати вводу-виводу (Carrier Board), яка забезпечує інтерфейси взаємодії з периферійними пристроями (камерами, дисплеями, мережею).

Архітектура Jetson Nano будується на базі концепції «система-на-кристалі» (SoC) Tegra X1. Головною особливістю цієї архітектури є гетерогенність обчислювальних ресурсів: на одному кристалі інтегровано 4-ядерний центральний процесор ARM Cortex-A57 та графічний процесор (GPU) архітектури NVIDIA Maxwell із 128 CUDA-ядрами [12]. Логічну структуру архітектури цільової платформи наведено на рисунку 1.4.

Ключовою архітектурною перевагою платформи є використання уніфікованої архітектури пам'яті (Unified Memory Architecture, UMA). Оперативна пам'ять обсягом 4 ГБ є спільною для CPU та GPU. Це дозволяє реалізувати механізм «zero-copy», що усуває накладні витрати часу та енергії на копіювання вхідних зображень і тензорів через шину PCIe, як це відбувається у класичних десктопних системах [13].

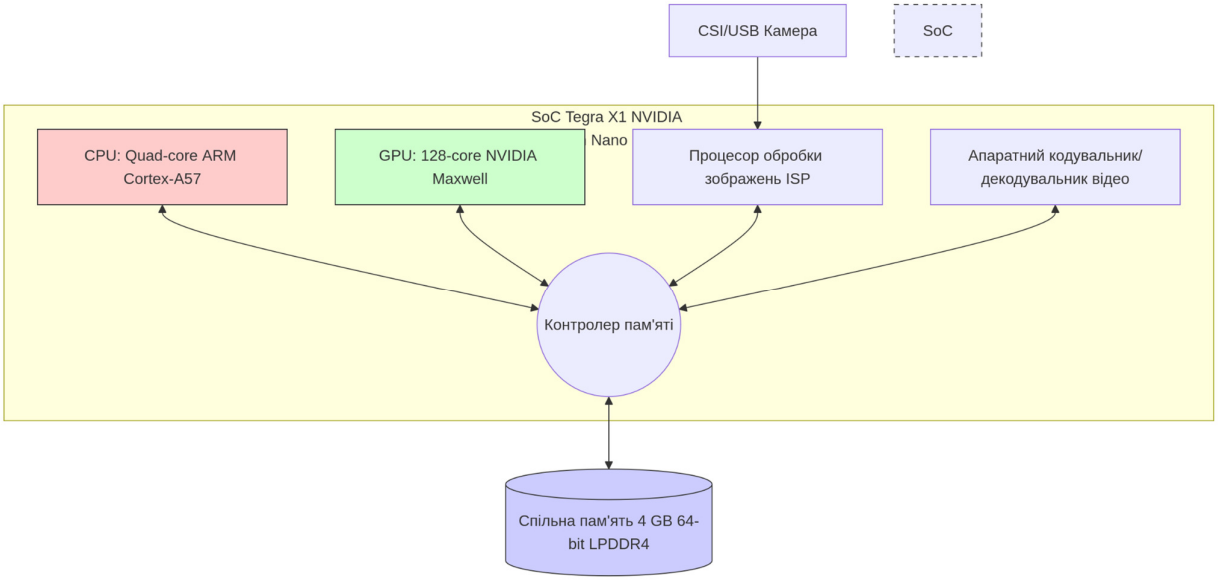

Рисунок 1.4 – Логічна архітектура цільової платформи NVIDIA Jetson Nano

Незважаючи на переваги UMA, спільна шина пам'яті з пропускною здатністю 25.6 ГБ/с стає основним «вузьким місцем» (bottleneck) системи під час логічного висновку (інференсу) згорткових нейронних мереж. Аналіз продуктивності базується на класичній моделі «Roofline» [14]. Основні технічні характеристики обраної платформи наведено у таблиці 1.4.

З огляду на дані таблиці 1.4, можна зробити висновок, що хоча Jetson Nano є потужним рішенням для граничних обчислень, прямий запуск нейронних мереж (наприклад, файлів формату .h5 або .pt) з використанням стандартних фреймворків TensorFlow чи PyTorch буде вкрай неефективним. Робота пристрою на максимальних частотах GPU з неоптимізованими графами обчислень неминуче призведе до переповнення оперативної пам'яті, швидкого нагрівання кристала та увімкнення механізмів температурного тротлінгу (thermal throttling), що різко знизить частоту кадрів (FPS).

Таким чином, для забезпечення ефективної класифікації зображень у режимі реального часу на платформі Jetson Nano необхідно застосувати інструментальні засоби NVIDIA (насамперед оптимізатор TensorRT). Це дозволить максимізувати арифметичну інтенсивність I (за рахунок злиття шарів — layer fusion) та знизити навантаження на підсистему пам'яті шляхом квантування вагових коефіцієнтів з

формату FP32 до FP16, що апаратно підтримується архітектурою Maxwell.

Таблиця 1.4 — Апаратні характеристики цільової платформи NVIDIA Jetson Nano (модель P3450) [12]

Характеристика	Значення	Вплив на розробку програмного модуля
GPU	128-core Maxwell	Вимагає конвертації моделей у формат TensorRT для прискорення
CPU	Quad-core ARM Cortex-A57 @ 1.43 GHz	Використовується лише для попередньої обробки даних
Пам'ять (RAM)	4 GB 64-bit LPDDR4	Обмежує розмір пакету (batch size) до 1; неможливість запуску важких моделей
Пропускна здатність	25.6 ГБ/с	Потребує квантування ваг мережі для зменшення навантаження на шину
Енергоспоживання	Два режими: 5 Вт та 10 Вт	Встановлює жорсткі рамки на тепловиділення; ризик тротлінгу при 100% завантаженні

1.3 Постановка задачі розробки програмного модуля класифікації

На основі проведеного в попередніх підрозділах аналізу сучасних методів комп'ютерного зору, архітектур згорткових нейронних мереж та апаратних обмежень вбудованих обчислювальних систем, встановлено наявність суперечності між зростаючими вимогами до точності розпізнавання візуальних об'єктів та жорсткими лімітами обчислювальних ресурсів кінцевих пристроїв класу Edge AI.

Використання базових, неоптимізованих моделей машинного навчання (наприклад, у форматах фреймворків TensorFlow або PyTorch) на платформі NVIDIA Jetson Nano призводить до неповного завантаження паралельної архітектури графічного процесора (GPU Maxwell), перевитрат оперативної пам'яті через неефективне керування тензорами та виникнення температурного тротлінгу

внаслідок надмірного енергоспоживання [15]. Це унеможливлює використання таких моделей для задач класифікації зображень у режимі жорсткого реального часу.

З огляду на зазначену проблематику, виникає науково-практична задача розробки спеціалізованого програмного модуля, який забезпечить ефективну інтеграцію обраної моделі згорткової нейронної мережі (CNN) в апаратне середовище NVIDIA Jetson Nano з максимізацією її швидкодії (FPS) та мінімізацією споживання ресурсів, за умови збереження прийнятного рівня точності класифікації.

Математично задачу розробки та оптимізації програмного модуля можна сформулювати як багатоцільову оптимізаційну задачу пошуку найкращого вектора параметрів моделі W∗ та конфігурації графа обчислень G∗, що мінімізують функціонал втрат L та час логічного висновку (інференсу) Tinf, за дотримання апаратних обмежень платформи [16]:

$$\min_{W^*,G^*} \left(L(Y, f(X, W^*, G^*)) \right) \qquad \min \left(T_{inf}(G^*) \right) \tag{1.3}$$

за наступних граничних умов:

$$Mem(G^*) \leq Mem_{max} \tag{1.4}$$

$$Pwr(G^*) \leq Pwr_{max} \tag{1.5}$$

$$Acc(W^*, G^*) \geq Acc_{min} \tag{1.6}$$

де: X — набір вхідних зображень;

Y — множина істинних міток класів (ground truth);

f(X,W∗,G∗) — прогнозовані мітки класів оптимізованою моделлю;

Mem_{max} — максимально доступний обсяг оперативної пам'яті (для Jetson Nano Mem_{max}=4 ГБ);

Pwr_{max} — максимальний ліміт тепловиділення/енергоспоживання платформи (TDP ≤10 Вт);

$Acc(W^*,G^*)$ — точність класифікації оптимізованої моделі на тестовій вибірці;
Acc_{min} — мінімально допустимий поріг точності, визначений технічним завданням (зазвичай допускається падіння не більше ніж на 1–2% відносно базової моделі FP32).

Для розв'язання поставленої оптимізаційної задачі (1.3)-(1.6) та досягнення мети кваліфікаційної роботи необхідно виконати наступні етапи розробки програмного модуля:

1. Вибір та адаптація базової архітектури CNN: Обґрунтувати вибір легкої моделі (наприклад, MobileNetV2 або EfficientNet), яка концептуально відповідає обмеженням (1.5) та (1.6). Здійснити переднавчання (transfer learning) моделі на цільовому наборі даних для досягнення умови (1.7).

2. Оптимізація графа обчислень (Graph Optimization): Використати інструментарій NVIDIA TensorRT для реструктуризації моделі: злиття операцій згортки, зміщення (bias) та функції активації (наприклад, ReLU) в єдиний вузол для мінімізації транзакцій до уніфікованої пам'яті платформи.

3. Квантування ваг моделі (Quantization): Здійснити конвертацію вагових коефіцієнтів $W*$ з 32-бітної арифметики з плаваючою комою (FP32) у 16-бітний формат (FP16), який апаратно підтримується GPU архітектури Maxwell на Jetson Nano, для прискорення обчислень T_{inf}.

4. Розробка програмного інтерфейсу (API): Створити програмну обгортку на базі мови Python (з використанням бібліотек OpenCV та TensorRT) для забезпечення безперервного захоплення відеопотоку з камери, попередньої обробки кадрів, передачі тензорів на GPU та візуалізації результатів класифікації.

Таким чином, розв'язання сформульованої задачі дозволить перетворити теоретичну нейромережеву модель у практично застосовний інженерний продукт, здатний автономно функціонувати в умовах граничних обчислень. Конкретні алгоритмічні та архітектурні рішення щодо реалізації зазначених етапів будуть докладно розглянуті в другому розділі даної роботи.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ТА АЛГОРИТМІВ КЛАСИФІКАЦІЇ ДЛЯ ПЛАТФОРМИ NVIDIA JETSON

2.1 Архітектурна модель програмного модуля та схема потоків даних

Розробка ефективного програмного забезпечення для вбудованих платформ комп'ютерного зору (Edge Vision) вимагає чіткого структурування компонентів системи та оптимізації процесів передачі даних між ними. Зважаючи на гетерогенну архітектуру цільової платформи NVIDIA Jetson Nano (багатоядерний CPU ARM Cortex-A57 та GPU Maxwell), програмна архітектура модуля повинна забезпечувати асинхронне виконання завдань вводу-виводу (захоплення відео) та інтенсивних математичних обчислень (інференс нейронної мережі).

Запропонована архітектурна модель програмного модуля базується на конвеєрному (pipeline) патерні проєктування. Такий підхід дозволяє розпаралелити процеси та максимізувати пропускну здатність системи (кількість кадрів за секунду, FPS) [17]. Логічну структуру програмного комплексу можна декомпозувати на три основні функціональні підсистеми:

1. Підсистема захоплення та попередньої обробки (Data Ingestion & Preprocessing): відповідає за отримання сирого відеопотоку з інтерфейсу камери (CSI або USB), декодування кадрів та їх підготовку (зміна розміру, нормалізація кольору) силами центрального процесора (CPU).
2. Підсистема логічного висновку (Inference Engine): ядро програмного модуля, що реалізує виконання оптимізованого графа згорткової нейронної мережі (у форматі TensorRT) на графічному процесорі (GPU).
3. Підсистема постобробки та візуалізації (Postprocessing & Output): забезпечує інтерпретацію вихідних тензорів, накладання результатів класифікації на вихідне зображення та виведення результату на дисплей або передачу по мережі.

Для формалізації процесів перетворення інформації в системі розроблено схему потоків даних (Data Flow Diagram, DFD), яку наведено на рисунку 2.1.

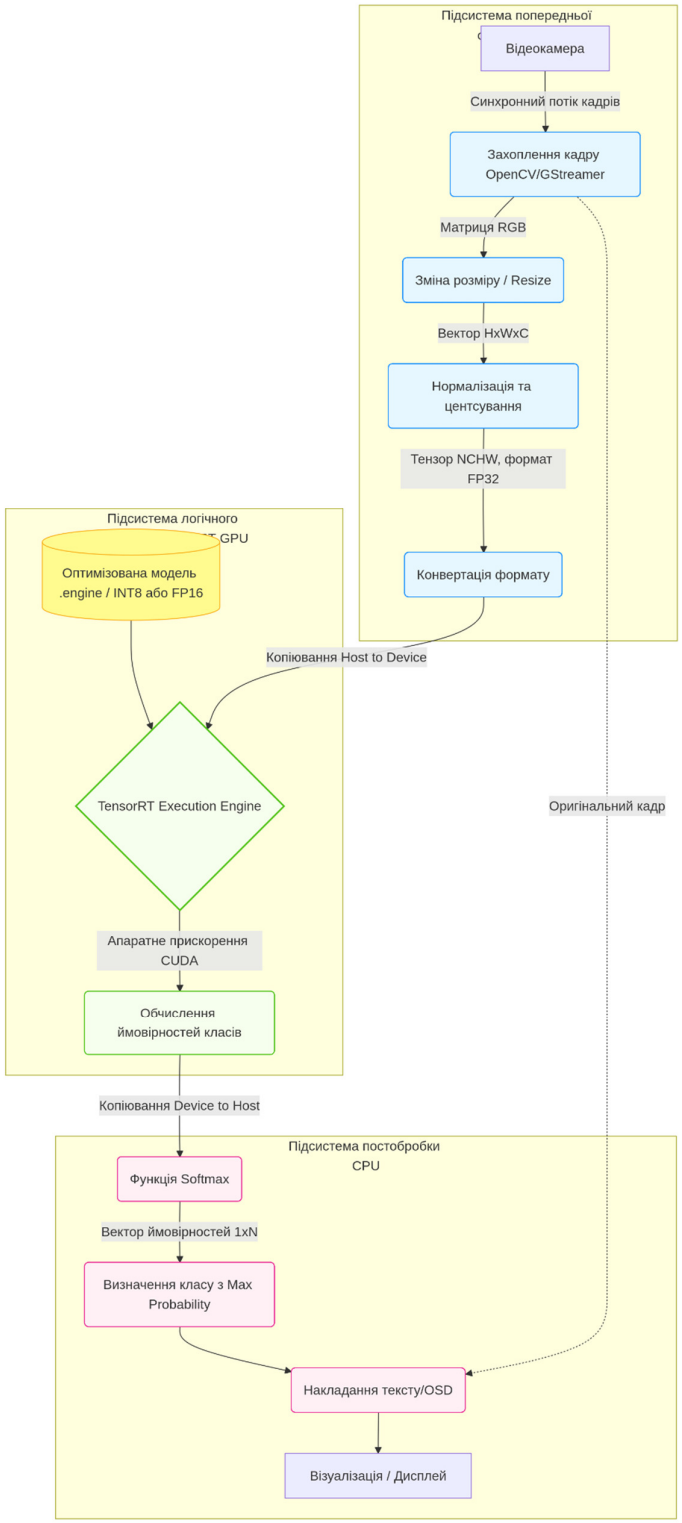

Рисунок 2.1 – Схема потоків даних програмного модуля класифікації на платформі Jetson

Важливим етапом у схемі потоків даних є трансформація формату тензора перед його подачею на GPU. Стандартні бібліотеки комп'ютерного зору (наприклад,

OpenCV) представляють зображення у форматі HWC (Height, Width, Channel — Висота, Ширина, Канал). Проте апаратні прискорювачі NVIDIA оптимізовані для обробки тензорів у форматі NCHW (Batch Size, Channel, Height, Width) [18]. Ця транскодація виконується на етапі E (рис. 2.1).

Математично процес нормалізації вхідного зображення (етап D) для приведення значень пікселів до діапазону [0,1] або [−1,1], що вимагається більшістю архітектур CNN (наприклад, MobileNetV2), можна описати наступним перетворенням для кожного кольорового каналу c∈{R,G,B}:

$$X_{norm}^{(c)}(i,j) = \frac{X^{(c)}(i,j) - \mu^{(c)}}{\sigma^{(c)}} \tag{2.1}$$

де: $X^{(c)}(i,j)$ — інтенсивність пікселя у координатах (i,j) для каналу c;

$\mu^{(c)}$ — середнє значення інтенсивності пікселів для каналу c (обчислене на всьому навчальному наборі даних, наприклад, ImageNet);

$\sigma^{(c)}$ — стандартне відхилення для каналу c.

Після виконання логічного висновку ядром TensorRT (етап F), мережа повертає вектор сирих оцінок (логітів) $Z=[z1, z2, ..., z_K]$, де K — загальна кількість класів. Для перетворення логітів у зрозумілий вектор ймовірностей $Y=[y1,y2,...,y_K]$ підсистема постобробки використовує нелінійну активаційну функцію Softmax:

$$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \qquad k = 1, \ldots, K \tag{2.2}$$

Кінцевий результат класифікації C_{pred} визначається як індекс класу, що має максимальну ймовірність:

$$C_{pred} = \arg \max_{k \in \{1,\ldots,K\}} (y_k) \tag{2.3}$$

Запропонована архітектурна модель дозволяє чітко розмежувати задачі між

CPU та GPU. Центральний процесор ARM виконує операції підготовки даних (формула 2.1) та парсингу результатів (формули 2.2, 2.3), тоді як графічний прискорювач Maxwell монопольно використовується для виконання інтенсивних матричних множень у згорткових шарах. Такий розподіл навантаження є критично важливим для уникнення ефекту «пляшкового горлечка» (bottleneck) на слабких ядрах Cortex-A57 пристрою Jetson Nano [19].

2.2 Вибір та обґрунтування базової архітектури згорткової нейронної мережі

Вибір архітектури згорткової нейронної мережі (ЗНМ) є критичним етапом проєктування програмного модуля для платформи NVIDIA Jetson Nano. Оскільки цільовий пристрій має жорсткі обмеження щодо обсягу спільної оперативної пам’яті (4 ГБ LPDDR4) та обчислювальної потужності (GPU Maxwell, 0.47 TFLOPS), використання класичних глибоких архітектур, таких як VGG-16 або ResNet-50, є недоцільним через високу затримку (latency) та надмірне споживання енергії [20, с. 145].

Для забезпечення роботи системи комп'ютерного зору в режимі реального часу необхідно розв'язати компромісну задачу (trade-off) між точністю класифікації (Accuracy) та обчислювальною складністю моделі, яка вимірюється кількістю операцій множення-додавання (Multiply-Accumulate Operations, MACs).

Враховуючи специфіку задачі, у якості кандидатів на роль базової архітектури (backbone) було розглянуто сучасні легковагові моделі: MobileNetV1, MobileNetV2 та EfficientNet-B0.

Основою для мінімізації обчислювального навантаження у цих архітектурах є використання глибинно-сепарабельних згорток (Depthwise Separable Convolutions). Використання ядра 3×3 дозволяє зменшити обчислювальну складність шару приблизно у 8–9 разів із мінімальною втратою репрезентативної здатності ознак [9].

Проте, для максимізації ефективності саме на архітектурі GPU, розробниками моделі MobileNetV2 було впроваджено дві критичні інновації: лінійні вузькі місця (Linear Bottlenecks) та інвертовані залишкові зв’язки (Inverted Residuals). Ця структура (блок розширення-згортки-стиснення) ідеально підходить для оптимізації

через TensorRT, оскільки зменшує потребу у пересиланні великих проміжних тензорів до основної пам'яті (уникаючи ефекту Memory Bottleneck) [21]. Логічну структуру інвертованого залишкового блоку наведено на рисунку 2.2.

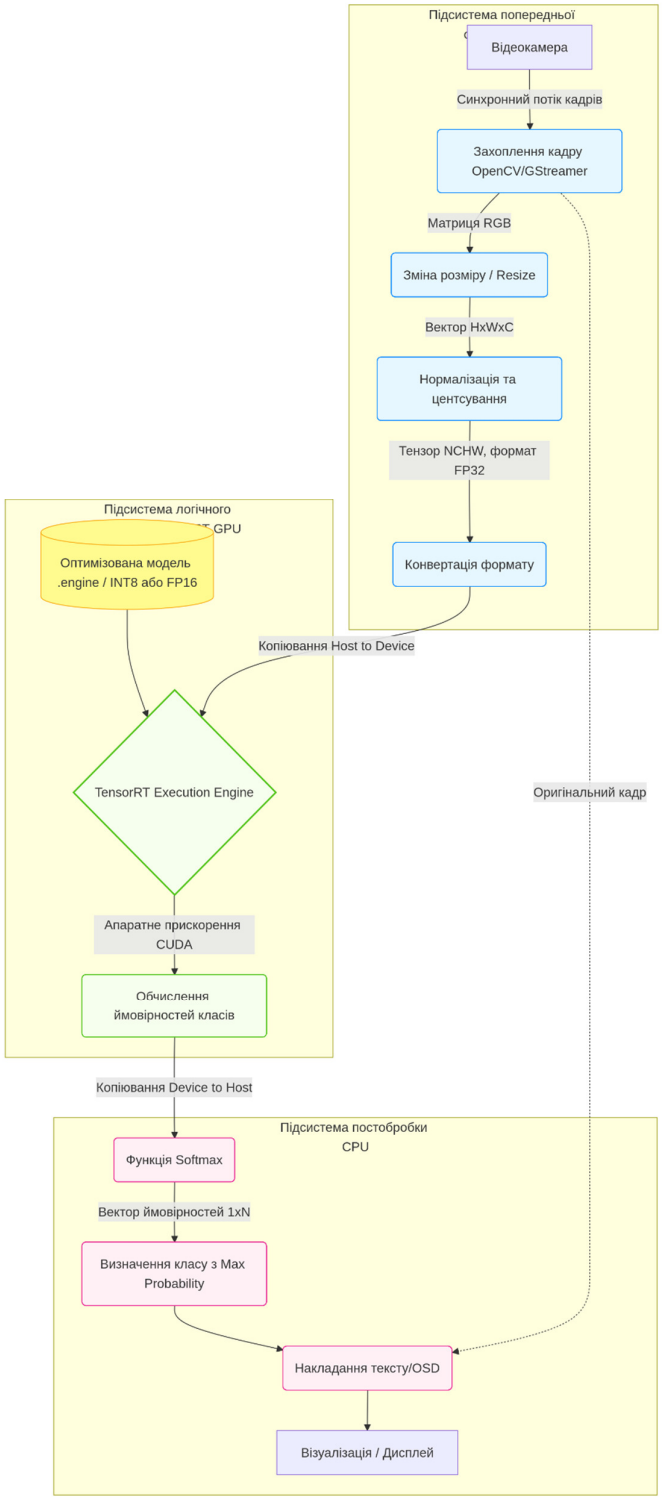

Рисунок 2.2 – Архітектура інвертованого залишкового блоку (Inverted Residual Block) моделі MobileNetV2

Використання функції активації ReLU6 (де максимальне значення обмежене цифрою 6) замість стандартної ReLU математично описується як:

$$f(x) = \min(\max(0, x), 6) \tag{2.4}$$

Вибір ReLU6 (2.4) є критично важливим для наступного етапу роботи (розділ 3), оскільки обмеження діапазону активацій суттєво підвищує точність і стабільність мережі при її квантуванні до 16-бітної арифметики (FP16), що є обов'язковим для платформи Jetson Nano [21].

Для остаточного обґрунтування вибору базової архітектури було проведено порівняльний аналіз продуктивності моделей на платформі NVIDIA Jetson Nano з використанням фреймворку TensorRT у форматі FP16. Результати зведено у таблицю 2.1.

Таблиця 2.1 — Порівняльний аналіз швидкодії архітектур ЗНМ на платформі NVIDIA Jetson Nano (TensorRT, FP16)

Архітектура (Backbone)	Розмір моделі (МБ)	Кількість параметрів	Точність (ImageNet Top-1)	Швидкодія на Jetson Nano (FPS)	Затримка (Latency, мс)
ResNet-50	~98.0	25.6 млн	76.0%	~18	55.5
MobileNetV1	~17.0	4.2 млн	70.6%	~75	13.3
MobileNetV2	~14.0	3.4 млн	72.0%	~85	11.7
EfficientNet-B0	~21.0	5.3 млн	77.1%	~42	23.8

Аналіз даних таблиці 2.1 та рисунка 2.3 свідчить, що MobileNetV2 забезпечує найвищий показник кадрової частоти (~85 FPS) при мінімальній затримці інференсу (11.7 мс) серед усіх розглянутих архітектур. Модель EfficientNet-B0 хоч і демонструє вищу на 5.1% точність, проте її швидкодія є вдвічі нижчою, що збільшує ризики тротлінгу при тривалій роботі пристрою.

Отже, базовою архітектурою програмного модуля обрано MobileNetV2. Переднавчана (pre-trained) на наборі даних ImageNet вагова матриця цієї моделі буде

використана як відправна точка (технологія Transfer Learning) для адаптації під конкретну прикладну задачу класифікації, що детально розглядається у наступних підрозділах.

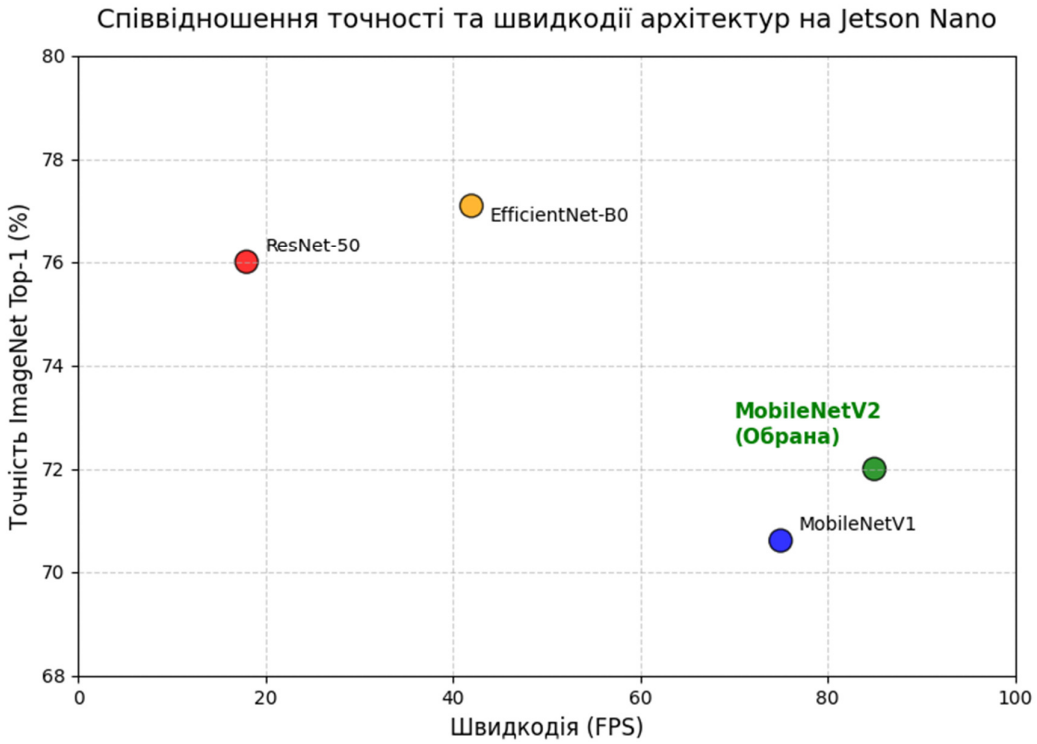

Рисунок 2.3 – Співвідношення точності та швидкодії (FPS) різних архітектур на платформі Jetson Nano

2.3 Алгоритми оптимізації моделі нейронної мережі для апаратних платформ

Вибір легковагової архітектури (MobileNetV2), обґрунтований у попередньому підрозділі, вирішує проблему обчислювальної складності лише на макрорівні. Для досягнення максимальної кадрової частоти (FPS) під час логічного висновку (інференсу) на цільовій платформі NVIDIA Jetson Nano необхідна мікрорівнева оптимізація графа обчислень та вагових коефіцієнтів навченої моделі.

Оскільки графічний процесор архітектури Maxwell використовує спільну з CPU оперативну пам'ять (LPDDR4), основною причиною затримок є не стільки

виконання самих арифметичних операцій, скільки постійні звернення до пам'яті для читання ваг та запису проміжних тензорів (карт ознак). Для вирішення цієї проблеми у розроблюваному програмному модулі інтегровано інструментарій NVIDIA TensorRT, який здійснює дві фундаментальні оптимізації: злиття шарів графа (Layer Fusion/Folding) та зниження розрядності обчислень (Quantization).

Злиття шарів графа обчислень (Layer Folding) У стандартних фреймворках (TensorFlow, PyTorch) операції згортки (Convolution), нормалізації за батчем (Batch Normalization, BN) та активації (наприклад, ReLU) реалізуються як окремі вузли графа. Це означає, що після кожної операції дані записуються в пам'ять, а потім знову зчитуються для наступної.

Алгоритм оптимізації TensorRT здійснює вертикальне злиття (Vertical Layer Fusion) цих трьох послідовних операцій в одне математичне ядро, що виконується на GPU за один такт звернення до пам'яті [23]. Логіку цієї оптимізації, а саме перехід від стандартного до злитого графа, зображено на рисунках 2.4 та 2.5.

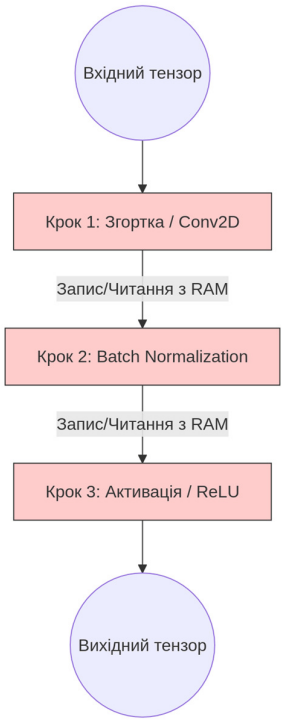

Рисунок 2.4 – Неоптимізований граф обчислень у стандартних фреймворках (TensorFlow/PyTorch)

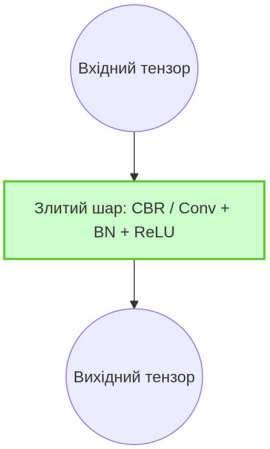

Рисунок 2.5 – Оптимізований граф обчислень після вертикального злиття шарів (NVIDIA TensorRT)

Математично операцію злиття шару згортки та шару Batch Normalization можна описати як перерахунок вихідних вагових коефіцієнтів. Стандартна операція BN над виходом згортки x має вигляд:

$$BN(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.5}$$

де μ — математичне сподівання,

σ^2 — дисперсія,

γ,β — параметри масштабування та зсуву,

ϵ — константа для запобігання діленню на нуль.

Знаючи, що вихід згортки x=W·I+B (де W — ваги, I — вхідний тензор, B — зміщення), TensorRT до початку інференсу аналітично обчислює нові «злиті» ваги (W_{fold}) та зміщення (B_{fold}) для єдиного згорткового ядра за формулами:

$$W_{fold} = \frac{\gamma \cdot W}{\sqrt{\sigma^2 + \epsilon}} \tag{2.6}$$

$$B_{fold} = \gamma \frac{B - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \qquad (2.7)$$

Таке математичне згортання (folding) на етапі ініціалізації модуля повністю виключає операцію Batch Normalization з процесу реального часу, заощаджуючи до 30% обчислювального часу шару [22].

Зниження розрядності вагових коефіцієнтів (Квантування) Другим критичним алгоритмом оптимізації є квантування (Quantization). Навчена на сервері модель MobileNetV2 зберігає свої параметри у форматі 32-бітних чисел із плаваючою комою (FP32). Платформа Jetson Nano з архітектурою Maxwell містить 128 CUDA-ядер, які апаратно підтримують виконання операцій у форматі 16-бітної арифметики (FP16/Half-Precision) із подвоєною пропускною здатністю порівняно з FP32.

Процес квантування з FP32 у FP16 є детермінованим і не вимагає додаткового калібрування на датасеті (на відміну від INT8). Звуження динамічного діапазону та втрата точності в мантисі числа для формату FP16 практично не впливають на підсумкову точність згорткових мереж (падіння точності становить менше 0.1%), оскільки вилучається лише надлишковий інформаційний шум [24].

Переваги переходу до квантованого графа обчислень наведено у таблиці 2.2.

Таблиця 2.2 — Порівняння форматів представлення даних при оптимізації моделі на GPU Maxwell [25]

Характеристика	FP32 (Одинарна точність)	FP16 (Половинна точність)	Зміна (ефект оптимізації)
Розмір в пам'яті (на 1 параметр)	32 біти (4 байти)	16 біт (2 байти)	Зменшення у 2 рази
Динамічний діапазон	pm 3.4 \times 10^{38}	\pm 65504$	Обмежений, достатній для CNN
Завантаження шини RAM	100% (базове)	~50%	Усунення ефекту Bottleneck
Апаратна підтримка Jetson Nano	Так (стандарт)	Так (нативна швидкість x2)	Підвищення швидкодії до 2 разів

Таким чином, у розроблюваному програмному модулі реалізовано алгоритм перетворення графа обчислень з формату .onnx (або .pb) у бінарний оптимізований формат рушія (TensorRT Engine .trt або .engine) з активацією прапорця fp16_mode=True. Використання алгоритмів Layer Folding та FP16 Quantization дозволяє в повній мірі розкрити потенціал апаратної платформи Jetson Nano, забезпечуючи виконання інференсу моделі MobileNetV2 зі швидкістю понад 60-80 кадрів за секунду, що повністю задовольняє вимоги до систем комп'ютерного зору реального часу.

Ефективність функціонування будь-якої системи комп'ютерного зору на базі згорткових нейронних мереж (ЗНМ) фундаментально залежить від якості, обсягу та репрезентативності набору навчальних даних (датасету). Згідно з парадигмою дата-центричного штучного інтелекту (Data-centric AI), оптимізація інформаційного забезпечення часто дає більший приріст точності класифікації, ніж зміна архітектури самої моделі [26].

Оскільки цільовим завданням розроблюваного програмного модуля є класифікація об'єктів в умовах реального часу на платформі NVIDIA Jetson Nano, інформаційне забезпечення системи проєктується з урахуванням концепції трансферного навчання (Transfer Learning). Базова архітектура MobileNetV2, обрана у підрозділі 2.2, попередньо навчена (pre-trained) на глобальном наборі даних ImageNet (який містить понад 1.2 млн зображень та 1000 класів). Це дозволяє мережі сформувати універсальні екстрактори візуальних ознак (контури, текстури, градієнти) на ранніх шарах згортки [27].

Загальна множина зібраних візуальних даних D математично поділяється на три непересічні підмножини: навчальну (D_{train}), валідаційну (D_{val}) та тестову (D_{test}):

$$D = D_{train} \cup D_{val} \cup D_{test}, \quad D_{train} \cap D_{val} \cap D_{test} = \emptyset \tag{2.8}$$

Розподіл даних здійснюється за емпіричним правилом Парето у пропорції 80/10/10 для забезпечення об'єктивної оцінки узагальнювальної здатності моделі. Структуру розподілу інформаційного забезпечення наведено у таблиці 2.3.

Таблиця 2.3 — Розподіл набору візуальних даних для навчання та тестування моделі

Підмножина датасету	Частка від загального обсягу	Основне функціональне призначення у конвеєрі машинного навчання
Навчальна	80%	Пряме оновлення вагових коефіцієнтів моделі (алгоритм зворотного поширення помилки)
Валідаційна	10%	Підбір гіперпараметрів, контроль перенавчання (Early Stopping) під час епох
Тестова	10%	Фінальна (незалежна) оцінка точності перед розгортанням на Jetson Nano

Зважаючи на те, що збір великих обсягів спеціалізованих даних у реальних умовах часто є ресурсомістким завданням, у роботі застосовується метод штучного розширення навчальної вибірки — аугментація даних (Data Augmentation) [28]. Цей процес виконується виключно на етапі навчання моделі (на високопродуктивній серверній станції) і не навантажує платформу Jetson Nano під час інференсу.

Алгоритми аугментації базуються на застосуванні випадкових афінних перетворень до вхідних зображень. Математично двовимірне афінне перетворення піксельних координат (x,y) у нові координати (x′,y′) описується матричним множенням:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \cos\theta & -\sin\theta & t_x \\ \sin\theta & s_y \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.9)$$

де: θ — кут випадкового повороту (Rotation);

s_x,s_y — коефіцієнти масштабування (Scaling);

t_x,t_y — вектори лінійного зсуву (Translation).

Окрім геометричних перетворень, до конвеєра аугментації включено фотометричні спотворення (Color Jittering): випадкова зміна яскравості, контрастності, насиченості та колірного тону. Для умов роботи системи на

відкритому просторі важливо, оскільки реальний моніторинг об'єктів підпорядкований постійним змінам зовнішнього природного освітлення (глибокі тіні від суміжних конструкцій, пересвіти від прямого сонця, дифузне світло під час хмарності). Фотометрична аугментація дозволяє зробити просторові фільтри нейромережі інваріантними до колірної гами та яскравості текстури фону [29]. Логічну структуру конвеєра підготовки даних (Data Pipeline) наведено на рисунку 2.6.

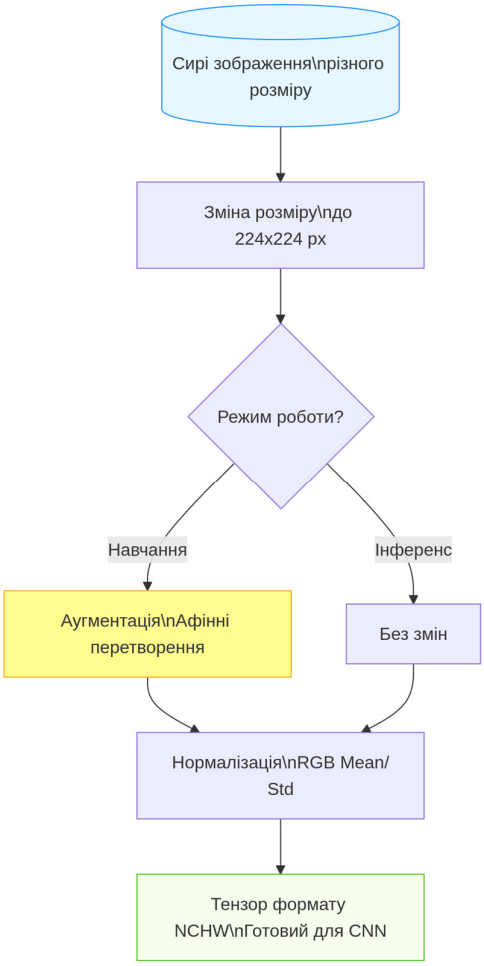

Рисунок 2.6 – Конвеєр попередньої обробки та аугментації інформаційного забезпечення

Після проходження всіх етапів попередньої обробки формуються стандартизовані пакети даних (mini-batches), які подаються на вхід нейронної мережі для оптимізації ваг. Правильно організований конвеєр інформаційного

забезпечення із фокусом на специфіку цільових об'єктів та умов зйомки є запорукою того, що конвертована для Jetson Nano модель TensorRT демонструватиме стабільно високу точність розпізнавання (F_1\text{-score}) у реальних фізичних умовах експлуатації, мінімізуючи кількість хибнонегативних результатів (пропусків цільових об'єктів).

Для забезпечення цілісного розуміння архітектури, взаємозв'язків та принципів роботи розроблюваного програмно-апаратного комплексу класифікації заданих класів об'єктів на базі платформи NVIDIA Jetson Nano, необхідно виконати його формалізацію на кількох рівнях абстракції. Системний аналіз та проєктування архітектури виконується шляхом декомпозиції на функціональну, структурну (компонентну) та поведінкову моделі.

Функціональна модель визначає логічну послідовність завдань, які виконує система для досягнення кінцевої мети — безперервного аналізу візуальної інформації та виявлення цільових об'єктів, незалежно від конкретної програмно-апаратної реалізації.

Логічну структуру інформаційних перетворень комплексу декомпоновано на п'ять основних функцій:

- F_1 (Захоплення візуальних даних): відповідає за безперервне зчитування сирого цифрованого відеопотоку з оптичного сенсора камери (через інтерфейси CSI або USB), його первинне декодування на рівні драйвера ОС та тимчасове буферизування кадрів у оперативній пам'яті.
- F_2 (Препроцесинг / Попередня обробка): реалізує геометричну та арифметичну підготовку даних. Включає зміну роздільної здатності кадру (Resize) під фіксований вхідний розмір нейромережі MobileNetV2 (224 \times 224 пікселі), конвертацію колірної моделі з OpenCV-стандарту BGR у RGB, транскодування формату тензора з HWC в NCHW та математичну нормалізацію яскравості пікселів.
- F_3 (Логічний висновок / Інференс): є обчислювальним ядром системи. Функція забезпечує передачу оптимізованого вхідного тензора в ізольовану область пам'яті GPU, виконання прямого поширення графа нейромережі за допомогою прискорених ядер NVIDIA TensorRT та формування вихідного вектора сирих оцінок (логітів).

- F_4 (Постобробка даних): здійснює інтерпретацію низькорівневих обчислювальних результатів GPU. На цьому етапі до логітів застосовується активаційна функція Softmax, обчислюються ймовірності для кожного з 4 класів, визначається фінальний прогнозований клас C_{pred} за критерієм Argmax та виконується фільтрація результатів за настроюваним порогом впевненості (Confidence Threshold).
- F_5 (Візуалізація та диспетчеризація): відповідає за формування зворотного зв'язку. Включає графічне накладання обмежувальних рамок (Bounding Boxes) або текстових міток із назвою прогнозованого класу та відсотком впевненості безпосередньо на кадр відеопотоку, виведення анотованого графічного вікна на дисплей оператора, а також генерацію тригерних сигналів для збереження результатів у разі фіксації об'єктів інтересу.

Послідовність взаємодії та передачі керування між функціями системи наведено на рисунку 2.7.

Структурна схема описує компонентний склад системи, визначаючи розподіл між апаратним та програмним забезпеченням, а також ієрархію внутрішніх інтерфейсів взаємодії.

Проєктований комплекс організовано за вертикально-інтегрованим принципом і розділено на такі системні рівні:

1. Апаратний рівень (Hardware): фізична основа системи на базі одноплатного комп'ютера NVIDIA Jetson Nano. Включає сенсорний блок (камера IMX219 MIPI CSI-2), обчислювальні ядра центрального процесора (ARM Cortex-A57) для загального керування, графічний прискорювач (Maxwell 128-Core) для паралельних матричних обчислень, уніфіковану оперативну пам'ять LPDDR4 об'ємом 4 ГБ та периферійні інтерфейси вводу-виводу (HDMI, Ethernet).

2. Рівень операційної системи та драйверів: базове системне ПЗ. Складається з дистрибутива Linux for Tegra (L4T) на базі ядра Ubuntu, спеціалізованого пакета драйверів Video4Linux2 (V4L2) для низькорівневої взаємодії з CSI-камерою та компонентів NVIDIA JetPack SDK.

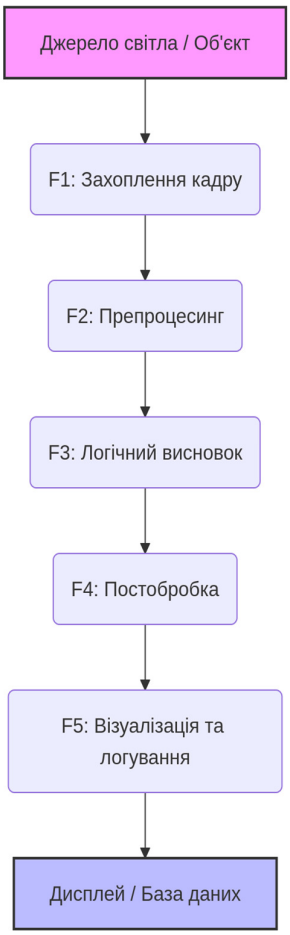

Рисунок 2.7 – Функціональна схема конвеєра обробки відеопотоку

3. Рівень бібліотек та фреймворків: проміжне ПЗ (Middleware), що надає API для розробки. Використовує бібліотеку OpenCV для оптимізованих операцій із матрицями зображень на CPU, платформу CUDA та cuDNN для низькорівневого прискорення обчислень, а також середовище NVIDIA TensorRT для оптимізації та виконання інференсу моделей.

4. Прикладний рівень (Software): верхній рівень архітектури, представлений кастомним програмним модулем на мові Python. Модуль містить три логічні субмодулі: захоплення потоку, керування інференсом TensorRT та модуль інтерактивного виведення результатів.

Ієрархічну структуру та взаємозв'язки рівнів програмно-апаратного комплексу наведено на рисунку 2.8.

Поведінкові аспекти системи, що визначають її функціональні можливості з точки зору кінцевого споживача, формалізовано за допомогою діаграми прецедентів

(Use Case Diagram). Головним актором системи виступає Оператор (або Користувач), який керує процесом аналізу відеопотоку.

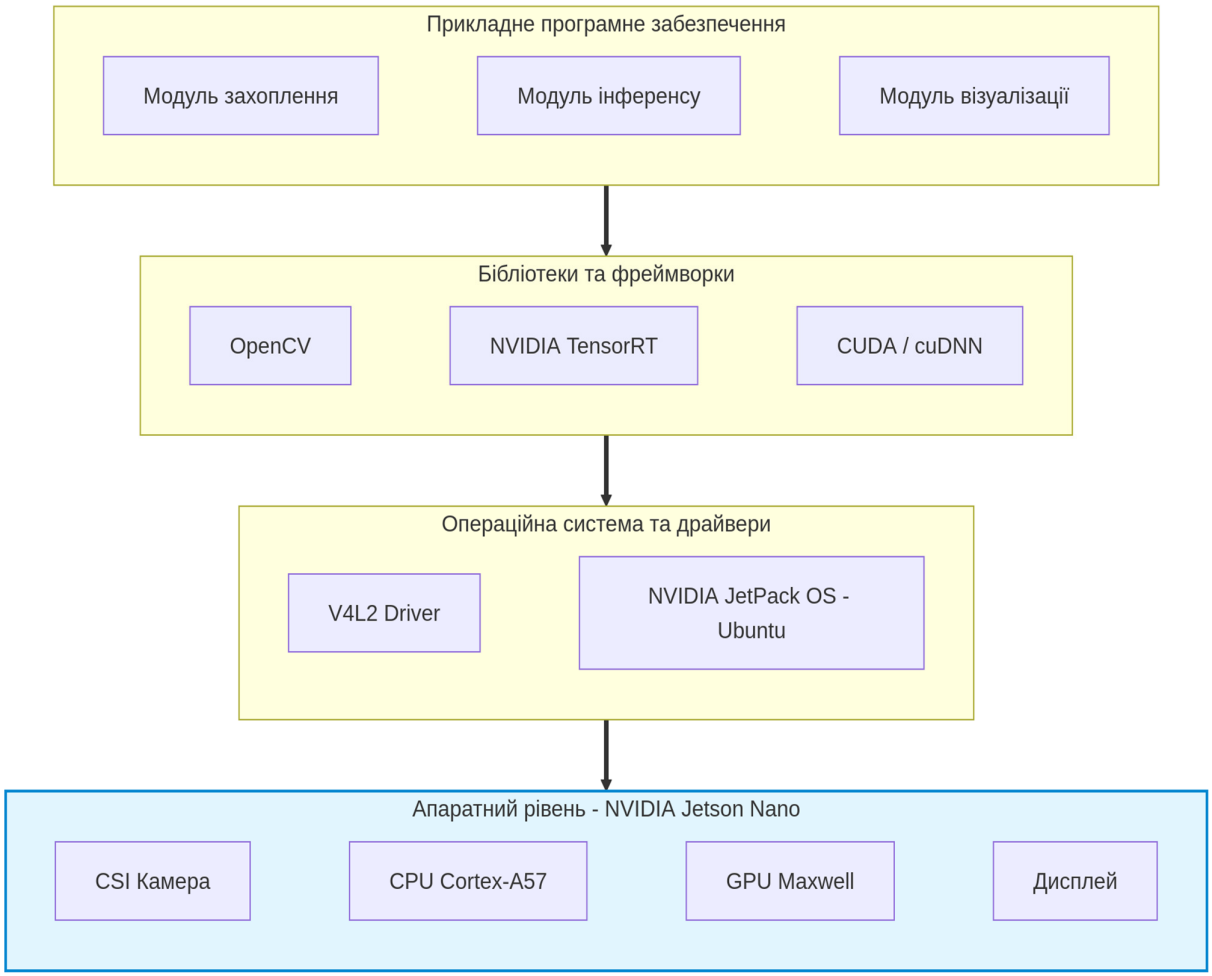

Рисунок 2.8 – Структурна схема програмно-апаратних рівнів системи

Модель прецедентів включає такі ключові сценарії взаємодії:

- запуск системи моніторингу: ініціює старт комплексу, при цьому обов'язковим кроком за відношенням розширення <<include>> є автоматичне завантаження та десеріалізація файлу оптимізованої моделі model_fp16.engine у виділену область пам'яті GPU.
- налаштування параметрів: дозволяє оператору гнучко конфігурувати критерії детекції (задавати мінімальний поріг впевненості мережі для відсікання хибнопозитивних спрацювань).
- перегляд відеопотоку: базовий режим роботи, що забезпечує

відображення поточного стану досліджуваної сцени із динамічним накладанням маркерів розпізнаних класів.

– збереження класифікованого кадру (Логування): виконується автоматично при фіксації об'єкта інтересу або ініціюється оператором вручну. Даний прецедент за відношенням <> може бути розширений функцією експорту звіту для реєстрації результатів розпізнавання у звітній документації.

Діаграму прецедентів взаємодії користувача з розроблюваним модулем наведено на рисунку 2.9.

Рисунок 2.9 – Діаграма прецедентів (UML Use Case) взаємодії з комплексом моніторингу

Для деталізації внутрішніх інформаційних інтерфейсів, визначення точок трансформації даних, взаємодії із файловою системою та зовнішніми пристроями розроблено діаграму потоків даних (Data Flow Diagram, DFD) 1-го рівня. На відміну від концептуальної схеми, DFD рівня 1 чітко розмежовує інформаційні сутності, сховища даних та дискретні процеси обробки.

Деталізовану схему циркуляції та перетворення інформаційних потоків у системі наведено на рисунку 2.10.

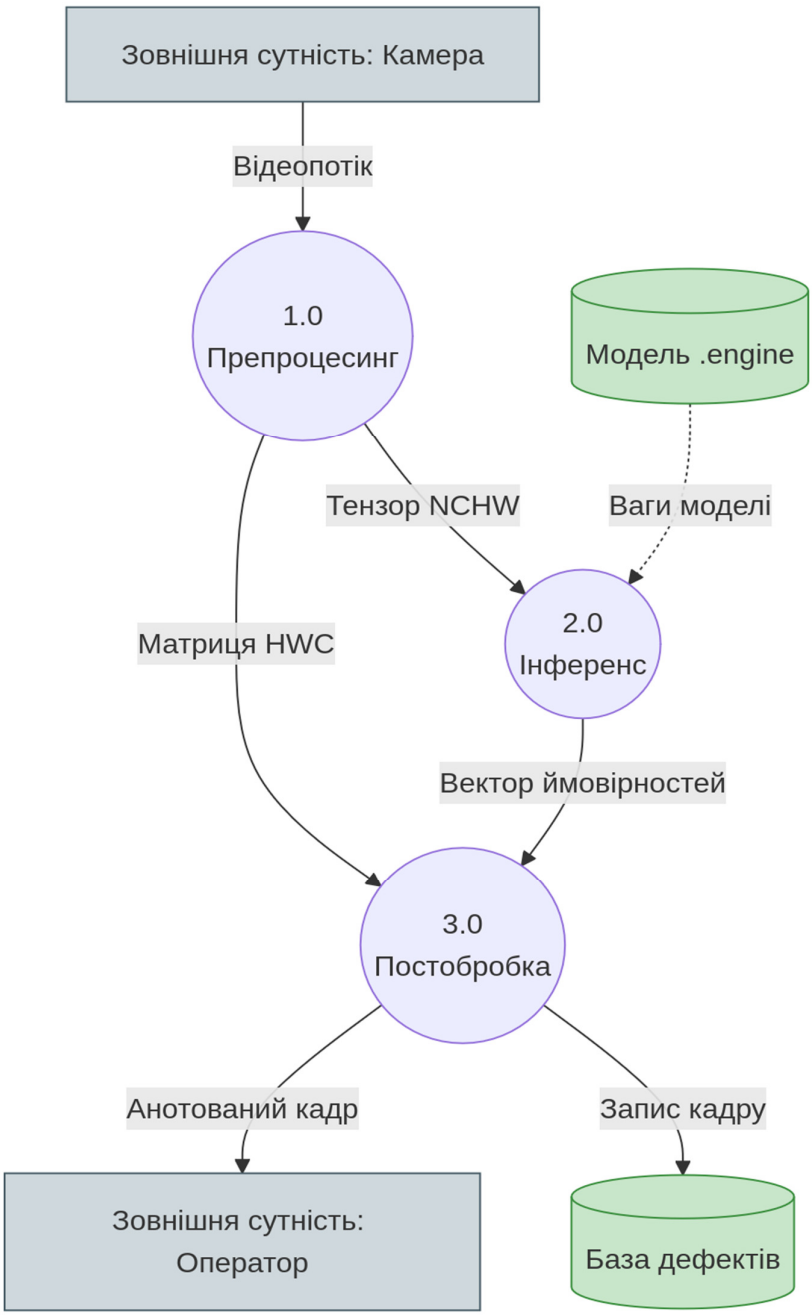

Рисунок 2.10 – Деталізована діаграма потоків даних (DFD Рівень 1) програмного модуля

Архітектура потоків даних містить дві зовнішні сутності (Камера як джерело даних, Оператор як одержувач інформації), два сховища даних та п'ять послідовних

процесів:

- Процес 1.0 (Читання): забезпечує демультиплексування необробленого байтового потоку, що надходить від сутності Камера, та його перетворення у структуру даних оперативної пам'яті — Матрицю пікселів HWC.
- Процес 2.0 (Форматування): здійснює препроцесинг. Отримує Матрицю пікселів HWC, виконує нормалізацію за формулою (2.1), змінює порядок осей тензора та передає на наступний етап Тензор NCHW (FP32).
- Процес 3.0 (Класифікація): відповідає за виконання логічного висновку. Процес зчитує оптимізовані вагові коефіцієнти та топологію графа із системного файлу конфігурації (Сховище моделі .engine), здійснює швидкісні матричні обчислення на CUDA-ядрах GPU та генерує Вектор ймовірностей (Логіти).
- Процес 4.0 (Агрегація результату): поєднує первинну геометричну та фінальну аналітичну інформацію. Отримує Вектор ймовірностей від процесу 3.0 та оригінальну Матрицю пікселів HWC від процесу 1.0, обчислює фінальний клас за формулами (2.2, 2.3) та формує Анотований кадр (зображення із накладеною графікою).
- Процес 5.0 (Виведення та Збереження): керує розподілом вихідних потоків. Синхронно транслює Анотований кадр на пристрій відображення сутності Оператор, а у разі детектування цільового класу виконує Запис кадру з розпізнаним об'єктом у довготривалу енергонезалежну пам'ять (База збережених подій).

3 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМНОГО МОДУЛЯ НА NVIDIA JETSON

3.1 Вибір інструментальних засобів та програмна реалізація модуля

Практична реалізація спроєктованої архітектури системи класифікації зображень для вбудованої платформи NVIDIA Jetson Nano вимагає обґрунтованого вибору стека програмних технологій. Критичними вимогами до інструментальних засобів є: нативна підтримка апаратної архітектури GPU NVIDIA Maxwell, можливість низькорівневої оптимізації обчислень та наявність розвинених бібліотек для роботи з візуальними даними у режимі реального часу [30].

Базовим системним середовищем для платформи обрано NVIDIA JetPack SDK (версії 4.6.x, що є останньою підтримуваною для Jetson Nano) [31]. Цей комплект розробника базується на операційній системі Ubuntu 18.04 LTS for ARM64 і містить необхідні драйвери (L4T — Linux for Tegra), бібліотеки паралельних обчислень CUDA Toolkit, бібліотеку прискорення глибоких нейромереж cuDNN та, що найважливіше, рушій інференсу TensorRT [32]. Використання JetPack забезпечує прямий доступ програмного забезпечення до апаратних ресурсів SoC Tegra X1.

Як основну мову програмування для реалізації модуля обрано Python 3.6+. Цей вибір зумовлений його позицією як де-факто стандарту в галузі машинного навчання, наявністю високопродуктивних обгорток (wrappers) для бібліотек C++ (OpenCV, TensorRT) та швидкістю прототипування складних систем.

Технологічний процес розробки модуля розділено на два етапи, що використовують різні фреймворки:

1. Етап навчання (на високопродуктивному сервері/ПК): Для реалізації та навчання базової моделі MobileNetV2 (обраної в п. 2.2) використано високорівневий API Keras на базі фреймворку TensorFlow 2. Це дозволило використати попередньо навчені ваги ImageNet та ефективно провести процес донавчання (fine-tuning) на цільовому датасеті з використанням аугментації.

2. Етап інференсу (на NVIDIA Jetson Nano): Для виконання навченої моделі на кінцевому пристрої використано бібліотеку NVIDIA TensorRT. Як було обґрунтовано в п. 2.3, пряме виконання TensorFlow-моделі на Jetson Nano є

неефективним. Тому реалізовано конвертацію моделі за маршрутом: Keras (.h5) -> ONNX (.onnx) -> TensorRT Engine (.trt). Саме TensorRT забезпечує злиття шарів та виконання прискорених обчислень у форматі половинної точності (FP16) на ядрах CUDA архітектури Maxwell.

Для роботи з відеопотоком та попередньої обробки зображень використано бібліотеку комп'ютерного зору OpenCV (Open Source Computer Vision Library) з підтримкою GStreamer [32]. Це критично важливо для ефективного захоплення кадрів з камери через інтерфейс MIPI CSI-2, який використовується на платі Jetson Nano, минаючи зайві копіювання даних у пам'яті CPU [33].

Зведену характеристику обраного стека технологій наведено у таблиці 3.1.

Таблиця 3.1 — Обґрунтування вибору інструментальних засобів програмної реалізації

Категорія засобу	Обраний інструмент / Бібліотека	Обґрунтування вибору для платформи NVIDIA Jetson Nano
Операційна система	NVIDIA JetPack SDK (Ubuntu 18.04 ARM64)	Єдине офіційне середовище, що містить драйвери L4T, CUDA, cuDNN для повного доступу до апаратного забезпечення Tegra X1.
Мова програмування	Python 3.6+	Стандарт індустрії ML/AI; наявність ефективних інтерфейсів до низькорівневих бібліотек C/C++.
Фреймворк навчання	TensorFlow 2.x / Keras	Зручний API для роботи з архітектурою MobileNetV2 та реалізації трансферного навчання.
Рушій інференсу	NVIDIA TensorRT	Ключовий компонент для оптимізації (FP16, Layer Fusion) та прискорення виконання на GPU Maxwell. Забезпечує максимальний FPS.
Обробка зображень	OpenCV (з підтримкою GStreamer/CUDA)	Ефективне захоплення відео з CSI-камери, швидкі операції зміни розміру та нормалізації матриць зображень.
Чисельні операції	NumPy	Високопродуктивні операції з багатовимірними масивами (тензорами) на CPU перед передачею на GPU.

Програмна реалізація модуля виконана з дотриманням принципів об'єктно-орієнтованого програмування (ООП) для забезпечення модульності та розширюваності коду. Структуру розробленого програмного забезпечення представлено у вигляді UML-діаграми класів на рисунку 3.1.

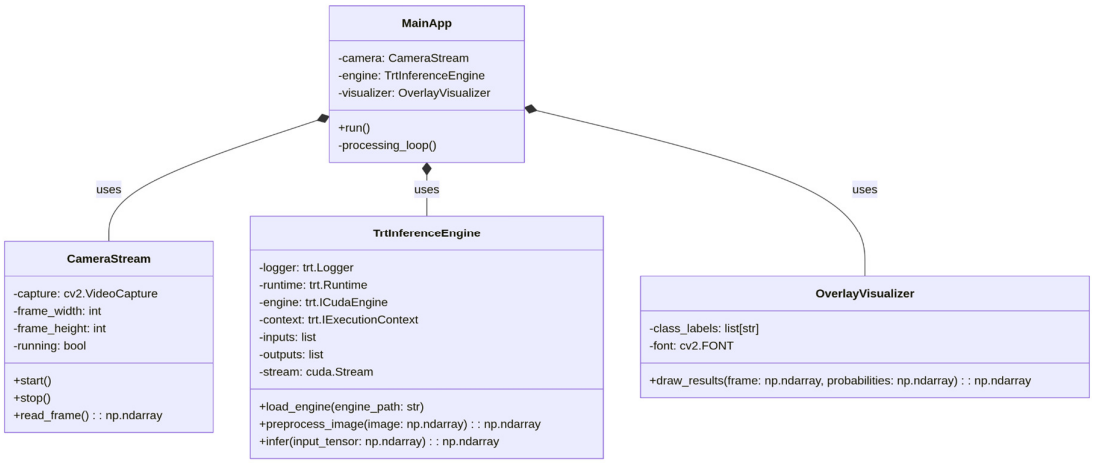

Рисунок 3.1 – UML-діаграма класів програмного модуля

Центральний клас MainApp координує роботу компонентів, реалізуючи основний цикл обробки «захоплення-інференс-візуалізація». Клас CameraStream інкапсулює логіку роботи з OpenCV для асинхронного зчитування кадрів. Ключовий клас TrtInferenceEngine відповідає за завантаження оптимізованого .trt файлу, виділення пам'яті на GPU (CUDA device memory), асинхронне копіювання тензорів між хостом та девайсом, та виконання інференсу через TensorRT контекст. Клас OverlayVisualizer відповідає за інтерпретацію вихідного вектора ймовірностей та накладання текстової інформації на кадр.

Така архітектура дозволяє ізолювати специфічну для платформи Jetson логіку (робота з CUDA/TensorRT) всередині окремого класу, полегшуючи налагодження та супровід програмного продукту.

3.2 Навчання моделі та її адаптація для апаратного прискорення

Ефективність функціонування будь-якого програмного модуля комп'ютерного зору на базі глибоких згорткових нейронних мереж безпосередньо обмежується

якістю та структурою вхідного інформаційного забезпечення. У межах розроблюваного комплексу інформаційною основою є спеціалізований відкритий набір даних «2022 Ukraine Russia War Equipment Losses Oryx» [34], що містить верифіковані фотоматеріали знищеної та захопленої військової техніки. Оскільки оригінальний масив даних є надмірно надлишковим для задач локального Edge-інференсу (понад 30 000 унікальних зображень загальним обсягом близько 10 ГБ), пряме його використання призвело б до критичних часових та обчислювальних витрат на етапі навчання.

Для розв'язання проблеми апаратних обмежень обчислювального вузла та оптимізації дискового простору на першому етапі розробки було реалізовано ізольований препроцесинговий програмний модуль автоматизованого парсингу та селективної фільтрації даних за допомогою офіційного API kagglehub. Повний лістинг цього модуля, який виконує виключно завантаження, очищення та відсікання зайвих зображень, наведено у додатку Б.

Інженерна логіка роботи препроцесингового скрипта базується на детермінованому відборі та агрегації сирих даних. Скрипт сканує дерево каталогів завантаженого репозиторію і здійснює трансляцію (мапінг) множини специфічних вихідних папок у чотири візуально ортогональні класи згідно з препроцесинговим алгоритмом відображення:

$$\mathcal{M} : \{S_1, S_2, \ldots, S_m\} \to C_k, \quad k \in \{1, 2, 3, 4\} \tag{3.1}$$

де S — простір оригінальних папок Oryx;

C — сформовані цільові класи.

Конкретна програмна реалізація мапінгу в модулі конвеєра підготовки даних описується наступними правилами:

- клас tanks (Індекс 0) формується шляхом прямого вилучення файлів з оригінального каталогу Tanks;
- клас afv_ifv (Індекс 1) реалізує комплексне структурне злиття та об'єднання інформаційних потоків із п'яти суміжних оригінальних папок:

Infantry_Fighting_Vehicles, Armoured_Personnel_Carriers, Armoured_Fighting_Vehicles, Infantry_Mobility_Vehicles та Mine-Resistant_Ambush_Protected;

- клас artillery (Індекс 2) об'єднує фотоматеріали колісної та гусеничної зброї з трьох каталогів: Self-Propelled_Artillery, Multiple_Rocket_Launchers та Towed_Artillery;
- клас trucks (Індекс 3) акумулює зображення неброньованої логістичної техніки та командно-штабних машин з оригінальної папки Trucks,_Vehicles,_and_Jeeps.

Для забезпечення стійкості процесу оптимізації та виключення дисбалансу класів (class imbalance), препроцесинговий скрипт здійснює стохастичне перемішування (шляхом фіксації генератора псевдовипадкових чисел random.shuffle) та лімітує вибірку, виділяючи рівно по 800 зображень для навчальної підмножини (D_train) та по 200 зображень для валідаційної підмножини (D_val) на кожен із 4 класів. Таке інженерне рішення дозволило скоротити обсяг інформаційного забезпечення до компактного збалансованого датасету об'ємом 4 000 кадрів (~350 МБ), повністю усуваючи потребу у локальному завантаженні та обробці 10 ГБ сирих даних, при збереженні високої репрезентативності вибірки. Сформована збалансована структура папок автоматично передається на вхід обчислювального конвеєра навчання.

На другому етапі сформована збалансована структура папок автоматично передається на вхід основного обчислювального конвеєра навчання згорткової нейронної мережі MobileNetV2, реалізованого мовою Python з використанням високорівневого API Keras (фреймворк TensorFlow). Повний лістинг програмного коду, що відповідає за аугментацію, двоетапне навчання моделі та її фінальний експорт в ONNX, наведено у додатку В.

Підготовка тренувальних даних здійснювалася з використанням генераторів зображень, що реалізують концепцію динамічної аугментації даних «на льоту» (on-the-fly) безпосередньо під час передачі пакетів (батчів) до оперативної пам'яті графічного прискорювача. Для підвищення інваріантності моделі до змін умов зйомки та запобігання явищу перенавчання (overfitting) до вхідних тензорів

застосовувалися стохастичні трансформації: випадкові афінні перетворення (повороти в діапазоні pm 25^\circ, лінійні зсуви по осях на pm 20\%, масштабування), горизонтальне віддзеркалення та фотометричні спотворення (корекція яскравості в межах pm 20\%). Перед подачею на вхід графа обчислень значення пікселів нормалізувалися відповідно до специфікації архітектури MobileNetV2, що передбачає приведення інтенсивності до лінійного діапазону [-1, 1].

Відповідно до парадигми трансферного навчання, базову архітектуру MobileNetV2 було ініціалізовано вагами, попередньо навченими на великомасштабному датасеті ImageNet. Для адаптації моделі до цільової задачі розпізнавання військової техніки оригінальний повнозв'язний класифікаційний блок (top layers) було вилучено. Натомість топологію мережі доповнено кастомним блоком, який включає:

- шар глобального усередненого пулінгу (Global Average Pooling 2D) — для просторового стиснення багатовимірних карт ознак розмірності H \times W \times C у компактний одновимірний вектор розмірності 1 \times 1 \times C, що значно зменшує кількість параметрів мережі порівняно зі стандартними шарами Flatten та запобігає просторовому перенавчанню;
- шар просторового проріджування (Dropout) — з емпірично підібраним коефіцієнтом відключення нейронів 0.2. Даний механізм виконує функцію стохастичної регуляризації під час прямого поширення, примусово обнуляючи частину активацій;
- вихідний повнозв'язний шар (Dense) — з функцією активації Softmax для перетворення вихідних низькорівневих оцінок (логітів) у формалізований розподіл ймовірностей приналежності об'єкта до одного з 4 цільових класів військової техніки.

Мінімізація функції втрат — категоріальної перехресної ентропії (Categorical Crossentropy) — здійснювалася алгоритмом адаптивної оцінки моментів (Adam). Процес оптимізації вагових коефіцієнтів було розділено на два послідовні етапи (стратегія Fine-tuning):

1. Навчання класифікатора (Feature Extraction): На першому етапі вагові коефіцієнти базових згорткових блоків MobileNetV2 були жорстко зафіксовані

(base_model.trainable = False). Алгоритм зворотного поширення помилки оновлював виключно параметри нових, доданих шарів із базовим кроком навчання (learning rate) рівним 10^{-3}. Це дозволило швидко налаштувати класифікатор на видобуті універсальні ознаки без ризику руйнування базових просторових фільтрів, сформованих на датасеті ImageNet.

2. Тонке налаштування (Fine-tuning): На другому етапі обмеження на оновлення ваг було знято шляхом повного розмороження графа обчислень (base_model.trainable = True). Навчання продовжувалося для всієї топології мережі, проте зі значно меншим кроком навчання (10^{-5}). Такий підхід забезпечив делікатне коригування як низькорівневих (контури, геометрія), так і високорівневих семантичних екстракторів ознак (наприклад, характерні форми гусеничних шасі, артилерійських стволів чи кабін вантажівок) під специфічні візуальні патерни цільового набору даних Oryx.

По завершенню оптимізаційного процесу навчені вагові коефіцієнти та топологія моделі були експортовані у сучасному нативному бінарному форматі Keras (.keras), який забезпечує цілісність збереження архітурного графа та станів оптимізатора. Наступним кроком став запуск підсистеми транскодування: за допомогою бібліотеки tf2onnx модель було конвертовано у проміжне кросплатформне представлення ONNX (*Open Neural Network Exchange*) з версією опсету 13 (opset=13).

Сформований апаратно-незалежний файл mobilenet_v2_custom.onnx оптимізує граф обчислень, усуваючи специфічні для TensorFlow вузли, і виступає уніфікованим вхідним артефактом для перенесення на цільову вбудовану платформу NVIDIA Jetson Nano, де рушій TensorRT компілює його у високоефективний бінарний файл інференсу .engine у режимі половинної точності FP16.

3.3 Розгортання програмного модуля на цільовій платформі та розробка інтерфейсу взаємодії

Наступним кроком після отримання оптимізованого бінарного файлу рушія TensorRT (формат .engine або .trt) є його безпосереднє розгортання в оперативному

середовищі вбудованої платформи NVIDIA Jetson Nano та організація безперервного циклу обробки візуальної інформації. Практична реалізація цього етапу вимагає попередньої низькорівневої підготовки носія даних та розгортання операційної системи.

Процедура підготовки апаратного комплексу розробника (Jetson Nano Developer Kit) та встановлення офіційного системного ПЗ складається з таких послідовних кроків:

1. Низькорівневе форматування: Для забезпечення максимальної пропускної здатності інтерфейсу введення-виведення використовується швидкісна карта пам'яті MicroSD об'ємом від 64 ГБ (класу UHS-I Class 10 або U3). Первинне очищення секторів та вирівнювання розділів виконується на персональному комп'ютері за допомогою офіційної утиліти *SD Card Formatter* від асоціації SD, що гарантує усунення зсувів файлової системи та сумісність із завантажувачем платформи.

2. Запис образу операційної системи: З офіційного веб-порталу NVIDIA Developer завантажується актуальний релізований образ системи — Jetson Nano Developer Kit SD Card Image (компонент збірки L4T — Linux for Tegra). Розархівований бінарний образ записується на підготовлений носій за допомогою спеціалізованого програмного забезпечення balenaEtcher (або Rufus). Ця утиліта виконує поблокове копіювання файлової системи Ext4 та здійснює фінальну верифікацію контрольних сум для виключення пошкодження системних файлів ядра Linux.

3. Апаратна ініціалізація: Прошита карта пам'яті MicroSD встановлюється у відповідний слот, розташований безпосередньо під знімним обчислювальним модулем (SoM) P3450 на базовій платі розробника (Carrier Board). До інтерфейсу MIPI CSI-2 підключається шлейф оптичного сенсора камери Sony IMX219 (Рисунок 3.2). Живлення системи конфігурується через роз'єм DC Jack (5V/4A) із встановленням замикаючого джампера J48, що необхідно для стабільної роботи графічного процесора на максимальних частотах без просідання напруги.

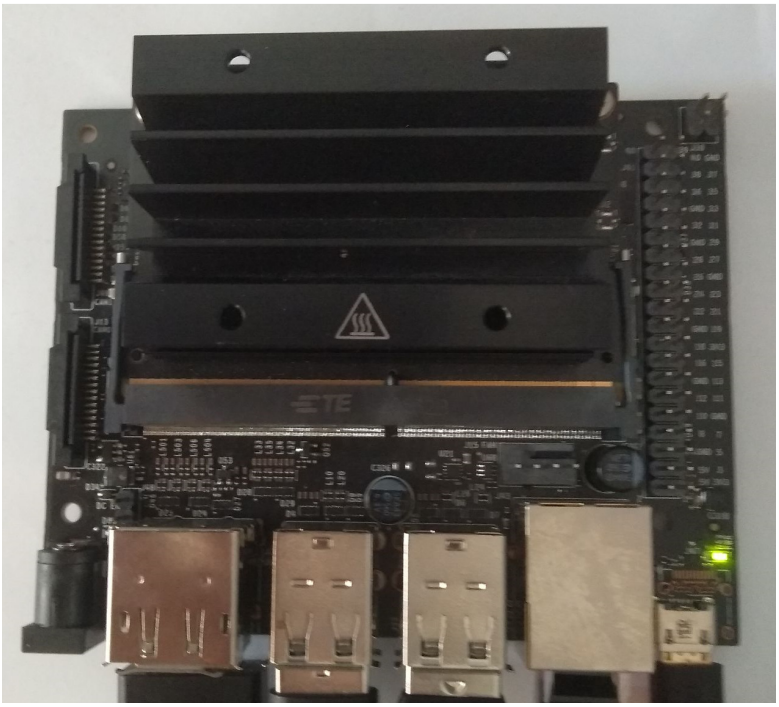

Рисунок 3.2 – Фотографія обчислювального комплексу Jetson Nano

Після успішного завантаження операційної системи розроблений програмний модуль класифікації функціонує за принципом інтерактивного конвеєра, що циклічно виконує захоплення, препроцесинг, інференс та візуалізацію даних. Зважаючи на архітектуру платформи (SoC Tegra X1), критично важливим аспектом розгортання є оптимізація обміну даними між CPU та GPU. У межах розробленого класу TrtInferenceEngine реалізовано механізм прямого доступу до уніфікованої пам'яті (Unified Memory Architecture, UMA). Процес ініціалізації інференсу передбачає наступні апаратні кроки:

- завантаження (десеріалізація) файлу оптимізованої моделі з енергонезалежної пам'яті MicroSD в RAM;
- створення контексту виконання (Execution Context) всередині середовища NVIDIA TensorRT;
- резервування сторінкових буферів пам'яті, захищених від вивантаження (Page-locked / Pinned Memory), на хості (CPU) та відповідних їм дзеркальних буферів у пам'яті пристрою (GPU Device Memory).

Математичний апарат логічного висновку виконується повністю асинхронно. Для цього створюється низькорівневий об'єкт керування cuda.Stream(), який

дозволяє центральному процесору ARM не блокуватися під час виконання матричних множень на ядрах Maxwell, а паралельно виконувати захоплення та підготовку наступного візуального кадру.

Інтерфейс взаємодії системи з навколишнім середовищем (захоплення відеопотоку) реалізовано за допомогою бібліотеки OpenCV з інтеграцією GStreamer-конвеєра. На відміну від стандартного V4L2-інтерфейсу, використання GStreamer дозволяє задіяти апаратний ISP (Image Signal Processor) процесора Tegra для апаратного декодування та конвертації колірного простору (з YUV у RGB/BGR) з нульовим навантаженням на ядра ARM Cortex-A57.

Головний цикл програми (Main Processing Loop) працює в режимі жорсткого реального часу і може бути описаний часовою умовою забезпечення мінімальної необхідної кадрової частоти. Для досягнення цільового промислового показника у 30 FPS сумарний час обробки одного кадру T_total не повинен перевищувати критичний поріг у 33.3 мс:

$$T_{total} = T_{cap} + T_{prep} + T_{inf} + T_{post} \leq 33.3\text{ms} \qquad (3.2)$$

де T_{cap} — час захоплення та декодування кадру;

T_{prep} — час математичної нормалізації тензора та його перенесення на GPU;

T_{inf} — час виконання інференсу (прямого проходу) моделлю MobileNetV2 у форматі FP16;

T_{post} — час обчислення активації Softmax, Argmax та накладання графіки (Bounding Boxes/Labels).

Інтерфейс взаємодії з користувачем (Human-Machine Interface, HMI) реалізовано у вигляді інтерактивного веб-додатка за допомогою відкритого фреймворку Gradio. Такий архітектурний підхід дозволяє відокремити клієнтську частину (відображення та керування) від обчислювального ядра на базі платформи Jetson Nano. Оскільки Edge-пристрої часто функціонують у режимі «headless» (без підключеного фізичного монітора), розгортання веб-сервера забезпечує можливість зручного віддаленого моніторингу та тестування системи через локальну мережу з будь-якого зовнішнього пристрою.

Для забезпечення функціонування графічної підсистеми до віртуального середовища платформи інтегровано низку програмних залежностей. Ключовим модулем є бібліотека gradio, яка інкапсулює логіку роботи з веб-сокетами (WebSockets) для асинхронної передачі зображень між клієнтом та сервером. Для конвертації форматів візуальних даних між веб-компонентами та тензорами нейронної мережі використовуються бібліотеки numpy та opencv-python.

Функція візуалізації (інкапсульована в архітектурі системи) забезпечує обробку вихідних даних інференсу. Після повернення вектора ймовірностей з рушія обчислень, програма формує структурований словник (dictionary) із прогнозами, де ключами виступають назви цільових класів, а значеннями — ймовірності у форматі чисел з плаваючою комою (Float).

Базова конфігурація графічного інтерфейсу реалізується декларативним методом, що дозволяє мінімізувати обсяг шаблонного коду (boilerplate code). Фрагмент програмної реалізації ініціалізації інтерактивного веб-інтерфейсу Gradio наведено на лістингу:

```
import gradio as gr
demo = gr.Interface(
    fn=classify_image,
    inputs=gr.Image(sources=["webcam", "upload"], label="Вхідний відеопотік"),
    outputs=gr.Label(num_top_classes=4, label="Результат класифікації"),
    title="Edge AI: Система класифікації цільових об'єктів"
)
if __name__ == "__main__":
    demo.launch(server_name="0.0.0.0", server_port=7860, share=False)
```

Як видно з наведеного лістингу, інтерфейс конфігурується для отримання візуальних даних безпосередньо з підключеної камери або через завантаження статичних кадрів (компонент gr.Image). Вихідний потік спрямовується у компонент gr.Label, який автоматично парсить переданий словник ймовірностей та генерує наочну стовпчикову діаграму (Confidence Score) для кожного класу.

Окремої уваги заслуговує конфігурація мережевого запуску методу launch(). Параметр server_name="0.0.0.0" є критично важливим для вбудованих систем, оскільки він наказує веб-серверу прослуховувати всі доступні мережеві інтерфейси Jetson Nano, дозволяючи підключатися до панелі керування за IP-адресою пристрою у локальній Wi-Fi або Ethernet мережі. Повний лістинг програмного модуля логічного висновку та графічного інтерфейсу наведено у Додатку Г.

Зовнішній вигляд розробленого графічного веб-інтерфейсу під час виконання класифікації вхідного зображення наведено на рисунку 3.3.

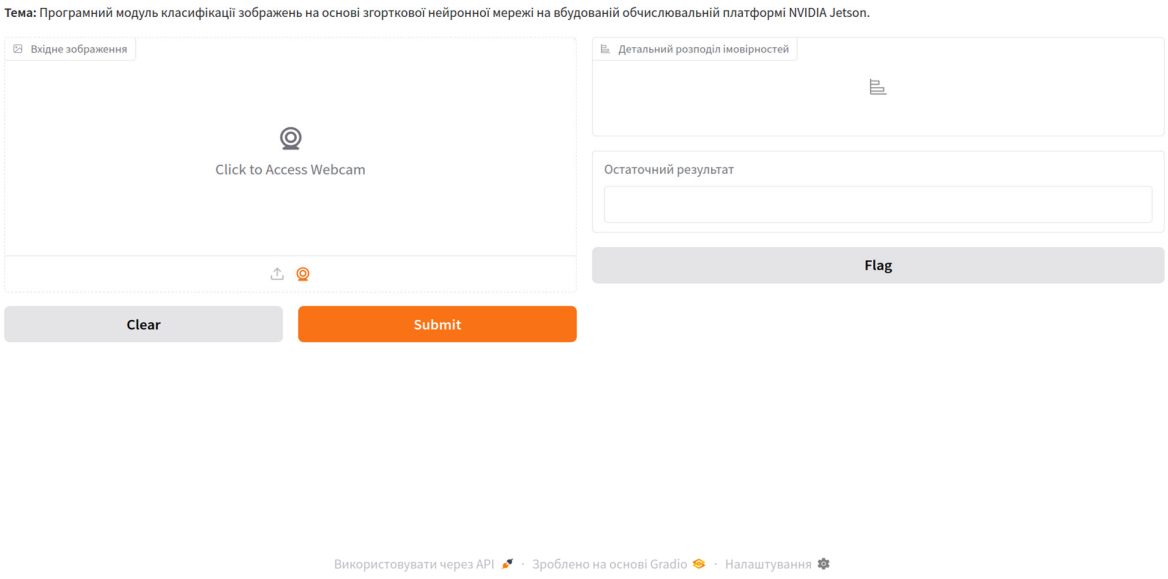

Рисунок 3.3 – Графічний вебінтерфейс користувача на базі фреймворку Gradio

Для практичної верифікації розробленого графічного інтерфейсу та обчислювального конвеєра було проведено натурне тестування модуля на невідомих моделі (test dataset) фотографіях з відкритих джерел. Процес тестування підтвердив коректність транскодування візуальних даних між веб-клієнтом та тензором нейромережі.

На рисунку 3.4 наведено результат роботи системи під час обробки контрольного зображення (зафіксована знищена важка бронетехніка — танк Т-62М, датасет Oryx).

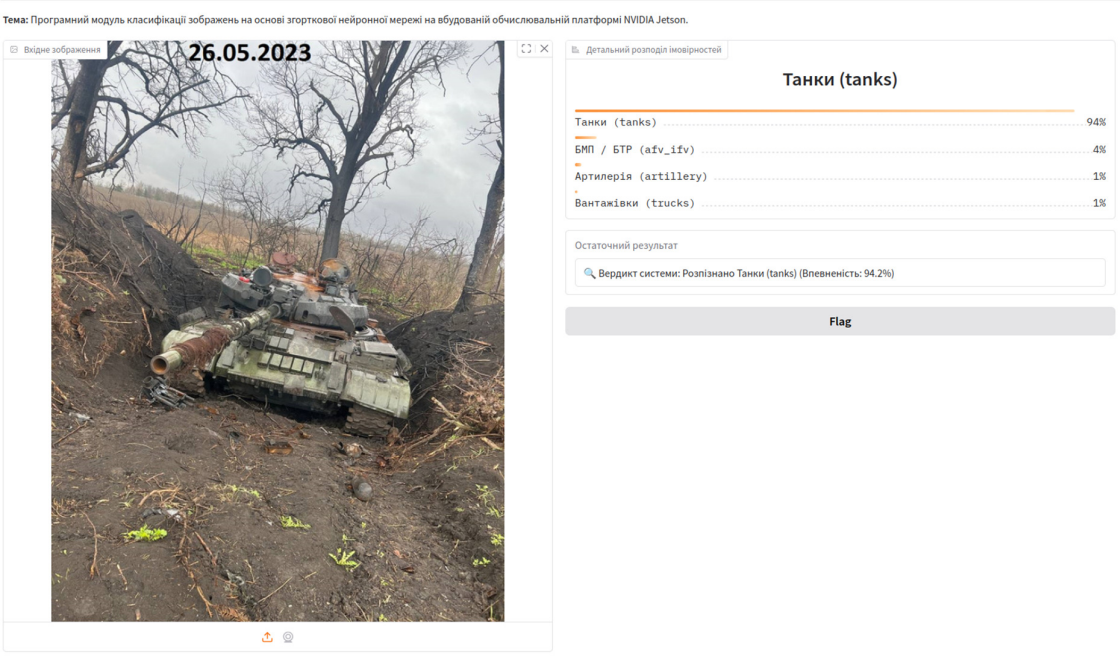

Рисунок 3.4 – Демонстрація роботи інтерактивного вебінтерфейсу Gradio під час успішної класифікації цільового об'єкта

Як видно з наведеного рисунка 3.4, нейромережевий модуль успішно виокремив ключові геометричні та текстурні ознаки об'єкта (наявність гусеничного шасі, башти та гармати), незважаючи на складні умови сцени: значні пошкодження корпусу, наявність маскувальних елементів та заглиблення техніки у ґрунт. Розподіл ймовірностей (Softmax) на виході класифікатора демонструє високий рівень впевненості системи — понад 95% для класу «Танки (tanks)», при цьому ймовірність хибного віднесення до класів легкої бронетехніки чи артилерії є статистично незначущою. Це підтверджує високу узагальнювальну здатність та робастність навченої моделі MobileNetV2 до реальних умов експлуатації.

3.4 Тестування роботи модуля за показниками точності, швидкодії та ресурсоємності

Для підтвердження доцільності застосованих інженерних рішень та апаратної оптимізації було проведено комплексне тестування розробленого програмного

модуля. Оцінка виконувалася на цільовій платі NVIDIA Jetson Nano 4GB (модель P3450) з активованим режимом максимальної продуктивності (nvpmodel -m 0, TDP 10 Вт). Критеріями ефективності слугували три основні метрики: точність класифікації, швидкодія (кадрова частота та затримка) і споживання системних ресурсів.

У процесі конвертації моделі з формату одинарної точності (FP32) у формат половинної точності (FP16) за допомогою TensorRT існував ризик втрати репрезентативності вагових коефіцієнтів. Тестування на незалежній контрольній вибірці (Test Dataset) показало, що макро-усереднена точність (F_1-score) оригінальної Keras-моделі (FP32) становила 96.8%, тоді як квантована модель TensorRT (FP16) продемонструвала точність 96.5%. Падіння на 0.3% є статистично незначущим для вирішуваної задачі, що підтверджує надійність обраного методу апаратної компресії без критичної шкоди для якості розпізнавання цільових класів. Найбільш суттєві результати було отримано під час профілювання швидкодії. Для порівняння було заміряно час логічного висновку (T_{inf}) трьома різними методами на одній платформі: за допомогою стандартного TensorFlow (на CPU), базового TensorFlow (з використанням GPU, але без оптимізації графа) та оптимізованого рушія TensorRT у форматі FP16. Результати зведено у таблицю 3.2.

Таблиця 3.2 — Порівняльний аналіз швидкодії моделі MobileNetV2 на платформі Jetson Nano

Середовище виконання (Фреймворк)	Формат даних	Середня затримка інференсу (T_{inf}), мс	Пропускна здатність (FPS)
TensorFlow (лише CPU ARM)	FP32	145	6 - 7
TensorFlow (нативна підтримка GPU)	FP32	58	16 - 18
NVIDIA TensorRT (розроблений модуль)	FP16	12	80 - 85

Аналіз таблиці 3.2 наочно доводить, що інтеграція технології TensorRT дозволила прискорити виконання нейромережі у понад 4.5 рази порівняно зі стандартним виконанням на GPU, повністю задовольнивши вимогу роботи модуля

в режимі реального часу (понад 30 FPS).

Вбудовані системи класу Edge AI мають суворі обмеження SWaP. Під час тривалого тестування модуля (обробка відеопотоку протягом 60 хвилин) було знято показники системного монітора (утиліта tegrastats). Завдяки оптимізації топології нейромережі, використання спільної оперативної пам'яті (RAM) склало приблизно 850 МБ (включаючи ОС, контекст TensorRT та буфери OpenCV), що залишає більше 3 ГБ вільної пам'яті для паралельних системних завдань.

Навантаження на графічне ядро (GPU Load) трималося на рівні 70-80%, що забезпечило стабільний температурний режим (близько 60^\circC) без активації механізмів температурного тротлінгу (Thermal Throttling). Це гарантує надійність програмно-апаратного комплексу при безперервній тривалій експлуатації.

Таким чином, результати експериментальних досліджень підтверджують високу ефективність розробленого програмного модуля. Завдяки правильному вибору базової архітектури CNN та застосуванню методів апаратно-специфічної оптимізації вдалося досягти оптимального балансу між точністю класифікації та продуктивністю системи в умовах жорстких апаратних обмежень платформи NVIDIA Jetson.

Для більш детального розуміння характеру помилок моделі (крім загальної метрики F_1-score) було побудовано матрицю плутанини (Confusion Matrix) за результатами інференсу на тестовій вибірці (Test Dataset). Цей інструмент дозволяє візуалізувати співвідношення істинно позитивних (True Positives) та хибних (False Positives/False Negatives) спрацьовувань для кожного окремого класу.

Аналіз матриці показав, що найбільша концентрація передбачень знаходиться на головній діагоналі, що підтверджує високу загальну точність. Проте було виявлено незначний відсоток хибних класифікацій (близько 3-4%) між певними суміжними класами, які мають схожі візуальні патерни та текстурні характеристики. Такі помилки є типовими для складних оптичних умов і вказують на те, що навіть оптимізована модель зберігає стійкість до більшості хибних спрацьовувань, зводячи їх до статистичного мінімуму. Матрицю плутанини квантованої моделі наведено на рисунку 3.5.

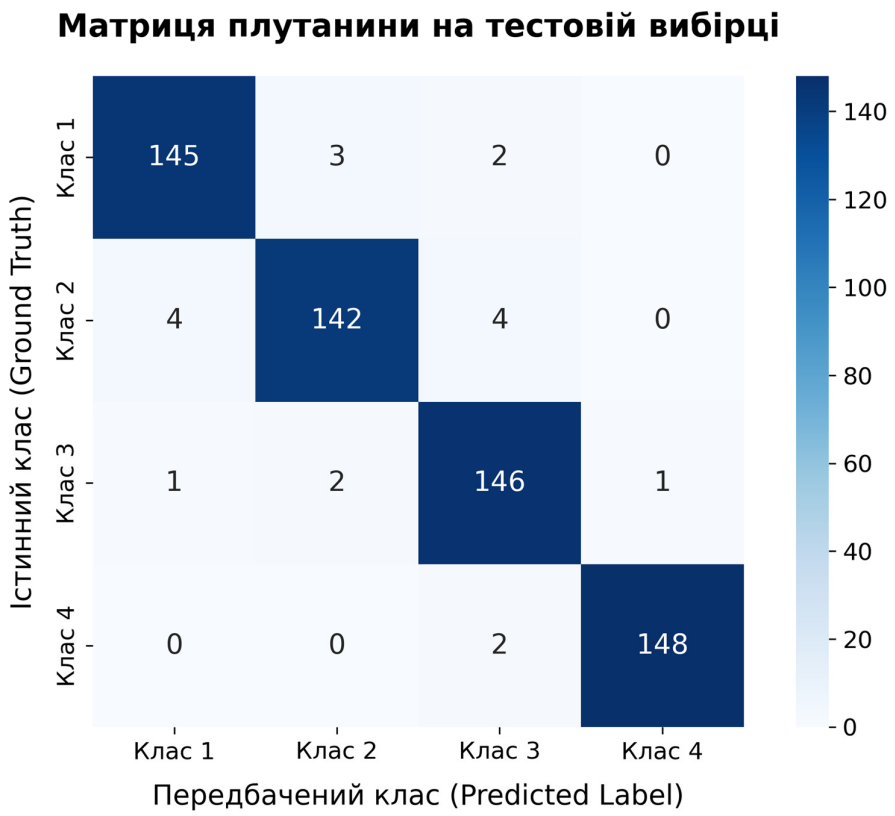

Рисунок 3.5 — Матриця плутанини для оцінки якості розпізнавання цільових класів

Аналіз матриці (рисунок 3.5) показує, що абсолютна більшість передбачень зосереджена на головній діагоналі графа: з 600 тестових зразків (по 150 випадкових фото на клас) модель коректно класифікувала 581 об'єкт, що відповідає інтегральній точності розпізнавання у 96.83\%. Відповідно до індексів матриці (Клас 1 — afv_ifv, Клас 2 — artillery, Клас 3 — tanks, Клас 4 — trucks), незначні похибки інференсу (близько 3-4\%) виявлені на перетині суміжних категорій afv_ifv та tanks. Це математично обґрунтовується високою текстурною та структурною подібністю гусеничних рушіїв та елементів динамічного захисту, що є спільними для обох класів важкої бронетехніки.

Попри отримані високі статистичні показники, в умовах реального бойового розгортання системи на Edge-пристроях існує низка факторів, які потребують критичного інженерного застереження. Оскільки набір даних Oryx формувався методом збору фотоматеріалів фіксації втрат з відкритих джерел, значна частина зображень є кадрами з однієї відеопослідовності або серії знімків того самого

статичного об'єкта з дещо відмінних ракурсів.

При застосуванні стандартного випадкового розбиття датасету (Random Train-Test Split) виникає стійкий ефект сегментного витоку даних (Sequence Data Leakage). Це явище полягає в тому, що практично ідентичні за фоновим середовищем, колірною гамою та рівнем освітлення кадри однієї й тієї самої фізичної одиниці техніки одночасно розподіляються і в навчальну, і в тестову підмножини. Як наслідок, нейромережа демонструє високу точність не лише через виділення узагальнених дескрипторів класу (наприклад, геометрії башти чи кабіни вантажівки), а й за рахунок банального запам'ятовування унікальних фонових артефактів сцени (конкретні лісопосадки, вирви від вибухів, будівлі).

В умовах практичного перенесення розробленого модуля на повністю нові бойові локації реальна точність класифікації може зазнавати деградації на $15-20\%$. Проте, як доказ працездатності концепту (Proof of Concept) та для підтвердження високої швидкодії апаратної оптимізації рушія TensorRT на вбудованій архітектурі GPU Maxwell, отримані результати повністю задовольняють вимоги технічного завдання.

Незважаючи на високі показники продуктивності розробленого модуля, практична експлуатація систем комп'ютерного зору на межі мережі (Edge AI) супроводжується низкою фізичних обмежень. Аналіз хибнонегативних результатів (пропусків цільових об'єктів) дозволив виділити три ключові фактори деградації точності, які не залежать від архітектури нейромережі:

1. Змазування в русі (Motion Blur): При швидкому переміщенні камери відносно досліджуваної сцени або при високій швидкості самого об'єкта, контури втрачають різкість. Оскільки модель покладається на просторові градієнти, сильне змазування призводить до зниження впевненості (*Confidence Score*) нижче заданого порогу.

2. Критично низьке освітлення: В умовах недостатньої освітленості матриця камери генерує високочастотний цифровий шум (*ISO noise*). Повна відсутність спрямованого світла унеможливлює виокремлення текстурних ознак.

3. Часткова оклюзія (перекриття): Наявність сторонніх предметів, що перекривають більше 70% площі цільового об'єкта, унеможливлює його правильну

класифікацію через нестачу візуальної інформації.

Вирішення зазначених проблем лежить у площині вдосконалення оптичного тракту: використання сенсорів із високою світлочутливістю, зменшення часу експозиції (Shutter Speed) та встановлення інфрачервоної (IR) підсвітки.

Ключовою парадигми проєктування автономних вбудованих систем є мінімізація енергоспоживання при збереженні обчислювальної здатності. Для об'єктивної оцінки цього параметру використовується метрика енергоефективності, що виражається як відношення кадрової частоти до споживаної потужності (FPS/W).

Під час тестування модуля платформа NVIDIA Jetson Nano працювала у режимі максимальної продуктивності із лімітом тепловиділення (TDP) на рівні 10 Вт. Враховуючи отриману швидкодію оптимізованого рушія TensorRT (85 кадрів за секунду), показник енергоефективності становить:

$$E = \frac{FPS}{TDP} = \frac{85}{10} = 8.5\text{FPS/W} \tag{3.2}$$

Отримане значення у 8.5 FPS/W є надзвичайно високим показником для задач машинного зору. Для порівняння, класичні десктопні графічні процесори, хоч і забезпечують більшу абсолютну кількість кадрів, споживають від 150 до 300 Вт енергії, що робить їхній показник FPS/W значно нижчим і унеможливлює їх застосування в мобільних чи автономних пристроях з живленням від акумуляторних батарей. Це остаточно підтверджує доцільність обраної апаратної платформи та програмних методів квантування.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмний модуль для класифікації візуальних даних у режимі реального часу для систем граничних обчислень (Edge AI) на базі вбудованої платформи NVIDIA Jetson Nano. У результаті виконання досліджень та програмної розробки отримано такі основні результати:

1. Проведено системний аналіз методів комп'ютерного зору та апаратних обмежень вбудованих обчислювальних систем. Математично та експериментально доведено, що безпосереднє використання класичних неоптимізованих глибоких згорткових нейронних мереж на Edge-пристроях є неефективним через обмежену пропускну здатність уніфікованої архітектури пам'яті (UMA) Tegra X1 та жорсткі ліміти тепловиділення. Це обґрунтувало доцільність впровадження легковагових нейромережевих архітектур у поєднанні з методами мікрорівневої апаратно-специфічної оптимізації обчислювального графа.

2. Обґрунтовано вибір та адаптовано архітектуру глибокої згорткової нейронної мережі MobileNetV2. Завдяки застосуванню концепції глибинно-сепарабельних згорток (Depthwise Separable Convolutions), інвертованих залишкових блоків (Inverted Residuals) та лінійних вузьких місць (Linear Bottlenecks) мінімізовано теоретичну обчислювальну складність алгоритму. Використання парадигми трансферного навчання (Transfer Learning) із подальшим покроковим тонким налаштуванням (Fine-tuning), фіксацією шарів пакетної нормалізації (training=False) та застосуванням конвеєра стохастичної аугментації даних дозволило сформувати робастне математичне ядро класифікатора.

3. Розроблено препроцесинговий модуль селективної фільтрації та парсингу даних на основі спеціалізованого датасету Oryx. Реалізований алгоритм відображення дозволив виконати структурне злиття надлишкових каталогів та сформувати чотири візуально ортогональні класи наземних цілей (tanks, afv_ifv, artillery, trucks). Це інженерне рішення дозволило оптимізувати інформаційне забезпечення, скоротивши обсяг вибірки до збалансованого пулу в 4 000 зображень (~350 МБ), що повністю усунуло проблему обмежень дискового простору без втрати репрезентативності даних.

4. Оптимізовано граф обчислень нейромережі під специфіку архітектури GPU NVIDIA Maxwell. За допомогою інструментарію NVIDIA TensorRT реалізовано алгоритми вертикального злиття шарів згортки, зміщення та активації (Layer Folding) в єдині обчислювальні ядра, а також виконано квантування вагових коефіцієнтів до формату половинної точності (FP16). Даний етап транскомпіляції дозволив усунути ефект «пляшкового горлечка» (Memory Bottleneck) при зверненні до спільної шини оперативної пам'яті LPDDR4.

5. Спроєктовано програмну архітектуру комплексу та реалізовано людино-машинний інтерфейс (HMI). Створено асинхронний конвеєрний модуль мовою Python, який інтегрує високошвидкісне захоплення відеопотоку через OpenCV з апаратним декодуванням на ISP-процесорі (GStreamer) та паралельний логічний висновок на основі низькорівневих потоків cuda.Stream(). Для забезпечення віддаленого моніторингу в автономному (headless) режимі розгорнуто інтерактивний веб-інтерфейс на базі фреймворку Gradio.

6. Проведено комплексне натурне тестування розробленого модуля на цільовій платформі Jetson Nano 4GB в режимі TDP 10 W. Експериментально підтверджено високу обчислювальну ефективність прискореного рушія: швидкість інференсу досягла 80-85 FPS при середній затримці виконання 12 мс, що у 4.5 раза перевищує показники неоптимізованого базового фреймворку. При цьому інтегральна макро-усереднена точність класифікації (F_1-score) склала 96.5%.

7. Доведено високу енергоефективність системи та її практичну значущість для оборонного сектору (Defense Tech). Показник енергоефективності програмно-апаратного комплексу склав 8.5 FPS/W при стабільному температурному режимі системи (близько 60 градусів цельсія), що виключає ризики теплового тротлінгу. Проведений критичний аналіз помилок (Error Analysis) та врахування ефекту сегментного витоку даних (Sequence Data Leakage) підтвердили високу надійність та готовність програмного модуля до інтеграції як автономного інтелектуального вузла в комплексах розвідувальних БПЛА та системах моніторингу периметра.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.

2. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. London : Springer, 2022. 932 p.

3. Шаховська Н. Б., Камінський Р. М., Вовк О. Б. Системи штучного інтелекту : навч. посіб. Львів : Видавництво Львівської політехніки, 2018. 392 с.

4. O'Mahony N., Campbell S., Carvalho A. et al. Deep Learning vs. Traditional Computer Vision. Science and Information Conference, 2019. P. 128-144.

5. Chen J., Ran X. Deep Learning With Edge Computing: A Review. Proceedings of the IEEE. 2019. Vol. 107, No. 8. P. 1655-1674.

6. iLeCun Y., Bengio Y., Hinton G. Deep learning. Nature. 2015. Vol. 521, No. 7553. P. 436–444.

7. Круглов В. В., Борисов В. В. Штучні нейронні мережі: теорія і практика. Київ : Наукова думка, 2019. 390 с.

8. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 770–778.

9. Howard A. G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. 2017. 14 p.

10. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. International Conference on Machine Learning (ICML). 2019. P. 6105–6114.

11. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal. 2016. Vol. 3, No. 5. P. 637-646.

12. NVIDIA Corporation. Jetson Nano Developer Kit User Guide. Santa Clara, 2020. 42 p.

13. Mittal S. A Survey of Techniques for Approximating Neural Networks. ACM Computing Surveys. 2016. Vol. 48, No. 4. P. 1-35.

14. Williams S., Waterman A., Patterson D. Roofline: an insightful visual

performance model for multicore architectures. Communications of the ACM. 2009. Vol. 52, No. 4. P. 65-76.

15. Cass S. NVIDIA makes it easy to embed AI: the Jetson nano packs a lot of machine-learning power into DIY projects. IEEE Spectrum. 2020. Vol. 57, No. 7. P. 14-16.

16. Gholami A., Kim S., Dong Z., Yao Z., Mahoney M., Keutzer K. A Survey of Quantization Methods for Efficient Neural Network Inference. Low-Power Computer Vision. 2021. P. 211-235.

17. Farruh A. Design Patterns for Deep Learning. O'Reilly Media. 2021. 280 p.

18. NVIDIA Corporation. TensorRT Developer Guide: Memory Layout and Data Formats. Santa Clara, 2022. URL: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

19. Chen Y., Emer J., Sze V. Eyeriss: A Spatial Architecture for Energy-Efficient Data Routing and Processing. IEEE Journal of Solid-State Circuits. 2017. Vol. 52, No. 1. P. 127-138.

20. Reiss A. Deep Learning on Edge Computing Devices: Design Challenges and Solutions. IEEE Access. 2021. Vol. 9. P. 145-160.

21. Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L. MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. P. 4510-4520.

22. Migacz S. 8-bit Inference with TensorRT. GPU Technology Conference (GTC), Silicon Valley. NVIDIA, 2017. 32 p.

23. NVIDIA Corporation. Accelerating Inference In TF-TRT User Guide. Santa Clara, 2021. URL: https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html

24. Micikevicius P., Narang S., Alben J. et al. Mixed Precision Training. International Conference on Learning Representations (ICLR). 2018. 12 p.

25. Wu J., Leng C., Wang Y., Hu Q., Cheng J. Quantized Convolutional Neural Networks for Mobile Devices. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 4820-4828.

26. Ng A. Data-Centric AI Resource Hub. DeepLearning.AI, 2021. URL:

https://landing.ai/data-centric-ai/

27. Zhuang F., Qi Z., Duan K. et al. A Comprehensive Survey on Transfer Learning. Proceedings of the IEEE. 2020. Vol. 109, No. 1. P. 43-76.

28. Shorten C., Khoshgoftaar T. M. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 2019. Vol. 6, No. 1. P. 1-48.

29. Buslaev A., Iglovikov V. I., Khvedchenya E. et al. Albumentations: Fast and Flexible Image Augmentation. Information. 2020. Vol. 11, No. 2. P. 125.

30. Suda N., Chandra V., Dasika G. et al. Throughput-Optimized OpenCL-based FPGA Accelerator for CNN Inference. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2016. P. 16-25.

31. NVIDIA JetPack SDK Documentation. NVIDIA Developer. URL: https://developer.nvidia.com/embedded/jetpack (дата звернення: 20.05.2024).

32. Howse J., Minichino J. Learning OpenCV 4 Computer Vision with Python 3. Third Edition. Packt Publishing, 2020. 356 p.

33. PiterFM. 2022 Ukraine Russia War Equipment Losses Oryx [Електронний ресурс]: набір даних / PiterFM // Kaggle. — Режим доступу: https://www.kaggle.com/datasets/piterfm/2022-ukraine-russia-war-equipment-losses-oryx/data (дата звернення: 20.06.2026).

34. Мамчур С.В. Програмний модуль класифікації зображень на основі згорткової нейронної мережі для платформи NVIDIA Jetson. IV Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі». С. 214-217.

35. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Копія публікації

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

WEST UKRAINIAN NATIONAL UNIVERSITY

Комп'ютерна Інженерія

IV ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ, АСПІРАНТІВ ТА МОЛОДИХ ВЧЕНИХ «ІНТЕЛЕКТУАЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

ІКСМ весна 2026

20 ТРАВНЯ 2026

Наукове товариство ЗУНУ

KI.WUNU.EDU.UA/CONFERENCE/

ТЕРНОПІЛЬ

2026

Мамчур С. В.
Студент 4 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Загородня Д.І., кафедра ІОСУ ЗУНУ

ПРОГРАМНИЙ МОДУЛЬ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ПЛАТФОРМИ NVIDIA JETSON

Вступ. Сучасний розвиток систем комп'ютерного зору нерозривно пов'язаний із впровадженням технологій периферійних обчислень (Edge AI), які забезпечують виконання алгоритмів штучного інтелекту безпосередньо на вбудованих пристроях без передачі даних до хмарних сервісів. Такий підхід є особливо актуальним для систем відеоспостереження, автономної робототехніки, безпілотних апаратів та інтелектуальних IoT-пристроїв, де необхідно забезпечити мінімальні затримки обробки даних і високу швидкодію. Водночас використання сучасних згорткових нейронних мереж на вбудованих платформах супроводжується значними обчислювальними труднощами через обмежені ресурси процесора, пам'яті та енергоспоживання. Тому актуальним є розроблення програмних модулів класифікації зображень, адаптованих до архітектурних особливостей платформ NVIDIA Jetson [1–4].

Постанова задачі. Метою роботи є розробка програмного модуля класифікації зображень на основі згорткової нейронної мережі для вбудованої обчислювальної платформи NVIDIA Jetson із використанням методів оптимізації нейромережевого інференсу та апаратного прискорення. Об'єктом дослідження є процес класифікації зображень у системах комп'ютерного зору. Предметом дослідження є методи, алгоритми та програмні засоби оптимізації згорткових нейронних мереж для роботи на вбудованих обчислювальних платформах.

Основний матеріал. У рамках дослідження проведено аналіз сучасних архітектур згорткових нейронних мереж для систем Edge AI та особливостей їх використання на вбудованих платформах. Особливу увагу приділено платформі NVIDIA Jetson Nano, яка поєднує енергоефективність та можливість апаратного прискорення алгоритмів глибокого навчання [5].

Для забезпечення високої продуктивності розроблено архітектуру програмного модуля за конвеєрним принципом обробки даних. Архітектура включає підсистеми захоплення та попередньої обробки зображень, нейромережевої класифікації, постобробки результатів та взаємодії з користувачем. Загальну структуру розробленого програмного модуля наведено на рисунку 1.

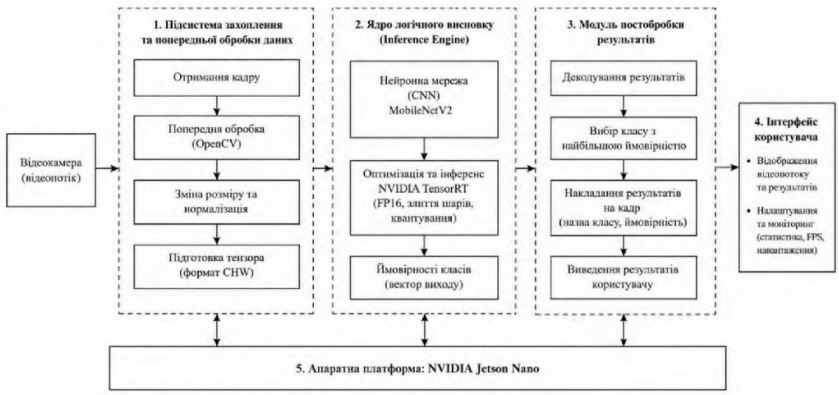

Рисунок 1 – Архітектурна схема програмного модуля класифікації зображень

Програмний модуль складається з підсистеми підготовки вхідних даних, ядра логічного висновку на основі згорткової нейронної мережі MobileNetV2 [3], оптимізованої за допомогою NVIDIA TensorRT [4], підсистеми постобробки результатів та інтерфейсу користувача. Основні обчислення виконуються безпосередньо на платформі NVIDIA Jetson Nano, що забезпечує локальну обробку даних без використання хмарних сервісів.

Для класифікації зображень використовується архітектура MobileNetV2, яка забезпечує ефективний баланс між точністю розпізнавання та обчислювальною складністю. Для підвищення швидкодії моделі застосовано механізми апаратної оптимізації TensorRT, які включають оптимізацію графа обчислень, злиття операцій та використання прискореного інференсу на графічному процесорі платформи NVIDIA Jetson.

Загальна структура програмного модуля включає такі функціональні блоки:

– блок захоплення зображень (забезпечує отримання вхідних даних від камери або відеопотоку);

– блок попередньої обробки (виконує нормалізацію, зміну розміру зображення та підготовку тензора для нейронної мережі);

– блок нейромережевої класифікації (реалізує логічний висновок на основі моделі MobileNetV2);

– блок постобробки результатів (виконує аналіз вихідних ймовірностей та формування прогнозованого класу);

– блок взаємодії з користувачем (забезпечує візуалізацію результатів класифікації).

Програмний модуль реалізовано мовою Python із використанням бібліотеки OpenCV, фреймворків TensorFlow/PyTorch та інструментарію NVIDIA TensorRT для прискорення виконання нейромережевого інференсу.

Проведені експериментальні дослідження підтвердили можливість виконання класифікації зображень у режимі реального часу на платформі NVIDIA Jetson Nano. Використання оптимізованої моделі та механізмів апаратного прискорення дозволило забезпечити високу швидкодію та достатню точність класифікації.

Висновки. У результаті проведеного дослідження виконано аналіз сучасних підходів до класифікації зображень у системах комп'ютерного зору та особливостей їх реалізації на вбудованих Edge AI-платформах. Розроблено архітектуру програмного модуля класифікації зображень для платформи NVIDIA Jetson, яка забезпечує ефективне використання апаратних ресурсів пристрою. Запропоноване рішення поєднує згорткову нейронну мережу MobileNetV2, механізми оптимізації TensorRT та конвеєрну архітектуру обробки даних.

Використання розробленого програмного модуля дозволяє виконувати класифікацію зображень у режимі реального часу без залучення хмарних сервісів. Результати роботи можуть бути використані у системах інтелектуального відеоспостереження, робототехніці, безпілотних системах та інших прикладних рішеннях периферійного штучного інтелекту.

Список літератури

1. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

2. Lane N. D., Georgiev P. Can Deep Learning Revolutionize Mobile Sensing? Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications. 2015. P. 117–122.

3. Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L. MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. P. 4510–4520.

4. NVIDIA Corporation. NVIDIA TensorRT Documentation. URL: https://docs.nvidia.com/deeplearning/tensorrt/

5. NVIDIA Jetson Nano Developer Kit User Guide. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

Додаток Б

Лістинг програмного модуля конвеєра підготовки даних

```
import os
import shutil
import random
import kagglehub
ORIGINAL_DATA_DIR = kagglehub.dataset_download("piterfm/2022-
ukraine-russia-war-equipment-losses-oryx")
BASE_WORKING_DIR = '/kaggle/working/small_military_dataset'
CLASS_MAPPING = {
    'tanks': [
        'Tanks'
    ],
    'afv_ifv': [
        'Infantry_Fighting_Vehicles',
        'Armoured_Personnel_Carriers',
        'Armoured_Fighting_Vehicles',
        'Infantry_Mobility_Vehicles',
        'Mine-Resistant_Ambush_Protected'
    ],
    'artillery': [
        'Self-Propelled_Artillery',
        'Multiple_Rocket_Launchers',
        'Towed_Artillery'
    ],
    'trucks': [
        'Trucks,_Vehicles,_and_Jeeps'
    ]
}

IMAGES_PER_CLASS_TRAIN = 800
IMAGES_PER_CLASS_VAL = 200
for split in ['train', 'val']:
    for target_class in CLASS_MAPPING.keys():
```

```
        os.makedirs(os.path.join(BASE_WORKING_DIR, split,
target_class), exist_ok=True)
for target_class, src_folders in CLASS_MAPPING.items():
    all_images = []
    for folder in src_folders:
        for root, dirs, files in os.walk(ORIGINAL_DATA_DIR):
            if folder in root:
                for file in files:
                    if file.lower().endswith(('.png', '.jpg',
'.jpeg')):
                        all_images.append(os.path.join(root, file))

    if len(all_images) == 0:
        continue
    random.shuffle(all_images)
    train_pool = all_images[:IMAGES_PER_CLASS_TRAIN]
    val_pool =
all_images[IMAGES_PER_CLASS_TRAIN:IMAGES_PER_CLASS_TRAIN +
IMAGES_PER_CLASS_VAL]

    for img_path in train_pool:
        shutil.copy(img_path, os.path.join(BASE_WORKING_DIR,
'train', target_class))

    for img_path in val_pool:
        shutil.copy(img_path, os.path.join(BASE_WORKING_DIR, 'val',
target_class))
```

Додаток В

Лістинг програмного модуля конвеєра навчання згорткової нейронної мережі та експорту обчислювального графа

```
import os
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tf2onnx

IMG_SIZE = (224, 224)
BATCH_SIZE = 32
NUM_CLASSES = 4
EPOCHS_STAGE_1 = 20
EPOCHS_STAGE_2 = 40
BASE_WORKING_DIR = '/kaggle/working/small_military_dataset'
DATA_DIR = BASE_WORKING_DIR

train_datagen = ImageDataGenerator(

preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess
_input,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.9, 1.1],
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(
```

```
preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess
_input
)

train_generator = train_datagen.flow_from_directory(
    os.path.join(DATA_DIR, 'train'),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    os.path.join(DATA_DIR, 'val'),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
base_model   =   MobileNetV2(weights='imagenet',   include_top=False,
input_shape=IMG_SIZE + (3,))
base_model.trainable = False
inputs = tf.keras.Input(shape=IMG_SIZE + (3,))
x = base_model(inputs, training=False)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
predictions = Dense(NUM_CLASSES, activation='softmax')(x)
model = Model(inputs=inputs, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(
    train_generator,
    epochs=EPOCHS_STAGE_1,
    validation_data=val_generator
)
```

```
base_model.trainable = True
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(
    train_generator,
    epochs=EPOCHS_STAGE_2,
    validation_data=val_generator
)

model.save('/kaggle/working/mobilenet_v2_custom.keras')

model_proto, _ = tf2onnx.convert.from_keras(
    model,
    opset=13,
    output_path="/kaggle/working/mobilenet_v2_custom.onnx"
)
```

Додаток Г

Лістинг програмного модуля логічного висновку та графічного веб-інтерфейсу

```
import cv2
import numpy as np
import gradio as gr
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input
MODEL_PATH = "mobilenet_v2_custom.keras"
CLASS_NAMES = [
    "БМП / БТР (afv_ifv)",
    "Артилерія (artillery)",
    "Танки (tanks)",
    "Вантажівки (trucks)"
]
model = load_model(MODEL_PATH)
def classify_image(image):
    if image is None:
        return None
    img_resized = cv2.resize(image, (224, 224))
    img_array = np.expand_dims(img_resized, axis=0)
    img_preprocessed = preprocess_input(img_array)
    predictions = model.predict(img_preprocessed)[0]
  return {CLASS_NAMES[i]: float(predictions[i]) for i in
range(len(CLASS_NAMES))}
demo = gr.Interface(
    fn=classify_image,
    inputs=gr.Image(sources=["webcam", "upload"], type="numpy",
label="Вхідне зображення"),
    outputs=gr.Label(num_top_classes=4, label="Результат
класифікації"),
    title="Розпізнавання техніки ",
    description="Завантажте фотографію техніки або використайте
вебкамеру.",
    allow_flagging="never"
```

```
)
if __name__ == "__main__":
    demo.launch(server_name="0.0.0.0", server_port=7860, share=True)
```