МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Кваліфікаційна наукова
праця на правах рукопису

СТЕЦЬ СЕРГІЙ ЮРІЙОВИЧ

УДК 004.032.26:[0.04.93’1:629.33

ПІДВИЩЕННЯ ТОЧНОСТІ ТА ШВИДКОДІЇ ДЕТЕКТУВАННЯ ЗОБРАЖЕНЬ АВТОМОБІЛІВ ЗАСОБАМИ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ YOLO

121 – Інженерія програмного забезпечення
12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ / Стець Сергій Юрійович /

Науковий керівник: **Баловсяк Сергій Васильович**, доктор технічних наук, доцент

Чернівці – 2026

АНОТАЦІЯ

Стець С.Ю. Підвищення точності та швидкодії детектування зображень автомобілів засобами згорткової нейронної мережі YOLO. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – «Інженерія програмного забезпечення» – Чернівецький національний університет імені Юрія Федьковича, Чернівці, 2026.

Детектування зображень транспортних засобів є базовим етапом для автоматизованого дистанційного зондування та контролю стану об'єктів у різноманітних системах Інтернету речей (IoT). Автомобілі суттєво впливають на стан дорожньої обстановки, тому підвищення ефективності їх виявлення є ключовим фактором надійності транспортних систем. Для розпізнавання та детектування зображень автомобілів широко застосовуються штучні нейронні мережі (ШНМ), зокрема, згорткові нейронні мережі (ЗНМ). Одним з найбільш ефективних засобів детектування зображень об'єктів є ЗНМ з архітектурою YOLO (англ. You Only Look Once), які застосовують технології глибокого навчання.

Проте стандартні моделі YOLO, особливо моделі малого розміру, часто демонструють недостатню точність детектування у складних сценах, а моделі великого розміру вимагають значних обчислювальних ресурсів, що ускладнює їх використання у вбудованих системах. Тому підвищення точності та швидкодії детектування зображень автомобілів є **актуальним завданням**.

Метою дисертаційної роботи є підвищення точності та швидкодії детектування зображень автомобілів шляхом вибору архітектури та версії згорткових нейронних мереж, попередньої обробки зображень та донавчання нейронних мереж з архітектурою YOLO на основі автоматизовано створених наборів даних.

Для досягнення мети вирішувались такі **завдання**:

1. Провести аналіз існуючих нейромережевих архітектур, методів та програмних засобів для детектування зображень об'єктів, і на основі такого аналізу виконати вибір архітектури та версії ЗНМ, які б задовольняли вимоги до швидкодії та точності детектування зображень автомобілів.

2. Розробити методику та програмні засоби для попередньої обробки зображень шляхом еквалізації їх гістограм та підвищення локального контрасту, які б забезпечували підвищення точності детектування зображень автомобілів.

3. Розробити методику та програмні засоби для навчання різних версій моделей YOLO та аналізу результатів навчання, які б давали змогу підвищити точність детектування за рахунок донавчання та вибирати версію YOLO з урахуванням вимог для конкретної задачі аналізу зображень.

4. Розробити методику та програмні засоби для автоматизованого створення набору даних із зображень автомобілів засобами моделі ЗНМ YOLO великого розміру, які б забезпечували селекцію унікальних кадрів відеопотоку.

5. Розробити методику та програмні засоби з архітектурою «Вчитель-Учень» для донавчання згорткової нейронної мережі YOLO малого розміру на основі датасету, створеного в автоматизованому режимі.

6. Виконати програмну реалізацію інтелектуальної системи для детектування зображень автомобілів із використанням навчених моделей YOLO.

7. Провести експериментальне дослідження точності та швидкодії розробленої системи на прикладі детектування об’єктів на тестових зображеннях.

Для досягнення мети застосовувалися такі **методи дослідження**: методи глибокого навчання для згорткових нейронних мереж архітектури YOLO; методи цифрової обробки зображень для еквалізації їх гістограм та підвищення локального контрасту; методи інтелектуального аналізу даних (регресійного аналізу, зокрема, лінійної регресії); перцептуальне хешування для селекції унікальних кадрів

відеопотоку; методи комбінаторної оптимізації (угорський алгоритм); методи класифікації даних та зображень; методи комп'ютерного експерименту з використанням програм на мові Python для емпіричного підтвердження адекватності отриманих результатів.

У вступі обґрунтовано актуальність теми дисертаційної роботи, сформульовано мету та завдання проведених досліджень, описано наукову новизну та практичне значення одержаних результатів, представлено методи, об'єкт і предмет досліджень, вказано особистий внесок здобувача, а також наведено дані щодо публікацій за темою дисертації.

У першому розділі здійснено комплексний аналіз сучасних методів і засобів обробки детектування зображень автомобілів. Детально розглянуто структуру та принципи роботи ЗНМ, проаналізовано вплив гіперпараметрів на процес навчання. Описано принципи побудови глибоких залишкових мереж (ResNet), архітектури MobileNet із застосуванням роздільних згорток, а також архітектури EfficientNet. Проведено огляд двостадійних (Faster R-CNN) та одностадійних (YOLO, SSD) детекторів зображень об'єктів. Обґрунтовано вибір архітектури YOLO як базової для подальших досліджень з огляду на її показники точності та швидкодії. Проаналізовано існуючі програмні інструменти (Google Cloud Vision, Amazon Rekognition) та бібліотеки (OpenCV), призначені для детектування зображень.

У другому розділі розроблено та програмно на мові Python реалізовано методику підвищення якості детектування зображень шляхом попередньої обробки вхідних даних. Розділ присвячено архітектурним особливостям обраної моделі YOLOv8 та інтеграції в її конвеєр модуля попередньої обробки. Детально описано структуру YOLOv8. Попередня обробка зображень полягає підвищенні їх контрасту трьома способами: 1) глобальне вирівнювання (еквалізація) гістограми; 2) підвищення локального контрасту методом адаптивної еквалізації гістограми (CLAHE); 3) вирівнювання та центрування гістограми. Точність детектування

зображень автомобілів оцінено за метриками Precision, Recall, F1-score та середньої точності (mAP). Встановлено, що запропонований спосіб попередньої обробки зображень шляхом вирівнювання та центрування гістограми забезпечує найвищу точність детектування автомобілів. Проведено інтелектуальний аналіз даних, а саме регресійний та кореляційний аналіз результатів детектування зображень для знаходження взаємозв'язків між метриками якості детектування.

У третьому розділі розроблено методику для донавчання різних версій ЗНМ YOLO, виконано програмну реалізацію методики, проведено машинне навчання моделей YOLO та проаналізовано його результати. Описано процес формування спеціалізованого набору зображень автомобілів, який включає збір, ручну анотацію зображень та балансування класів. Аугментація даних полягала у розширенні навчальної вибірки зображень шляхом застосування геометричних (повороти, відображення, зміна масштабу) та колористичних (зміна яскравості, насиченості, додавання шуму) перетворень. Це дозволило збільшити обсяг датасету з 1542 до 3578 зображень та запобігти перенавчанню моделі. Виконано донавчання моделей YOLO, попередньо натренованих на датасеті COCO. Шляхом детектування тестових зображень отримано, що донавчання дозволяє підвищити точність детекції за метрикою повноти Recall з 0.58 до 0.91, а за інтегральною метрикою mAP50-95 – з 0.37 до 0.72. Проведено порівняльний аналіз ефективності ЗНМ двох версій: YOLOv8m та новітньої YOLOv11m. Встановлено, що при співставній точності детектування (mAP50 ≈ 0.96) модель YOLOv11m демонструє на 22% менший розмір файлу ваг. Показано, що модель YOLOv8m доцільно вибирати для забезпечення максимальної точності, а YOLOv11m – максимальної швидкодії.

У четвертому розділі розроблено методику для автоматизованого створення набору даних із зображень автомобілів засобами моделі згорткової нейронної мережі YOLO великого розміру. Методика передбачає автоматичну селекцію унікальних кадрів з відеопотоку з використанням методу перцептуального

хешування (pHash). Це дозволяє оцінювати візуальну схожість кадрів і відфільтровувати дублікати, зменшуючи надлишковість даних на 88% (з 993 до 121 кадра у тестовому експерименті). Виявлення схожих об'єктів (автомобілів) на зображеннях виконано угорським алгоритмом з урахуванням метрики детектування IoU та середньоквадратичної помилки (RMSE) між дескрипторами зображень.

Розроблено методику з архітектурою «Вчитель-Учень», яка полягає у донавчанні згорткової нейронної мережі YOLO малого розміру «Учень» (YOLOv8n) на основі автоматизовано створеного спеціалізованого датасету зображень мережею YOLO середнього розміру «Вчитель» (YOLOv8m). Експериментально підтверджено, що такий підхід дозволяє підвищити точність моделі «Учень» за метрикою mAP50 з 0.6864 до 0.9292.

Розроблені методики інтелектуальної системи реалізовано в програмі на мові Python, яка виконується на апаратній платформі одноплатного комп’ютера Raspberry Pi 5 з використанням технологій контейнеризації (Docker). Розроблено веб-інтерфейс програми. Результати обробки тестових зображень показали високу точність детектування автомобілів на зображеннях, отриманих засобами БПЛА.

Наукова новизна отриманих результатів полягає у наступному:

1. Вперше розроблено методику з архітектурою «Вчитель-Учень», особливістю якої є донавчання згорткової нейронної мережі YOLO малого розміру на основі автоматизовано створеного спеціалізованого датасету зображень мережею YOLO середнього або великого розміру, що забезпечує високу швидкодію детектування зображень автомобілів засобами моделі нейромережі малого розміру та підвищує точність детектування за метрикою mAP50 (до 24 %).

2. Вперше розроблено методику для попередньої обробки зображень, які подаються на входи згорткової нейронної мережі з архітектурою YOLO, особливістю якої є підвищення контрасту зображень запропонованим способом вирівнювання (еквалізації) та центрування гістограм зображень, що призводить до підвищення

точності детектування зображень автомобілів за метрикою перекривання рамок IoU (на 16 %).

3. Подальшого розвитку отримала методика для донавчання різних версій ЗНМ YOLO з використанням датасетів, створених у ручному режимі, особливістю якої є порівняльний аналіз результатів донавчання, що забезпечує цілеспрямований вибір версії YOLO відповідно вимогам до точності, швидкодії та обсягу використаних ресурсів при детектуванні зображень автомобілів.

4. Подальшого розвитку отримала методика для автоматизованого створення набору даних із зображень автомобілів засобами моделі згорткової нейронної мережі YOLO середнього або великого розміру, особливістю якої є селекція унікальних кадрів відеопотоку з використанням перцептивного хешування, що дає змогу суттєво скоротити час формування датасету і отримувати спеціалізовані набори без дублювання даних із мінімальною участю людини-експерта.

Практичне значення отриманих результатів полягає у тому, що розроблені у дисертаційній роботі методики та програмні засоби на мові Python можуть застосовуватися для автоматизованого донавчання нейронних мереж YOLO та для високоточного детектування з їх допомогою зображень автомобілів та інших учасників дорожнього руху в прикладних системах комп’ютерного зору, які застосовуються для оцінки дорожнього трафіку, аналізу зайнятості автомобільних парковок, контролю безпеки дорожнього руху, аналізу дорожньої обстановки та в технологіях автономного водіння. Розроблені програмні засоби штучного інтелекту можуть функціонувати на мобільних обчислювальних пристроях.

Ключові слова: аналіз даних (інтелектуальний аналіз даних), безпілотні літальні апарати (БПЛА), глибоке навчання, дистанційне зондування, інтелектуальна система, класифікація (класифікація даних), класифікація зображень, машинне навчання, регресійний аналіз, лінійна регресія, штучний інтелект, штучні нейронні мережі (ШНМ), YOLO, згорткові нейронні мережі (ЗНМ).

ABSTRACT

Serhii Stets. Improving the accuracy and speed of car image detection using the YOLO convolutional neural network. – Qualification scientific work as a manuscript.

Thesis for the degree of Doctor of Philosophy in the specialty 121 – "Software Engineering" – Yuriy Fedkovych Chernivtsi National University, Chernivtsi, 2026.

Detecting images of vehicles is a basic step for automated remote sensing and control of objects in various Internet of Things (IoT) systems. Cars have a significant impact on road conditions, so improving the efficiency of their detection is a key factor in the reliability of transport systems. Artificial neural networks (ANN), in particular convolutional neural networks (CNN), are widely used for recognizing and detecting images of cars. One of the most effective means of detecting images of objects is CNN with YOLO (You Only Look Once) architecture, which use deep learning technologies.

However, standard YOLO models, especially small ones, often demonstrate insufficient detection accuracy in complex scenes, while large models require significant computing resources, which complicates their use in embedded systems. Therefore, improving the accuracy and speed of vehicle image detection **is a pressing task**.

The goal of this dissertation is to improve the accuracy and speed of car image detection by selecting the architecture and version of convolutional neural networks, preprocessing images, and retraining neural networks with YOLO architecture based on automatically generated datasets.

To achieve this goal, the following **tasks** were solved:

1. Analyze existing neural network architectures, methods, and software tools for detecting images of objects, and based on this analysis, select the architecture and version of the CNN that would meet the requirements for speed and accuracy of detecting images of cars.

2. Develop methods and software tools for preprocessing images by equalizing their histograms and increasing local contrast, which would improve the accuracy of vehicle image detection.

3. Develop methods and software tools for training different versions of YOLO models and analyzing training results, which would make it possible to improve detection accuracy through retraining and select the YOLO version based on the requirements for a specific image analysis task.

4. Develop a methodology and software tools for the automated creation of a dataset of car images using large-scale YOLO models, which would ensure the selection of unique frames from the video stream.

5. Develop a methodology and software tools with a “Teacher-Student” architecture for retraining a small YOLO convolutional neural network based on a dataset created in an automated mode.

6. Implement software implementation of an intelligent system for detecting images of cars using trained YOLO models.

7. Conduct an experimental study of the accuracy and performance of the developed system using the example of object detection in test images.

To achieve this goal, the following **research methods** were used: remote learning methods for convolutional neural networks of YOLO architecture; digital image processing methods for equalizing their histograms and increasing local contrast; methods of data mining (regression analysis, in particular, linear regression); perceptual hashing for selecting unique frames from a video stream; combinatorial optimization methods (Hungarian algorithm); data and image classification methods; computer experiment methods using Python programs for empirical confirmation of the adequacy of the results obtained.

The introduction substantiates the relevance of the dissertation topic, formulates the purpose and objectives of the research, describes the scientific novelty and practical

significance of the results obtained, presents the methods, objects, and subject of the research, indicates the personal contribution of the applicant, and provides data on publications related to the dissertation topic.

The first chapter provides a comprehensive analysis of modern methods and means of processing and detecting images of vehicles. The structure and principles of operation of deep neural networks are examined in detail, and the influence of hyperparameters on the learning process is analyzed. The principles of constructing deep residual networks (ResNet), MobileNet architecture with the use of separate convolutions, and EfficientNet architecture are described. A review of two-stage (Faster R-CNN) and single-stage (YOLO, SSD) object image detectors are provided. The choice of YOLO architecture as the basis for further research is justified, given its accuracy and performance indicators. Existing software tools (Google Cloud Vision, Amazon Rekognition) and libraries (OpenCV) designed for image detection are analyzed.

The second chapter develops and implements in Python a methodology for improving image detection quality through preprocessing of input data. The chapter is devoted to the architectural features of the selected YOLOv8 model and the integration of a preprocessing module into its pipeline. The structure of YOLOv8 is described in detail. Preprocessing of images consists of increasing their contrast in three ways: 1) global histogram equalization; 2) increasing local contrast using the adaptive histogram equalization (CLAHE) method; 3) histogram equalization and centering. The accuracy of car image detection is evaluated using the metrics Precision, Recall, F1-score, and mean accuracy (mAP). It has been established that the proposed method of preprocessing images by equalizing and centering the histogram provides the highest accuracy of car detection. Intelligent data analysis was performed, namely regression and correlation analysis of image detection results to find correlations between detection quality metrics.

The third section develops a methodology for retraining different versions of the YOLO neural network, implements the methodology in software, machine learning of

YOLO models is performed, and its results are analyzed. It describes the process of forming a specialized set of car images, which includes collection, manual annotation of images, and class balancing. Data augmentation consisted of expanding the training sample of images by applying geometric (rotations, reflections, scaling) and coloristic (brightness, saturation, noise addition) transformations. This allowed to increase the size of the dataset from 1542 to 3578 images and prevent model overfitting. Performed retraining of YOLO models previously trained on the COCO dataset. By detecting test images, it was found that retraining allows increasing the detection accuracy according to the Recall completeness metric from 0.58 to 0.91, and according to the mAP50-95 integral metric – from 0.37 to 0.72. A comparative analysis of the effectiveness of two versions of CNNs was performed: YOLOv8m and the latest YOLOv11m. It was found that with comparable detection accuracy (mAP50 ≈ 0.96), the YOLOv11m model has a 22 % smaller weight file size. It was shown that the YOLOv8m model should be chosen to ensure maximum accuracy, and YOLOv11m – for maximum performance.

The fourth section develops a methodology for the automated creation of a dataset of car images using a large YOLO convolutional neural network model. The methodology involves the automatic selection of unique frames from a video stream using the perceptual hashing (pHash) method. This allows evaluating the visual similarity of frames and filtering out duplicates, reducing data redundancy by 88% (from 993 to 121 frames in a test experiment). The detection of similar objects (cars) in images is performed by a Hungarian algorithm, taking into account the IoU detection metric and the root mean square error (RMSE) between image descriptors.

A methodology with a “Teacher-Student” architecture was developed, which consists in retraining a small convolutional neural network YOLO ‘Student’ (YOLOv8n) based on an automatically created specialized image dataset by a medium-sized network YOLO “Teacher” (YOLOv8m). It has been experimentally confirmed that this approach allows increasing the accuracy of the “Student” model according to the mAP50 metric

from 0.6864 to 0.9292. The developed methods of the intelligent system are implemented in a Python program that runs on a Raspberry Pi5 single-board computer hardware platform using containerization technologies (Docker). A web interface for the program has been developed. The results of processing test images showed high accuracy in detecting cars in images obtained by unmanned aerial vehicles (UAV).

The scientific novelty of the results obtained lies in the following:

1. For the first time, a methodology with a “Teacher-Student” architecture has been developed, the feature of which is the retraining of a small-sized YOLO convolutional neural network based on an automatically created specialized image dataset by a medium or large-sized YOLO network, which ensures high-speed detection of car images using a small neural network model and increases detection accuracy according to the mAP50 metric (up to 24 %).

2. For the first time, a method has been developed for preprocessing images that are fed to the inputs of a convolutional neural network with YOLO architecture, which features an increase in image contrast using the proposed method of equalization and centering of image histograms, resulting in an increase in the accuracy of vehicle image detection according to the IoU metric (by 16%).

3. A method for retraining different versions of the YOLO CNN using manually created datasets has been improved, featuring a comparative analysis of retraining results, which ensures a targeted selection of the YOLO version in accordance with the requirements for accuracy, speed, and the amount of resources used in detecting car images.

4. The methodology for automated creation of a dataset of car images using a medium or large YOLO convolutional neural network model has been improved. which features the selection of unique video stream frames using perceptual hashing, which significantly reduces the time required to form a dataset and obtain specialized sets without data duplication with minimal human expert involvement.

The practical significance of the results obtained lies in the fact that the methods and software tools developed in the dissertation in Python can be used for automated retraining of YOLO neural networks and for high-precision detection of images of cars and other road users in applied computer vision systems used for traffic assessment, analyzing car park occupancy, monitoring road safety, analyzing road conditions, and in autonomous driving technologies. The developed artificial intelligence software tools can run on mobile computing devices.

Keywords: data analysis (data mining), unmanned aerial vehicles (UAV), deep learning, remote sensing, intelligent system, classification (data classifier), image classification, machine learning, regression analysis, linear regression, artificial intelligence, artificial neural networks (ANN), YOLO, convolutional neural networks (CNN).

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:
публікації у фахових виданнях України:

1. **Стець С.** Аналіз точності та швидкодії детекції автомобілів за допомогою нейронних мереж YOLOV8 та YOLOV11. *Herald of Khmelnytskyi National University*. Technical Sciences. 2025. Т. 357 (5.2). С. 123-130. https://doi.org/10.31891/2307-5732-2025-357-74.
2. Balovsyak S., **Stets S**. Preprocessing of object images before their detection using YOLO neural network. *Security of Infocommunication Systems and Internet of Things*. 2025. Vol. 3, No. 2, Paper 02002. P. 1-5. ISSN 2786-8443. https://doi.org/10.31861/sisiot2025.2.02002. (Баловсяк С. – наукове керівництво і

редагування; **Стець С.** – розробка методики та програмного забезпечення для попередньої обробки зображень перед їх детектуванням).

3. Баловсяк С., **Стець С.** Автоматизоване створення спеціалізованого датасету для зображень автомобілів. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. Т. 359 (6.2). С. 278-285. https://doi.org/10.31891/2307-5732-2025-359-111. (Баловсяк С. – наукове керівництво і редагування; **Стець С.** – розробка методики та програмного забезпечення для автоматичного створення набору даних та донавчання згорткової нейронної мережі).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

4. Баловсяк С.В., **Стець С.Ю.** Аналіз сучасних методів розпізнавання зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XI Міжнародної науково-практичної конференції (ПІКТ – 2022), м. Чернівці, 10–13 листопада 2022 р. Чернівці: Черн. нац. ун-т, 2022. С. 77-80. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – аналіз сучасних методів розпізнавання зображень автомобілів).
5. Баловсяк С.В., **Стець С.Ю.** Використання модуля Inception для підвищення точності розпізнавання зображень у згорткових нейронних мережах. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*: матеріали міжнародної наукової інтернет-конференції, 8-9 червня 2023 р. Вип. 78. Тернопіль, 2023. С. 30-32. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмного забезпечення з використанням модуля Inception).
6. **Стець С.Ю.** Підвищення точності нейромережного розпізнавання зображень за допомогою модуля Inception. *Проблеми інформатики та комп'ютерної техніки*:

праці XII Міжнародної науково-практичної конференції (ПІКТ – 2023), м. Чернівці, 10-12 листопада 2023 р. Чернівці: Черн. нац. ун-т, 2023. С. 61-63.

7. Баловсяк С.В., **Стець С.Ю.** Детектування зображень пішохідних переходів за допомогою нейронних мереж YOLO. *Проблеми інформатики та комп'ютерної техніки*: праці XIII Міжнародної науково-практичної конференції (ПІКТ – 2024), м. Чернівці, 1–3 листопада 2024 р. Чернівці: Черн. нац. ун-т, 2024. С. 49-51. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмного забезпечення для детектування зображень пішохідних переходів).
8. Баловсяк С.В., **Стець С.Ю.** Попередня обробка зображень об'єктів перед їх детектуванням засобами нейронної мережі YOLO. *Фізико-технологічні проблеми передачі, обробки та зберігання інформації в інфокомунікаційних системах*: матеріали X Міжнародної науково-практичної конференції (ПРЕДТ-2025). 15-17 травня 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 181-182. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмних засобів для попередньої обробки зображень).
9. **Стець С.Ю.** Донавчання нейронної мережі YOLO за допомогою автоматизовано створеного датасету зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XIV Міжнародної науково-практичної конференції (ПІКТ – 2025), м. Чернівці, 13-15 листопада 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 64-66.

Публікації, які додатково відображають наукові результати дисертації:

10. Balovsyak S., Kroitor O., Odaiska Kh., Salem A.B.M., **Stets S.** Car image recognition using convolutional neural network with efficient met architecture. *CEUR Workshop Proceeding*. 2024. Vol. 3675: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, IntelITSIS 2024. P. 182-195

(Scopus). (Balovsyak S. – наукове керівництво, Kroitor O. – аналіз наборів даних, Odaiska Kh. – редагування, Salem A.B.M. – аналіз архітектур згорткових нейронних мереж, **Stets S.** – розробка моделі згорткової нейронної мережі з архітектурою EfficientNet, призначеної для розпізнавання зображень автомобілів).

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AP (англ. Average Precision) – середня точність детектування об’єктів.

API (англ. Application Programming Interface) – інтерфейс програмування додатків.

CLAHE (Contrast Limited Adaptive Histogram Equalization) – адаптивна еквалізація гістограми з обмеженням контрасту.

COCO (англ. Common Objects in Context) – масштабний набір даних для навчання моделей детектування та сегментації об’єктів.

FPS (англ. Frames Per Second) – кількість кадрів за секунду.

HOG (англ. Histogram of Oriented Gradients) – гістограма орієнтованих градієнтів.

IoT (англ. Internet of Things) – Інтернет речей.

IoU (англ. Intersection over Union) – метрика перекриття рамок об’єктів, відношення площі перетину двох рамок до площі їх об’єднання.

LiDAR (англ. Light Detection and Ranging) – технологія вимірювання відстані до об'єкта за допомогою лазерного променя.

mAP (англ. mean Average Precision) – усереднене значення середньої точності по всіх класах об’єктів.

R-CNN (англ. Region-based Convolutional Neural Network) – згорткова нейронна мережа на основі регіонів.

SSD (англ. Single Shot MultiBox Detector) – метод детектування об'єктів за один прохід мережі.

YOLO (англ. You Only Look Once – «Ви дивитеся лише раз») – архітектура згорткової нейронної мережі для детектування об’єктів на зображеннях.

БПЛА – безпілотні літальні апарати.

ЗНМ – згорткова нейронна мереж (англ. convolutional neural networks – CNN).

ІТС – інтелектуальна транспортна система.

ШНМ – штучні нейронні мережі.

ВСТУП

Обґрунтування вибору теми дослідження. Стрімкий розвиток сучасних інтелектуальних транспортних систем і технологій автономного водіння вимагає точних, швидких та надійних методів аналізу дорожньої обстановки [1-6]. Інформація про дорожній рух отримується від різноманітних сенсорів, зокрема, від далекомірів, проте основний обсяг інформації зчитується з цифрових відеокамер і обробляється в системах комп'ютерного зору. Через значний обсяг відеоданих їх аналіз доцільно виконувати автоматично з використанням методів штучного інтелекту, а саме методів розпізнавання (класифікації) та детектування зображень учасників дорожнього руху. У сучасних системах відеоспостереження в одному кадрі, як правило, може міститися кілька об'єктів, тому найбільш поширеним завданням аналізу дорожнього руху є детектування зображень об'єктів (англ. Object Detection): автомобілів, пішоходів, велосипедистів та ін. У процесі детектування визначається просторове положення об'єктів, яке в подальшому застосовується для аналізу дорожньої обстановки [7-10]. Зокрема, детектування зображень транспортних засобів застосовується для дистанційного зондування і моніторингу стану дорожнього руху в різноманітних системах Інтернету речей (англ. Internet of Things – IoT) [11-13]. Під час аналізу відеопотоків у транспортних системах у першу чергу детектуються автомобілі, які суттєво визначають стан дорожньої обстановки.

Для розпізнавання та детектування зображень автомобілів застосовуються штучні нейронні мережі (ШНМ) [14-16], зокрема, згорткові нейронні мережі (ЗНМ), які використовують технології глибокого навчання і є одним з провідних напрямків комп'ютерного зору. Згорткові нейронні мережі (англ. Convolution Neural Network – CNN) демонструють високу точність у задачах класифікації та детектування (виявлення, локалізації) об'єктів завдяки здатності автоматично виділяти ієрархічні ознаки із зображень, що робить їх стійкими до змін освітлення та ракурсу [17-20].

Проте, незважаючи на успіхи існуючих методів машинного навчання та архітектур ЗНМ (таких як YOLO [21], SSD [22] чи R-CNN [23]), основною проблемою залишається баланс між точністю детектування та обчислювальною складністю алгоритмів. Складність детектування зображень автомобілів зумовлена їх значною варіабельністю, різними відстанями та ракурсами отримання зображень, умовами освітлення, щільним дорожнім трафіком, перекриванням об’єктів та ін.

Одним з найбільш ефективних засобів детектування зображень об’єктів є ЗНМ з архітектурою YOLO (англ. You Only Look Once – «Ви дивитеся лише раз»), які характеризуються високою точністю та швидкодією [24-27]. Назва YOLO означає, що нейронна мережа одностадійна, а відповідно володіє високою швидкодією. Висока точність локалізації (детектування) об’єктів мережею YOLO забезпечується завдяки покращеним механізмам виявлення ознак. Мережа YOLO дозволяє виявляти об’єкти не тільки на окремих зображеннях, але й на відео в режимі реального часу [28-29]. Проте, точність просторової локалізації об’єктів на зображеннях за допомогою мереж YOLO у деяких випадках є недостатньо високою.

Тому підвищення точності та швидкодії детектування зображень автомобілів засобами нейронної мережі YOLO є актуальним завданням з наукової та практичної точок зору. Підвищити точність детектування об’єктів засобами YOLO можливо за рахунок попередньої обробки зображень та донавчання ЗНМ на спеціально підібраних наборах даних (датасетах). Швидкодію детектування можливо підвищити за рахунок використання моделей YOLO малого розміру, що дає змогу застосовувати такі ЗНМ у мобільних пристроях або у вбудованих системах.

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення точності та швидкодії детектування зображень автомобілів шляхом вибору архітектури та версії згорткових нейронних мереж, попередньої обробки зображень та донавчання нейронних мереж з архітектурою YOLO на основі автоматизовано створених наборів даних.

Для досягнення мети вирішувались такі **завдання**:

1. Провести аналіз існуючих нейромережевих архітектур, методів та програмних засобів для детектування зображень об'єктів, зокрема, транспортних засобів, і на основі такого аналізу виконати вибір архітектури та версії згорткових нейронних мереж, які б задовольняли вимоги до швидкодії та точності детектування зображень автомобілів, а також до обсягу використаних ресурсів.

2. Розробити методику та програмні засоби для попередньої обробки зображень шляхом еквалізації їх гістограм та підвищення локального контрасту, які б забезпечували підвищення точності детектування зображень автомобілів засобами згорткової нейронної мережі з архітектурою YOLO.

3. Розробити методику та програмні засоби для навчання різних версій моделей YOLO та аналізу результатів навчання, які б давали змогу підвищити точність детектування за рахунок донавчання та вибирати версію YOLO з урахуванням вимог для конкретної задачі аналізу зображень.

4. Розробити методику та програмні засоби для автоматизованого створення набору даних із зображень автомобілів засобами моделей згорткових нейронних мереж YOLO середнього або великого розмірів, які б забезпечували селекцію унікальних кадрів відеопотоку та аугментацію даних.

5. Розробити методику та програмні засоби з архітектурою «Вчитель-Учень» для донавчання згорткової нейронної мережі YOLO малого розміру на основі датасету, створеного в автоматизованому режимі.

6. Виконати програмну реалізацію інтелектуальної системи для детектування зображень автомобілів із використанням навчених моделей YOLO.

7. Провести експериментальне дослідження точності та швидкодії розробленої системи на прикладі детектування об’єктів на тестових зображеннях.

Об’єктом дослідження є процес детектування зображень автомобілів на цифрових зображеннях.

Предметом дослідження є методики та програмні засоби детектування зображень автомобілів із використанням згорткової нейронної мережі YOLO, які забезпечують задані вимоги до точності та швидкодії детектування.

Методи дослідження. У роботі використовувались: методи глибокого навчання для згорткових нейронних мереж архітектури YOLO; методи цифрової обробки зображень для еквалізації їх гістограм та підвищення локального контрасту; методи інтелектуального аналізу даних (регресійного аналізу, зокрема, лінійної регресії); перцептуальне хешування для селекції унікальних кадрів відеопотоку; методи комбінаторної оптимізації (угорський алгоритм); методи класифікації даних та зображень; методи комп'ютерного експерименту з використанням програм на мові Python для емпіричного підтвердження адекватності отриманих результатів та порівняльного аналізу точності, швидкодії та споживаних ресурсів для моделей ЗНМ.

Наукова новизна отриманих результатів полягає в наступному:

1. Вперше розроблено методику донавчання згорткової нейронної мережі YOLO малого розміру для задачі детектування зображень автомобілів, яка базується на автоматизованому формуванні спеціалізованого датасету зображень за допомогою нейронної мережі YOLO більшого розміру в рамках підходу «Вчитель-Учень», що дозволяє підвищити точність детектування до 24 % за метрикою mAP50 при збереженні високої швидкодії.

2. Вперше запропоновано методику попередньої обробки вхідних зображень для моделей згорткових нейронних мереж з архітектурою YOLO, яка включає узгоджене застосування еквалізації та центрування гістограм зображень з метою підвищення їх контрасту, що забезпечує збільшення точності детектування зображень автомобілів на 16 % за метрикою IoU.

3. Подальшого розвитку отримала методика для донавчання різних версій ЗНМ YOLO з використанням датасетів, створених у ручному режимі, особливістю якої є

порівняльний аналіз результатів донавчання, що забезпечує цілеспрямований вибір версії YOLO відповідно вимогам до точності, швидкодії та обсягу використаних ресурсів при вирішенні конкретної задачі детектування зображень автомобілів.

4. Подальшого розвитку отримала методика для автоматизованого створення набору даних із зображень автомобілів засобами моделі згорткової нейронної мережі YOLO середнього або великого розміру, особливістю якої є селекція унікальних кадрів відеопотоку з використанням перцептивного хешування, що дає змогу суттєво скоротити час формування датасету і отримувати спеціалізовані набори без дублювання даних із мінімальною участю людини-експерта.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до планів науково-дослідницьких робіт кафедри програмного забезпечення комп'ютерних систем Чернівецького національного університету імені Юрія Федьковича за держбюджетною тематикою: «Дослідження, моделювання та розробка програмного забезпечення складних динамічних систем» (Державний реєстраційний номер 0121U109232).

Достовірність результатів обумовлюється формуванням релевантних датасетів зображень, поділом їх на навчальну, контрольну та тестові вибірки, контролем навчання згорткових нейронних мереж із використанням метрик якості навчання для навчальної та контрольної вибірок, порівнянням отриманих результатів із результатами програм-аналогів, з відомими теоретичними і практичними даними.

Практичне значення отриманих результатів. Розроблені у дисертаційній роботі методики та програмні засоби на мові Python можуть застосовуватися для автоматизованого донавчання нейронних мереж YOLO та для високоточного детектування з їх допомогою зображень автомобілів та інших учасників дорожнього руху в прикладних системах комп'ютерного зору, які застосовуються для оцінки автомобільного трафіку, аналізу зайнятості автомобільних паркувань, контролю безпеки дорожнього руху, аналізу дорожньої обстановки та в технологіях

автономного водіння. Розроблені програмні засоби штучного інтелекту можуть функціонувати на мобільних обчислювальних пристроях (наприклад, на базі мікрокомп’ютера Raspberry Pi), що значно знижує вартість впровадження інтелектуальних систем для детектування зображень. Як сенсори зображень можуть застосовуватися відеокамери, розміщені на стаціонарних або рухомих платформах, зокрема, на безпілотних літальних апаратах (БПЛА) у системах дистанційного зондування.

Результати дисертаційної роботи впроваджено (додаток В):

1. У ТОВ «ДВА ВІДРА», де розроблене програмне забезпечення використовується для визначення стану паркомісць та положення автомобілів компанії на автомобільній стоянці за їх зображеннями (акт впровадження від 10 грудня 2025 р.).

2. У ТОВ «ТРК А.С.С.», де розроблене програмне забезпечення використовується для детекції автомобілів компанії та визначення стану паркомісць на автомобільній стоянці за їх зображеннями (акт впровадження від 10 грудня 2025 р.).

3. В освітній процес кафедри комп’ютерних систем та мереж Чернівецького національного університету імені Юрія Федьковича в межах навчальних дисциплін: «Методи цифрової обробки зображень», «Комп’ютерні системи штучного інтелекту» та «Системи комп'ютерного зору» (акт впровадження від 23 грудня 2025 р.).

Публікації. За темою дисертації опубліковано 10 робіт, в яких подано результати досліджень. З них в українських фахових виданнях – 3 статі; у збірниках матеріалів міжнародних наукових конференцій – 6 робіт; одна наукова праця додатково відображає наукові результати дисертації.

Особистий внесок здобувача. Усі наукові результати дисертації одержано автором самостійно. У працях, надрукованих у співавторстві, автору належить наступне: [2] – розробка методики та програмного забезпечення для попередньої

обробки зображень перед їх детектуванням; [3] – розробка методики та програмного забезпечення для автоматизованого створення набору даних та донавчання згорткової нейронної мережі; [4] – аналіз сучасних методів розпізнавання зображень автомобілів; [5] – розробка програмного забезпечення з використанням модуля Inception; [7] – розробка програмного забезпечення для детектування зображень пішохідних переходів; [8] – розробка програмних засобів для попередньої обробки зображень; [10] – розробка моделі згорткової нейронної мережі з архітектурою EfficientNet, призначеної для розпізнавання зображень автомобілів.

Апробація матеріалів дисертації. Матеріали дисертаційної роботи обговорювалися на міжнародних наукових конференціях. Зокрема на наукових конференціях: «Проблеми інформатики та комп'ютерної техніки (ПІКТ)» (Чернівці, 2022, 2023, 2024, 2025); «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (Тернопіль, 2023); «Фізико-технологічні проблеми передачі, обробки та зберігання інформації в інфокомунікаційних системах (ПРЕДТ-2025)» (Чернівці, 2025).

Структура та обсяг дисертації. Дисертація складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Дисертаційна робота містить 59 рисунків, 3 таблиці, 8 додатків. Список використаних джерел містить 184 найменувань. Загальний обсяг роботи складає 230 сторінок, обсяг основного тексту – 148 сторінок.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ТА ДЕТЕКТУВАННЯ ЗОБРАЖЕНЬ АВТОМОБІЛІВ

Розпізнавання та детектування зображень транспортних засобів, зокрема, автомобілів, має велике практичне значення для інтелектуального управління дорожнім рухом у різноманітних системах комп’ютерного зору [30-35]. На даний час існує розвинена мережа стаціонарних і мобільних відеокамер, які дають досить детальну відеоінформації про дорожній рух. Програмне розпізнавання автомобілів на зображеннях дозволяє автоматизувати процес аналізу зображень, що забезпечує менші витрати часу для людини-оператора, підвищує точність і швидкодію обробки зображень [36-37]. Потреба в програмних засобах для розпізнавання зображень автомобілів виникає, наприклад, при побудові комп’ютерних систем відеоспостереження та систем паркування автомобілів. Комп’ютерні засоби для розпізнавання зображень автомобілів, пішоходів та інших об’єктів потенційно можуть забезпечити вищу безпеку дорожнього руху, зокрема, при керуванні безпілотними автомобілями. Проте, у загальному випадку завдання виявлення автомобілів є досить складним через зміну зображень об’єктів за рахунок різної відстані до відеокамери, кутів повороту, освітлення, перекривання іншими об’єктами та ін. Сучасні методи детектування зображень автомобілів не завжди забезпечують потрібну точність, тому в даній роботі проведено аналіз таких методів із метою виявлення шляхів їх подальшого удосконалення [6]. Прогрес у сфері штучного інтелекту значно трансформував методи аналізу візуальних даних. Завдання, які були надзвичайно складними для традиційних алгоритмів комп'ютерного зору, на даний час ефективно розв'язуються завдяки застосуванню штучних нейронних мереж (ШНМ) [38-43]. Тому в розділі значна увага приділена розпізнаванню та детектуванню зображень об’єктів засобами ШНМ [15-19].

1.1 Постановка задач розпізнавання та детектування зображень транспортних засобів

Сучасні процеси урбанізації та стрімке зростання рівня автомобілізації створюють безпрецедентне навантаження на дорожню інфраструктуру міст. Забезпечення безпеки дорожнього руху та ефективне керування транспортними потоками стає неможливим без впровадження концепції Інтелектуальних транспортних систем (ITC) та елементів «Розумного міста» (англ. Smart City) [2-4]. Ключовим викликом при проєктуванні таких систем є необхідність обробки великих масивів даних у режимі реального часу.

Перевагою систем комп'ютерного зору є забезпечення комплексного аналізу дорожньої обстановки, оскільки традиційні засоби моніторингу (наприклад, із використанням ультразвукових або магнітних сенсорів) фіксують, як правило, тільки присутність об'єкта. Візуальний канал є найбільш інформативним для аналізу дорожніх ситуацій, проте його використання висуває специфічні вимоги до алгоритмів обробки даних. У цьому аспекті розпізнавання та детектування зображень автомобілів виділяються як окремі науково-технічні проблеми в межах комп'ютерного зору [44-46]. Робота з транспортними засобами у відеопотоці, на відміну від задач розпізнавання статичних об'єктів у контрольованому середовищі, вимагає врахування високої динаміки сцени, значної варіативності форм і розмірів об’єктів, а також складних умов освітлення та ефектів часткового перекриття (оклюзії). Комп’ютерний зір застосовується не лише для детектування транспортних засобів, але й для автоматичного виявлення порушень правил дорожнього руху, що сприяє підвищенню безпеки дорожнього руху [47]. У теорії комп'ютерного зору [44-46] прийнято виділяти такі задачі візуального сприйняття:

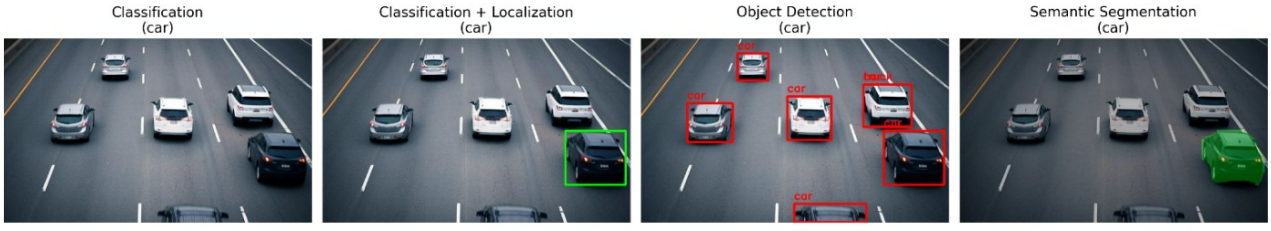
1. Розпізнавання (класифікація) зображень (англ. Image Recognition, Image Classification), яка полягає у встановленні належності зображення об’єкта до певного

класу (наприклад, до класу «автомобілі») (рис. 1.1 а). Задачі розпізнання та класифікації є подібними, проте задача розпізнавання вважається більш загальною, оскільки у випадку розпізнавання правила належності об’єктів до класів не обов’язково вказуються явно. Результатом роботи алгоритму розпізнавання є відповідь на питання «Що зображено?», але при цьому ігнорується просторова структура зображення.

2. Локалізація об’єктів (англ. Object Localization), яка полягає у визначенні просторового положення (координат обмежувальної рамки) та класу для певного об’єкту на зображенні (передбачається, що на зображенні є один основний об'єкт) (рис. 1.1 б). Результатом роботи алгоритму локалізації є відповідь на питання «Де знаходиться об'єкт певного класу на зображенні?».

3. Детектування (виявлення) об’єктів (англ. Object Detection) [48-52], яка полягає просторовій локалізації довільної кількості об'єктів різних класів на одному зображенні (рис. 1.1 в). Це вимагає від алгоритму не лише розпізнати ознаки певного об’єкта (наприклад, автомобіля), а й просторово відокремити один об’єкт від іншого та від фону [53-57]. Результатом роботи алгоритму детектування є відповідь на питання «Де знаходяться об'єкти певних класів на зображенні?».

4. Сегментація об’єктів (англ. Image Segmentation), яка полягає у виділенні ділянок (піксельних масок) на зображеннях, які відповідають об’єктам (рис. 1.1 г).



а) б) в) г)

Рисунок 1.1 – Приклади задач комп'ютерного зору: а) розпізнавання (класифікації); б) локалізації; в) детектування; г) сегментації

Вищенаведені задачі комп'ютерного зору розміщено у порядку зростання їх складності. Задачі локалізації та детектування об'єктів на зображеннях часто не розділяють, а розглядають як задачу детектування довільної кількості об'єктів на зображенні. Перевагою детектування є здатність не тільки фіксувати присутність об'єкту певного класу, але також точно локалізувати та ідентифікувати кожен транспортний засіб, що є фундаментом для вирішення прикладних задач вищого рівня: від автоматичної фіксації порушень правил дорожнього руху до навігації безпілотних автомобілів. Тому в даній роботі розробляється система для детектування (виявлення) зображень автомобілів, в якій задача розпізнавання є частковою і полягає у визначенні класу об'єкта у межах детектованої ділянки на зображенні [43].

Задачу розпізнавання (класифікації) зображень можна формалізувати як встановлення належності вхідного зображення до певного класу з фіксованого набору (множини) класів [48-50, 58]. Математично задача розпізнавання зводиться до знаходження функції $f(I) \to c$, де I – вхідне зображення, $c \in C$ (набір класів).

У процесі вирішення задачі детектування [42-43] для кожного об'єкта на зображенні визначається (рис. 1.2):

1. Клас об'єкту (наприклад, «автомобіль», «пішохід»).
2. Прямокутна обмежувальна рамка (англ. Bounding Box), яка просторово локалізує об'єкт і описується координатами (x, y, w, h), де x, y – координати центру рамки (або її лівого верхнього кута), а w, h – ширина та висота рамки відповідно.

У контексті автономних транспортних систем та систем відеоспостереження важливим є не лише детектування, а й відстеження (англ. Tracking) об'єктів у часі [59]. Проте базовим етапом відстеження залишається саме детектування, яке має бути стійким до змін масштабу, оклюзії (перекриття) об'єктів та зміни умов освітлення. Важливим аспектом вирішення задачі детектування є суперечність між точністю виявлення об'єктів та обчислювальною складністю алгоритму.

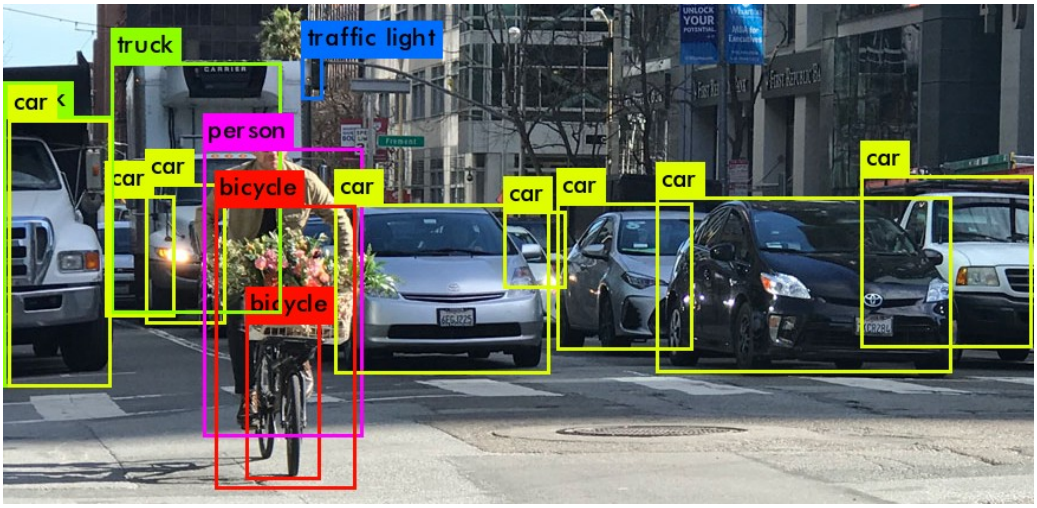


Рисунок 1.2 – Приклад детектування зображення учасників дорожнього руху

Для систем автопілотування або моніторингу дорожнього трафіку затримка обробки кадру навіть у 100-200 мс може бути неприпустимою, оскільки за цей час дорожня ситуація суттєво зміниться. Вимоги до швидкодії накладають жорсткі обмеження на архітектуру нейронної мережі: вона повинна забезпечувати високу точність детектування об’єктів, обробляючи при цьому зображення з частотою кадрів за секунду (англ. Frames per Second – FPS) не нижче 30 на доступному обладнанні. Тому в більшості існуючих рішень надається перевага високій точності детектування або високій швидкодії (обробка в реальному часі).

Додатковим викликом, який виникає при виявленні об’єктів, є значна варіативність їх масштабу. Система повинна точно детектувати як автомобіль, що знаходиться безпосередньо перед камерою, так і транспортні засоби на значній відстані. В той же час втрата інформації про дрібні об'єкти під час операцій згортки є типовою проблемою класичних архітектур ЗНМ.

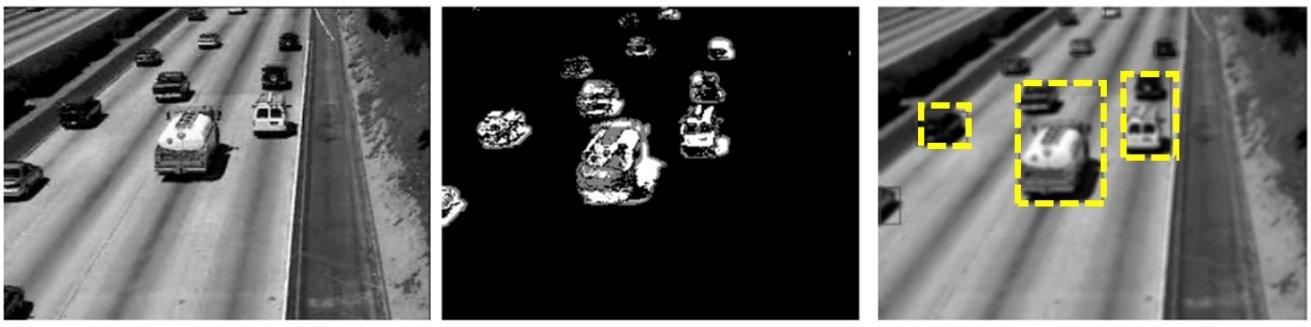
Таким чином, у рамках даного дослідження основна увага буде зосереджена на методах і засобах детектування зображень автомобілів та інших учасників дорожнього руху, оскільки саме результати детектування забезпечують необхідний рівень інформативності для прикладних задач моніторингу дорожнього трафіку.

1.2 Еволюція методів детектування об'єктів

Методи детектування зображень [42-43] об’єктів аналізують: 1) окремі (статичні) зображення; 2) відеопотік (рух об’єкта виявляється через різницю кадрів). Існують такі методи аналізу відеопотоку: віднімання фону, використання різниці відеокадрів та оптичного потоку. Метод відніманні фону заснований на тому, що на основі серії відеокадрів визначається нерухомий фон зображення, а рухомі об’єкти виявляються через різницю початкових кадрів з фоном.

У методі різниці відеокадрів обчислюється дисперсія яскравості пікселів на основі значень пікселів двох або трьох послідовних відеокадрів, а області рухомих об’єктів виявляються як ділянки зображень, для яких дисперсія яскравості пікселів перевищує заданий поріг (рис. 1.3). Для підвищення точності детектування рівень шуму на зображеннях зменшується шляхом фільтрації. За допомогою методу різниці відеокадрів можливо також виявити зупинку транспортного засобу.

Метод оптичного потоку [51] дає змогу виявити рухому область на відео. Згенероване поле оптичного потоку визначає напрямок руху для кожного пікселя та його швидкість. Такий метод використовується у сучасних системах аналізу дорожнього руху та в системах відеоспостереження.



а) б) в)

Рисунок 1.3 – Приклад розпізнавання автомобілів через різницю відеокадрів: а) початковий кадр; б) області з рухомими об’єктами; в) кадр зображення з ділянками автомобілів, виділеними прямокутниками [52]

Широке застосування отримали методи виявлення автомобілів із використанням ознак транспортного засобу: метод масштабно-інваріантного перетворення ознак (англ. Scale Invariant Feature Transform – SIFT) та метод прискорених стійких ознак (англ. Speeded Up Robust Features – SURF) [49].

Деякі методи розпізнавання будують 3D-моделі об’єктів [18], що дозволяє точніше оцінити їх форму та розміри. Тривимірні моделі будуються не тільки на основі серії зображень методом фотограмметрії, але й за допомогою лазерних далекомірів (LIDAR) [6].

Еволюцію методів комп’ютерного зору, призначених для детектування об’єктів, прийнято поділяти на два основні етапи: традиційні методи (англ. Traditional Computer Vision) та нейромережні методи, зокрема, методи глибокого навчання (англ. Deep Learning).

1.2.1 Традиційні методи обробки зображень

До початку ери глибокого навчання та масового впровадження згорткових нейронних мереж (2012 рік) задача детектування зображень об'єктів здебільшого вирішувалася за допомогою підходу, що базувався на «ручному конструюванні ознак» (англ. Hand-crafted features). Ця парадигма передбачала двоетапний процес: спочатку експерт розробляв математичний опис локальних ознак зображення (кути, грані, перепади яскравості), а потім ці ознаки передавалися на вхід класичному класифікатору. Як класифікатори застосовувалися, зокрема, метод опорних векторів SVM та адаптивний ансамблевий метод AdaBoost (англ. adaptive boosting).

Фундаментальним підходом до локалізації об'єктів на зображеннях у класичних методах була техніка ковзного вікна (англ. Sliding Window). Алгоритм переміщував вікно фіксованого розміру по всьому зображенню з певним кроком за висотою та шириною, вирізаючи фрагменти та класифікуючи їх як «об'єкт» або

«фон». Для детектування об'єктів різного розміру (наприклад, автомобіля на передньому та задньому плані) використовувалася піраміда зображень – серія копій вхідного кадру, зменшених у різну кількість разів. Головним недоліком цього методу була надзвичайна обчислювальна надлишковість та низька швидкість роботи [53].

Одним із перших алгоритмів, що дозволив наблизитися до роботи в реальному часі, став метод Віоли-Джонса (англ. Viola-Jones), запропонований ще у 2001 році [54-55]. Метод базується на використанні ознак Хаара (англ. Haar-like features) – прямокутних примітивів, що описують різницю інтенсивності пікселів у суміжних зонах зображення. Виявлення прямокутних ділянок шуканих об’єктів виконується каскадним класифікатором на основі ознак Хаара. Спочатку метод Віоли-Джонса розроблявся для детектування обличь [56], але в подальшому був адаптований і для детектування автомобілів. У сучасних дослідженнях зазначається, що каскадні класифікатори досі залишаються актуальними для систем з обмеженими обчислювальними ресурсами, де неможливе використання графічного процесора (англ. Graphics Processing Unit – GPU) [53].

Найбільш ефективним класичним методом для детектування зображень пішоходів та автомобілів став метод гістограми орієнтованих градієнтів HOG (англ. Histogram of Oriented Gradients) у поєднанні з лінійним класифікатором – методом опорних векторів SVM (англ. Support Vector Machine) [57]. Сутність методу полягає у розбитті зображення на невеликі комірки та обчисленні гістограми напрямків градієнтів (змін яскравості) для кожної з них. Оскільки форма автомобіля характеризується чіткими геометричними лініями та контурами, HOG дозволяє сформувати стійкий дескриптор об'єкта, інваріантний до незначних змін освітлення.

Згідно з дослідженнями [57-58], комбінація HOG + SVM демонструє прийнятну точність детектування, проте має суттєві обмеження:

1. Відсутність семантичного розуміння, оскільки метод аналізує лише на локальні перепади контрасту і не враховує контекст сцени.

2. Чутливість до оклюзії – у випадку часткового перекривання об’єкту гістограма градієнтів спотворюється, що призводить до пропуску об'єкта.

3. Ручне налаштування – вибір параметрів (розмір комірки, кількість орієнтацій) вимагає експертного втручання і погано масштабується на нові класи.

Традиційні методи детектування, такі як метод масштабно-інваріантного трансформування ознак SIFT (англ. Scale-Invariant Feature Transform), метод пришвидшених стійких ознак SURF та HOG, вимагали від розробника глибокого розуміння предметної області для вибору релевантних характеристик зображення [59]. Процес розпізнавання зазвичай складався з двох етапів: вилучення ознак та класифікації за допомогою алгоритмів машинного навчання, наприклад, методу опорних векторів (SVM) або випадкового лісу (Random Forest) [59]. У процесі детектування визначалося просторове положення вікна, в якому локалізувався розпізнаних об’єкт. Головним недоліком таких класичних підходів була низька узагальнююча здатність у неконтрольованих умовах. Зміни освітлення, деформації об’єктів та складний фон призводили до значного падіння точності детектування. Процес детектування значно спростився при застосуванні ШНМ.

1.2.1 Детектування об’єктів на зображеннях засобами штучних нейронних мереж

Сучасний етап детектування зображень об’єктів характеризується застосуванням багатошарових ШНМ, для тренування яких використовуються методи глибокого навчання. Ключовою відмінністю цього підходу є здатність нейронних мереж автоматично отримувати ієрархії ознак об’єктів безпосередньо з непідготовлених ("сирих") даних (raw data) [60-66]. В ШНМ не потрібен ручний підбір дескрипторів, оскільки модель ШНМ самостійно формує фільтри, які в початкових шарах реагують на прості примітиви зображення (лінії, кути), а в

глибоких шарах – на складні семантичні об’єкти [67]. Широке застосування ШНМ зумовлене трьома ключовими факторами: появою великих анотованих наборів даних (таких як ImageNet, COCO), значним зростанням обчислювальної потужності графічних процесорів (GPU) та вдосконаленням алгоритмів навчання багатошарових ШНМ.

Фундаментальна відмінність багатошарових (глибоких) ШНМ полягає у відмові від жорстко заданих дескрипторів (як у методах HOG чи SIFT). Такий підхід, відомий як навчання "від кінця до кінця" (англ. end-to-end learning), дозволив досягти значного прориву в точності сегментації та детектування зображень. Глибокі нейронні мережі демонструють стійкість до шуму та здатність обробляти візуальну інформацію на рівні, що наближається до людського або перевищує його, за умови наявності достатнього обсягу розмічених даних для тренування [67]. Суттєвою перевагою ШНМ є можливість навчання за прикладами, тобто за множиною зображень навчальної вибірки. Саме ці переваги зумовили домінування ШНМ, зокрема ЗНМ, у задачах розпізнавання зображень транспортних засобів.

Згорткові нейронні мережі (англ. Convolutional Neural Network – CNN) є поширеною архітектурою багатошарових ШНМ, яка використовує технології глибинного навчання [68-72]. ЗНМ особливо ефективні в комп’ютерному зорі при розпізнаванні зображень, оскільки враховують геометрію зображень, особливості зорового сприйняття людини, використовують багатошарову обробку сигналів [73-75]. У порівнянні з повнозв’язними ШНМ (багатошаровими перцептронами) ЗНМ потребують менше обчислювальних ресурсів, що забезпечує їх високу швидкодію. Це зумовлено тим, що в ЗНМ кожен нейрон наступного шару пов’язаний тільки з невеликою частиною нейронів попереднього шару (у межах ядра згортки). ЗНМ використовують три ключові архітектурні ідеї: локальні рецептивні поля, розділення ваг та підвибірку [76].

Проте, при детектуванні зображень автомобілів із використанням існуючих архітектур ЗНМ виникають труднощі, які пов’язані з низькими швидкістю навчання ЗНМ та точністю детектування. Значний час навчання ЗНМ пояснюється великою кількістю шарів нейронної мережі та значним розміром навчальної вибірки. У загальному випадку точність детектування зменшується через неповноту навчальної вибірки, невідповідність архітектури та параметрів навчання ЗНМ до особливостей зображень об’єктів, які виявляються. Тому завдання підвищення точності детектування зображень автомобілів за допомогою ЗНМ є актуальним.

1.3 Сучасні архітектури штучних нейронних мереж для детектування зображень об’єктів

Розвиток багатошарових (глибоких) ШНМ [66-70] призвів до виникнення широкого спектру архітектур ШНМ, призначених для детектування зображень об’єктів [77-79]. У сучасній науковій літературі їх прийнято класифікувати на основі наявності етапу генерації регіонів-кандидатів. За цим критерієм виділяють два основні типи ШНМ: двостадійні (англ. two-stage) та одностадійні (англ. one-stage) детектори. Двостадійний метод спочатку генерує область-кандидат об'єкта на зображенні, а потім розпізнає об'єкт у межах області за допомогою ЗНМ. Одностадійний метод не генерує область-кандидат, а безпосередньо детектує об’єкт на зображенні. Як окремий напрям розвиваються архітектури ШНМ на основі трансформерів (англ. Transformer-based) [23].

1.3.1 Глибокі залишкові мережі

Архітектура глибоких залишкових мереж ResNet (англ. Residual Network), стала фундаментальним проривом у розробці глибоких нейронних мереж,

дозволивши ефективно тренувати моделі глибиною понад 100 шарів [44]. До появи ResNet в багатошарових ШНМ існували проблеми зникаючого градієнта та деградації точності: при додаванні нових шарів до вже глибокої мережі точність навчання не зростала, а навпаки, починала падати [80]. Автори ResNet висунули гіпотезу, що оптимізувати залишкову функцію відображення сигналів між шарами ШНМ легше, ніж оптимізувати оригінальне, незміщене відображення сигналів. Відповідно до цієї гіпотези розроблено структуру залишкового блоку, який базовим елементом ResNet (рис. 1.4).

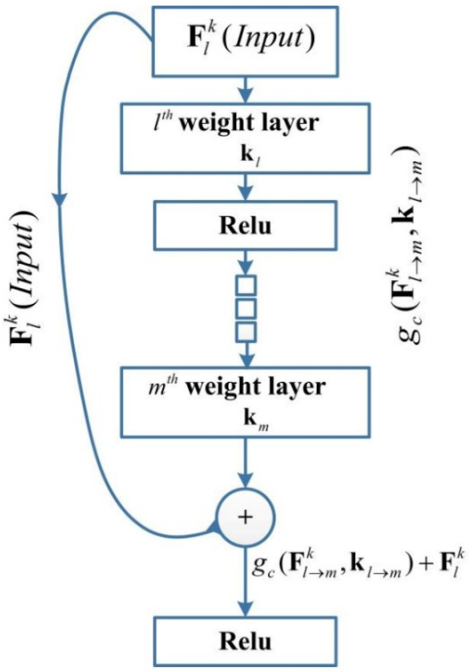


Рисунок 1.4 – Залишковий блок як основна структурна одиниця ResNet [81]

Залишковий блок реалізує механізм навчання залишкової функції (англ. residual learning) через паралельну обробку вхідних даних двома шляхами. Процес перетворення інформації у блоці можна описати наступними компонентами:

1. Вхідні дані (F_l^k). У верхній частині схеми представлено вхідний тензор ознак F_l^k, де індекси l та k позначають відповідний шар мережі та канал (або ядро згортки).

2. Основний шлях перетворення даних (англ. Main Path). Центральна вертикальна гілка схеми відображає послідовність нелінійних перетворень вхідного сигналу, що формалізується як залишкова функція $g_c\left(F_{l\to m}^k, k_{l\to m}\right)$. Основний шлях містить:

- Ваговий шар l (k_l), в якому виконується перший етап згортки в межах блоку.
- ReLU (англ. Rectified Linear Unit, випрямлений лінійний вузол) – функція активації нейронів, що вносить нелінійність у процес обробки сигналів.
- Проміжні операції позначені квадратами, які означають можливість присутності додаткових шарів нормалізації або перетворень між шарами l та m.
- Ваговий шар m (k_m) – завершальний шар вагової обробки перед операцією сумування.

3. Зв'язок швидкого доступу (англ. Skip Connection / Shortcut), який реалізує механізм тотожного відображення (англ. identity mapping) (зв’язок показано лівою вигнутою стрілкою). Цей зв'язок передає вхідний сигнал F_l^k безпосередньо до виходу блоку, минаючи вагові шари. Даний архітектурний елемент є важливим для нівелювання проблеми зникаючого градієнта, забезпечуючи безперешкодне поширення сигналу помилки під час навчання ШНМ.

4. Агрегація сигналів, яка реалізується по елементним додаванням сигналів (позначена оператором $\bigoplus$). Вихідний сигнал U залишкового блоку формується шляхом суперпозиції перетворених ознак та оригінального входу за формулою:

$$U = g_c\left(F_{l\to m}^k, k_{l\to m}\right) + F_l^k. \quad (1.1)$$

Такий підхід дозволяє мережі навчатися лише змінам (залишку) відносно вхідного сигналу, а не відтворювати повне відображення сигналу. Це дозволяє градієнту помилки під час зворотного поширення (англ. backpropagation) проходити безпосередньо до попередніх шарів без суттєвого затухання.

В глибоких ШНМ з архітектурою ResNet (ResNet-50, ResNet-101, ResNet-152), які часто застосовуються для детектування зображень об’єктів (зокрема, автомобілів), використовується більш економна структура залишкового блоку, яка називається "Bottleneck" (пляшкове горло). У контексті розпізнавання автомобілів це важливо, оскільки дозволяє ШНМ формувати складну ієрархію ознак об’єктів (від країв кузова до специфічних форм фар та решіток радіатора) без критичного навантаження на обчислювальні ресурси [80].

1.3.2 Штучні нейронні мережі з архітектурою MobileNet

Застосування ЗНМ з архітектурою MobileNet є особливо доцільним в системах з обмеженими обчислювальними ресурсами та енергоспоживанням, зокрема, у вбудованих та мобільних системах. Особливістю MobileNet є орієнтація на максимізацію співвідношення точності детектування до продуктивності системи.

Фундаментальною інновацією архітектури MobileNet є заміна стандартної операції згортки на роздільну за глибиною згортку (англ. Depthwise Separable Convolution). Така операція розділяє стандартну згортку на два послідовні етапи:

1. Глибинна згортка (англ. Depthwise Convolution) – застосовується один фільтр до кожного вхідного каналу окремо (просторова фільтрація).

2. Точкова згортка (англ. Pointwise Convolution) – застосовується згортка 1×1 для об’єднання виходів з глибинної згортки (лінійна комбінація каналів) [80].

Оцінимо обчислювальну складність роздільної згортки. Нехай вхідний тензор має розмір $D_F \times D_F \times M$, де D_F – просторова розмірність, M – кількість каналів. Вихідний тензор має N каналів. Розмір ядра згортки – $D_K \times D_K$. У такому випадку кількість операцій стандартної згортки C_{std} дорівнює:

$$C_{std} = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F. \quad (1.2)$$

Кількість операцій роздільної згортки C_{sep} складається з суми кількості операцій глибинної та точкової згорток:

$$C_{sep} = \underbrace{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F}_{\text{Dephwise}} + \underbrace{M \cdot N \cdot D_F \cdot D_F}_{\text{Pointwise}} \tag{1.3}$$

Коефіцієнт зменшення кількості операцій становить:

$$\frac{C_{sep}}{C_{std}} = \frac{D_K^2 \cdot M \cdot D_F^2 + M \cdot N \cdot D_F^2}{D_K^2 \cdot M \cdot N \cdot D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \tag{1.4}$$

При типовому розмірі ядра згортки 3×3 ($D_K = 3$) ЗНМ MobileNet зменшує кількість обчислень у 8–9 разів порівняно зі стандартною згорткою, а точність обробки сигналів при цьому зменшується незначно. За рахунок цього MobileNet дозволяє обробляти відеопотік в реальному часі (з частотою понад 30 FPS) навіть на процесорах мобільних пристроїв [80].

1.3.3 Детектування зображень із використанням модуля Inception

При практичних задач детектування зображень об’єктів засобами ЗНМ часто виникає потреба у підвищенні точності. Одним з способів підвищення точності ЗНМ є збільшення кількості параметрів нейронної мережі, але це може призвести до перенавчання та зменшення точності детектування для зображень, що не належать до навчальної вибірки. Ризик перенавчання ЗНМ можна зменшити завдяки використанню модуля Inception [82-83], який також дозволяє зменшити кількість параметрів у нейронній мережі [84].

Вперше модуль Inception було продемонстровано як компонент в архітектурі нейронної мережі GoogLeNet, яка є одним з найбільш відомих та успішних проєктів

у галузі комп'ютерного зору. Застосування модуля Inception дозволило досягти кращих результатів класифікації зображень на деяких популярних наборах даних, зокрема, ImageNet. Архітектура GoogLeNet стала переможцем у конкурсі ILSVRC 2014 з класифікації зображень. Вона забезпечила значне зниження частоти помилок порівняно з попередніми переможцями AlexNet. Загальна архітектура GoogLeNet містить 22 шари та два допоміжні шари класифікатора, з'єднані з виходами шарів Inception (4a) та Inception (4d). Модуль Inception використовує різні розміри фільтрів та шарів підсумування для отримання різноманітних ознак зображення. Це дає змогу забезпечити вищу точність розпізнавання, знизити кількість параметрів та запобігти перенавчанню ЗНМ. Кожен модуль Inception складається з кількох згорткових (CONV) та пулінгових шарів (шарів підвибірки, MAXPOOL) з різними розмірами та кількістю фільтрів (рис. 1.5). Ідея застосування модуля Inception полягає у тому, щоб знайти оптимальну локальну розріджену структуру в ЗНМ [32]. В модулі Inception вхідні дані (Previous Activation) проходять кілька шляхів обробки, які називаються гілками і містять певний набір фільтрів для обробки даних. Такий підхід дозволяє мережі вибирати оптимальний розмір ядра для кожної частини зображення при обчисленні вихідних значень (англ. Channel Concat) [85, 86].

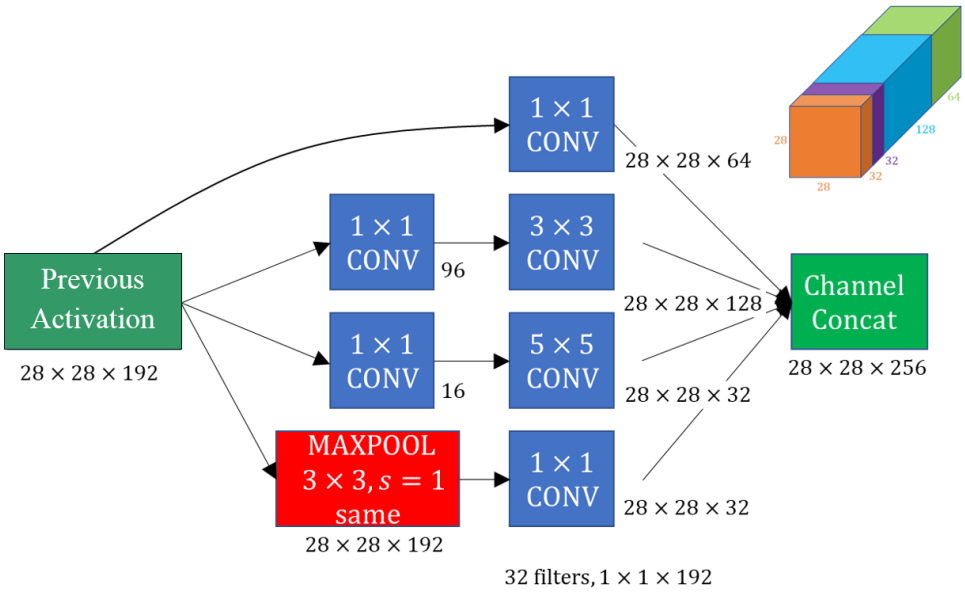


Рисунок 1.5 – Приклад структури модуля Inception [85]

Удосконалити архітектуру Inception можна шляхом застосування нових технік, таких як ефективні засоби регуляризації та оптимізації, що реалізується в архітектурі Inception-v4. Внутрішній механізм модуля Inception полягає в тому, що використовується 3 типи згорткових фільтрів (з розмірами 1×1, 3×3, 5×5 елементів) та оператор шару підвибірки MAXPOOL з ядром 3×3. Для кожного з цих типів фільтрів виконуються операції згортки над вхідним тензором inputs. Результатом таких операцій є три тензори, що містять результати фільтрів 1×1, 3×3, 5×5 відповідно та один тензор, що містить вихід оператора MAXPOOL. Далі ці чотири тензори об'єднуються в один тензор за допомогою функції concatenate. Отриманий результат є виходом модуля Inception, який можна використовувати для створення ЗНМ, зокрема, заснованих на архітектурі GoogLeNet [84].

Вищеописаний підхід з використанням модуля Inception дозволив досягти вищої точності при класифікації зображень за допомогою ЗНМ. Це було підтверджено при розпізнаванні зображень поширених наборів даних, зокрема, набору ImageNet. Також перевагою модуля Inception є зменшення часу навчання ЗНМ, що особливо важливо для «глибоких» нейронних мереж [86].

1.3.4 Двостадійне детектування об'єктів на зображеннях згортковими нейронними мережами

Двостадійні методи історично першими продемонстрували високу точність у задачах детектування зображень, в яких процес обробки розділений на дві послідовні стадії (етапи):

1. Генерація пропозицій регіонів – ЗНМ виділяє області зображення, в яких з високою ймовірністю знаходиться об'єкт.

2. Класифікація та уточнення координат – виділені регіони нормалізуються та класифікуються, а їхні обмежувальні прямокутні рамки коригуються.

Прикладом двостадійних детекторів є ЗНМ з архітектурою Faster R-CNN [44]. Архітектура Faster R-CNN стала першою системою детектування зображень, яка об'єднала вилучення ознак, генерацію пропозицій регіонів та класифікацію в єдиний конвеєр навчання. Ключовим нововведенням стала мережа пропозицій регіонів (англ. Region Proposal Network – RPN) [23]. RPN є повністю згортковою мережею, яка приймає на вхід карту ознак і видає набір прямокутних рамок (пропозицій) об'єктів. Для виявлення об'єктів різного масштабу RPN використовує механізм якорів. У кожній позиції ковзного вікна на карті ознак одночасно передбачається k регіонів-кандидатів (якорів). Такі якорі мають різні масштаби та співвідношення сторін, наприклад, $1{:}\,1, 1{:}\,2, 2{:}\,1$. Навчання мережі пропозицій регіонів RPN відбувається шляхом мінімізації мультизадачної функції втрат. Для кожного якоря з індексом i функція втрат обчислюється за формулою:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \qquad (1.5)$$

де: p_i – передбачена ймовірність того, що якір є об'єктом;

p_i^* – істинна мітка: 1 – якщо якір позитивний; 0 – якщо якір негативний; якір позитивний, якщо перетин якоря з ділянкою об’єкта більше порогу (наприклад, 0.7); в іншому випадку якір негативний;

t_i – вектор із 4-х параметризованих координат передбаченої рамки;

t_i^* – вектор правильних (істинних) координат рамки;

λ – коефіцієнт балансу між двома доданками функції (наприклад, λ = 1);

N_{cls} – розмір пакету даних; N_{reg} – кількість якорів;

L_{cls} – логарифмічна функція втрат для класифікації (об'єкт / фон);

L_{reg} – функція втрат регресії, яка активується лише для позитивних якорів.

Архітектура Faster R-CNN забезпечує інваріантність до масштабу та трансляції об'єкта. Подальший розвиток ідеї Faster R-CNN отримали в архітектурах Cascade R-CNN та DetectoRS, які використовують рекурсивні піраміди ознак (англ. Recursive Feature Pyramid) для покращення детектування об’єктів різного масштабу [23]. ШНМ Faster R-CNN можуть ефективно застосовуватися не тільки для детектування зображень автомобілів, але також їх деталей [44, 87-88].

Головною перевагою двостадійних детекторів є висока точність локалізації та здатність ефективно працювати зі щільними скупченнями об’єктів на зображеннях. Однак їхнім суттєвим недоліком є низька швидкість роботи (зазвичай менше 10-15 кадрів на секунду), що обмежує їх використання в системах реального часу. Сучасні архітектури ЗНМ в цьому напрямку, такі як мережі з узгодженістю вибірок SCNet (англ. Sample Consistency Network), зосереджені на покращенні узгодженості між навчальними вибірками та тестовими вибірками, що дозволяє підвищити точність детектування зображень об’єктів [89].

1.3.5 Детектування зображень об’єктів із використанням рекурсивних пірамід ознак

Одним із потужних сучасних рішень для детектування зображень об’єктів є ЗНМ з архітектурою DetectoRS [90], в якій розвинуто ідею застосування піраміди ознак об’єкту. В архітектурі DetectoRS впроваджено два ключові механізми:

1. Рекурсивна піраміда ознак (англ. Recursive Feature Pyramid – RFP), яка використовує зворотні зв’язки. Вихідні карти ознак з виходу RFP подаються назад у "хребет" (англ. backbone) нейронної мережі. Це дозволяє нейромережі аналізувати зображення багатократно.

2. Перемикальна розширена (дилатаційна) згортка (англ. Switchable Atrous Convolution – SAC), яка застосовується для адаптації зображень об'єктів до різного

масштабу (що важливо для зображень автомобілів, отриманих з різної відстані). В SAC використовується механізм, який динамічно змінює розмір ядра згортки. Математично SAC можна описати як зважену суму двох згорток (англ. Conv) із різними коефіцієнтами розширення $\Delta r_1, \Delta r_2$ для ядер згорток:

$$Conv(x, w, r) = S(x) \cdot Conv(x, w, \Delta r_1) + \big(1 - S(x)\big) \cdot Conv(x, w, \Delta r_2) \qquad (1.6)$$

де x – вхідний сигнал ЗНМ; w – ваги ЗНМ;

$S(x)$ – функція перемикання, яка залежить від просторового положення x.

Такий підхід дозволив DetectoRS досягти рекордних показників точності детектування зображень за метрикою mAP на датасеті COCO [90].

1.3.6 Одностадійне детектування об’єктів на зображеннях

На відміну від двостадійних, одностадійні детектори розглядають задачі детектування (локалізації) та класифікації як єдину задачу. Це дозволяє позбутися "вузького місця" двостадійних детекторів у вигляді генерації пропозицій регіонів, що забезпечує значний приріст швидкості обробки та можливість роботи в режимі реального часу (>30 FPS). Фундаментальна ідея, закладена в основу одностадійних детекторів, полягає у поділі вхідного зображення на сітку розміром $S \times S$ клітин (елементів). Для кожної клітинки сітки мережа передбачає B обмежувальних рамок та вектор ймовірностей розпізнавання класів C. Якщо центр об'єкта потрапляє в певну клітинку, то вона стає відповідальною за його детектування.

Одним з найбільш перспективних одностадійних детекторів зображень з ЗНМ з архітектурою YOLO (англ. You Only Look Once – «ти бачиш тільки раз») [21, 25 - 29]. Назва YOLO означає, що зображення об’єктів детектуються за одну стадію. Перевагою YOLO є висока точність детектування об’єктів, яка поєднується також з

високою швидкодією (можлива обробка відеопотоку в режимі реального часу). На даний час розроблено ряд версій YOLO, кожна з яких пропонує кілька розмірів моделей ЗНМ. Таким чином, для певної задачі детектування зображень об'єктів можливо вибрати відповідну їх версію та розмір моделі YOLO.

1.3.7 Детектування зображень із використанням компаундного масштабування

Традиційно для підвищення точності детектування зображень об'єктів засобами ЗНМ виконували масштабування одного з вимірів: глибини (кількості шарів, як у ResNet), ширини (кількості каналів) або роздільної здатності зображення. Ці виміри є залежними, тому з урахуванням такої залежності запропоновано метод компаундного масштабування (англ. Compound Scaling) [91]. Метод формалізує масштабування вимірів ЗНМ через єдиний коефіцієнт ϕ, який задає доступні обчислювальні ресурси. Глибина d, ширина w та роздільна здатність r масштабуються за формулою:

$$d = \alpha^{\phi}, w = \beta^{\phi}\ ,\ r = \gamma^{\phi}, \tag{1.7}$$

де α, β, γ – константи, знайдені методом ґратчастого пошуку (grid search).

Значення констант α, β, γ повинні задовольняти обмеження:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2,\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1. \tag{1.8}$$

Компаундне масштабування гарантує, що нейронна мережа використовує ресурси оптимально, рівномірно нарощуючи "потужність" за всіма вимірами. Економне використання ресурсів є особливо важливим для мобільних пристроїв із обмеженою продуктивністю [91].

1.3.8 Детектування зображень об’єктів засобами трансформерів

Поява у 2017 році нейромережевої архітектури трансформер (англ. Transformer) революціонізувала обробку природної мови, а з 2020 року трансформери почали активно застосовуватися в комп’ютерному зорі. Піонером став метод детектування засобами трансформерів DETR (англ. Detection Transformer), який запропонував відмовитися від ручних евристик, таких як якорі та придушення не максимумів. Використовуючи механізм самоуваги (англ. self-attention), трансформер здатний аналізувати взаємозв’язки між частинами зображення, що корисно для розуміння контексту дорожньої сцени [92].

Розвиток цього напрямку продовжила архітектура Swin Transformer, в якій впроваджено ієрархічну структуру з механізмом зсуву вікон. Це дозволило трансформерам ефективно обробляти зображення високої роздільної здатності та стати потужною альтернативою класичним ЗНМ [93]. Проте, у порівнянні з ЗНМ трансформери зазвичай вимагають більших обсягів даних для навчання.

1.4 Аналіз програмних засобів для детектування зображень об’єктів

Задачі детектування зображень транспортних засобів вирішуються за допомогою широкого спектру програмних засобів (інструментів). Для обґрунтування вибору методів та засобів власного дослідження проведено детальний аналіз існуючих рішень-аналогів.

1.4.1 Обробка та детектування зображень засобами бібліотеки OpenCV

Бібліотека комп’ютерного зору з відкритим програмним кодом OpenCV (англ. Open-Source Computer Vision Library) залишається фундаментальним інструментом

для вирішення широкого спектра задач комп'ютерного зору [33, 94]. Бібліотека OpenCV написана мовою C++, що забезпечує високу продуктивність обчислень, проте має зручні обгортки (англ, wrappers) для мови Python. Це дозволяє поєднувати швидкість виконання низькорівневого коду з гнучкістю високорівневої розробки [95]. У контексті підготовки даних для ШНМ бібліотека OpenCV відіграє ключову роль на етапі попередньої обробки. OpenCV надає ефективні інструменти для наступних видів обробки зображень:

- Геометричні перетворення – зміна розміру початкового зображення до розмірів вхідних зображень ШНМ (наприклад, 640 × 640 пікселів).
- Колірна корекція – конвертації колірних просторів (зокрема, перетворення зі стандартного для OpenCV формату BGR у формат RGB).
- Нормалізація – зміна інтенсивності пікселів та вирівнювання гістограми зображення для покращення контрастності в умовах слабкого освітлення [26].

Особливої уваги заслуговує модуль глибоких нейронних мереж DNN (англ. Deep Neural Networks) бібліотеки OpenCV. Хоча бібліотека OpenCV не призначена для навчання глибоких мереж, вона є потужним інструментом нейромережевої обробки зображень засобами навчених ШНМ. У бібліотеці модуль cv2.dnn підтримує завантаження моделей ШНМ, експортованих у формат ONNX (англ. Open Neural Network Exchange), що є стандартом для перенесення моделей ШНМ між фреймворками (наприклад, з PyTorch у C++ середовище). Це дозволяє запускати попередньо навчені моделі YOLO, SSD, ResNet на центральному процесорі (CPU) з високою ефективністю, часто перевершуючи за швидкістю оригінальні фреймворки завдяки оптимізації під інструкції процесора AVX, SSE (які забезпечують розпаралелювання обчислень) та можливості використання бекенду OpenVINO [94]. Наприклад, засобами бібліотеки OpenCV з використанням каскадних класифікаторів виконується детектування зображень транспортних засобів (рис. 1.6).

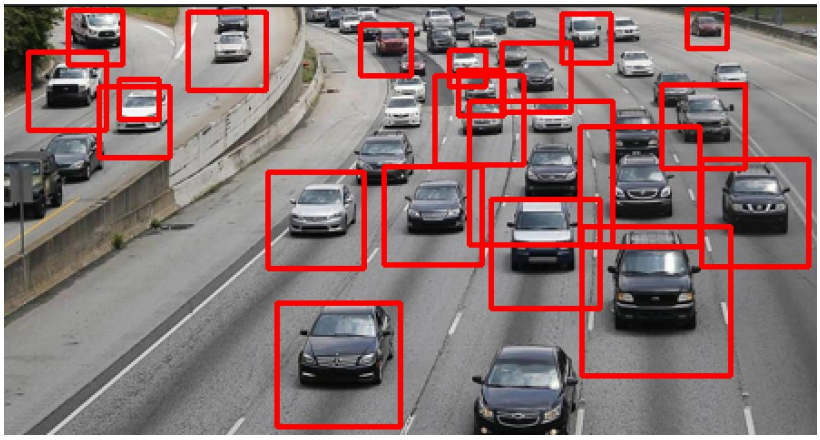


Рисунок 1.6 – Приклад детектування автомобілів засобами бібліотеки OpenCV [94]

Крім того, бібліотека OpenCV надає можливості для роботи з відео потоками. Клас VideoCapture забезпечує уніфікований інтерфейс для отримання кадрів як з відео файлів різних форматів, так і з USB-камер або IP-камер через протокол RTSP. Для задач аналізу трафіку часто застосовуються гібридні підходи: класичні алгоритми OpenCV (наприклад, віднімання фону) використовуються для детектування руху, а ШНМ застосовується тільки для класифікації знайдених рухомих об'єктів, що дозволяє суттєво економити обчислювальні ресурси [95].

1.4.2 Детектування зображень засобами хмарної платформи Google Cloud Vision

Сервіс хмарного комп’ютерного зору Google Cloud Vision API надає розробникам доступ до передових моделей машинного навчання Google через RESTful API [95]. Ключовою особливістю хмарної платформи Google Cloud Vision є використання моделей ШНМ, попередньо навчених на наборах даних великого розміру (Google Images, Street View), що у багатьох випадках дозволяє виконувати класифікацію та детектування об'єктів з високою точністю без донавчання ШНМ.

Точність детектування об'єктів у Google Cloud Vision в стандартних умовах освітлення сягає 90–95%, що досягається завдяки використанню архітектури AutoML та постійному автоматичному оновленню моделей на стороні сервера. Для задач аналізу дорожнього трафіку важливою є функція Object Localization, яка повертає координати обмежувальної рамки для кожного транспортного засобу. Функціональні можливості API платформи Google Cloud Vision також дозволяють виконувати глибокий семантичний аналіз сцени:

- Label Detection – визначення типу транспортного засобу (вантажівка, седан, автобус, ...) та умов навколишнього середовища (асфальт, дорожній знак, ...).
- Image Properties – аналіз параметрів об’єктів (наприклад, кольору).
- Text Detection – потужний модуль оптичного розпізнавання символів (англ. Optical Character Recognition – OCR), який розпізнає номерні знаки автомобілів навіть під кутом або при забрудненні, що важливо для систем контролю [95]

Технічна взаємодія з сервісом відбувається шляхом надсилання зображення (у форматі Base64 або посилання на Google Cloud Storage) у тілі JSON-запиту. У відповідь система повертає структурований JSON-файл з координатами об'єктів та оцінкою впевненості (англ. confidence score) розпізнавання об’єкту певного класу.

Однак, використання Google Cloud Vision API має суттєві обмеження, зокрема, для задач автономного водіння або удосконаленої системи допомоги водієві (англ. Advanced Driver Assistance Systems – ADAS). Головним бар'єром є залежність від стабільності інтернет-з'єднання [95]. Затримка на передачу зображення на сервер, його обробку та отримання відповіді може складати до кількох секунд, що є неприпустимим для систем реального часу.

Економічний фактор також обмежує масове використання цього інструменту. Висока вартість API при обробці великих масивів відеоданих (наприклад, потік 30 кадрів/с) робить його фінансово витратним для постійного моніторингу.

1.4.3 Детектування зображень засобами Amazon Rekognition

Сервіс Amazon Rekognition від Amazon Web Services (AWS) є одним із ключових лідерів на ринку хмарних рішень для комп'ютерного зору [96-97]. Amazon Rekognition базується на передових моделях глибокого навчання, які натреновані на мільярдах зображень, що здебільшого дозволяє виконувати детектування зображень без необхідності донавчання моделей ШНМ.

Важливою перевагою Amazon Rekognition для систем моніторингу дорожнього руху є здатність сервісу працювати не лише зі статичними зображеннями (рис. 1.7), а й з відео потоком у реальному часі. Завдяки інтеграції з Amazon Kinesis Video Streams система здатна приймати відео потік та виконувати детектування транспортних засобів з мінімальною затримкою [96].

Окремої уваги в Amazon Rekognition заслуговує функція Custom Labels, яка реалізує механізм автоматизованого машинного навчання (AutoML) для передавального навчання (англ. Transfer Learning) (або донавчання) попередньо натренованих моделей ШНМ до специфічних задач користувача.

У роботі [96] проведено порівняльний аналіз ефективності AWS Rekognition та Azure Custom Vision. Показано, що хоча AWS демонструє стабільно високі показники точності детектування, проте існують певні обмеження при роботі в складних погодних умовах або при поганому освітленні.

Рисунок 1.7 – Приклад детектування автомобіля в Amazon Rekognition [97]

Недоліком Amazon Rekognition є залежність від інтернет-з'єднання. Тарифікація здійснюється за кожну хвилину обробки відео або за тисячу зображень, тому при цілодобовому моніторингу трафіку витрати можуть бути значними.

1.4.5 Детектування зображень засобами OpenPilot

OpenPilot [98] є однією з найбільш прогресивних систем допомоги водієві (ADAS) з відкритим вихідним кодом (рис. 1.8). Система розроблена для функціонування на процесорах мобільних пристроїв (наприклад, Qualcomm Snapdragon). Значна частина обчислень виконується засобами цифрових сигнальних процесорів DSP (англ. Digital Signal Processor), які містяться в швидкодіючих чипах Qualcomm. Архітектура OpenPilot базується на використанні власних моделей глибокого навчання (Supercombo model), які виконують детектування смуг руху, інших автомобілів та пішоходів у єдиному потоці.

Особливістю системи є відмова від лідарів на користь візуальної одометрії та аналізу відео, що підтверджує гіпотезу про достатність комп'ютерного зору для задач навігації. Програмне забезпечення демонструє можливість обробки відеопотоку з частотою 20 Гц на мобільних чипах. ШНМ Supercombo model приймає на вхід послідовність кадрів (відеопотік) і на виході видає не тільки класи об'єктів, але також їх траєкторії [99].

Рисунок 1.8 – Приклад розпізнавання дорожньої розмітки засобами OpenPilot [98]

1.4.5 Детектування зображень засобами Baidu Apollo

Платформа Baidu Apollo є прикладом промислового підходу до автономного керування, яка аналізує дані з відеокамер, радарів та лідарів [100]. Для детектування об'єктів на зображеннях платформа використовує модифіковані архітектури ЗНМ (наприклад, на базі YOLO або Faster R-CNN), оптимізовані для мультикласового детектування (автомобілі, велосипедисти, пішоходи, перешкоди) (рис. 1.9).

Аналіз цієї платформи показує, що для досягнення найвищої точності в умовах міста часто використовується каскад нейромереж: одна для виявлення регіонів інтересу, інша – для детальної класифікації.

Ключовим елементом Baidu Apollo є модуль Perception, який обробляє багато сенсорні дані в режимі реального часу з використанням ЗНМ.

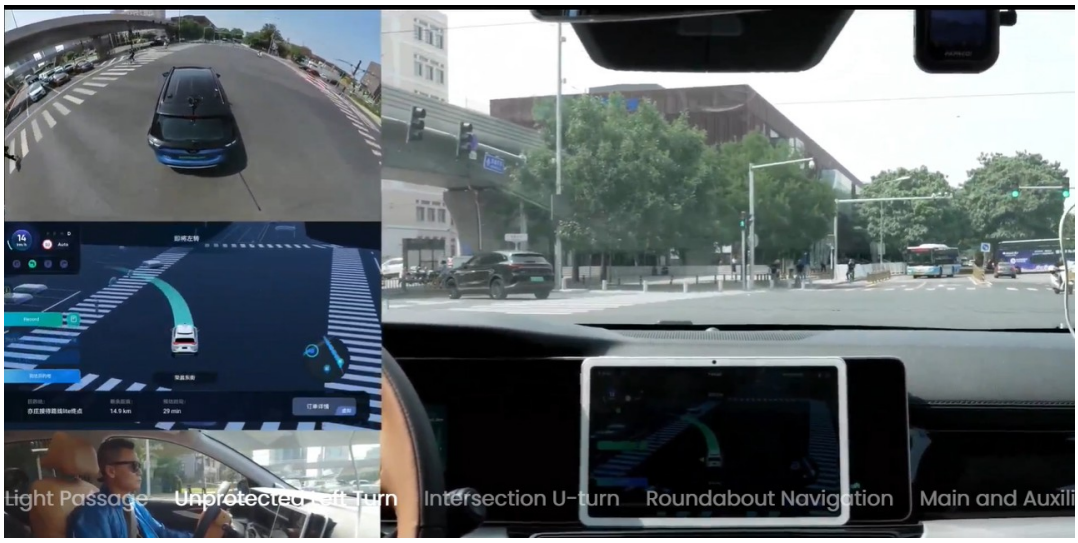


Рисунок 1.9 – Приклад аналізу відеопотоку засобами Baidu Apollo [101]

Загалом, Baidu Apollo є добре структурованою системою і готовністю до промислового застосування, однак має певні обмеження. Зокрема, стабільність модуля Perception залежить не лише від моделей ЗНМ, але й від точності калібрування сенсорів, візуальної якості даних, атмосферних умов та ін.

Висновки до розділу 1

Проведено аналіз теоретичних засад та сучасних підходів до вирішення задачі детектування автомобілів та інших об’єктів у системах комп’ютерного зору. Встановлено, що класичні методи детектування зображень (зокрема, віднімання фону, оптичного потоку, дескриптори SIFT/HOG) мають обмежену ефективність в умовах динамічних змін дорожньої сцени. Показано перспективність застосування ШНМ, зокрема ЗНМ, для вирішення задачі детектування транспортних засобів.

Розглянуто сучасні архітектури штучних нейронних мереж для детектування зображень об’єктів: двостадійні та одностадійні детектори, а також трансформери. Проведено порівняльний аналіз ШНМ з архітектурою глибоких залишкових мереж ResNet, MobileNet та з використанням модуля Inception. Розглянуто двостадійні детектори з архітектурою Faster R-CNN та з архітектурою рекурсивної піраміди ознак DetectoRS. Обґрунтовано доцільність використання для детектування зображень автомобілів одностадійних детекторів, а саме ЗНМ з архітектурою YOLO, які забезпечують потрібну точність та швидкодію детектування.

Проведено аналіз програмних засобів для детектування зображень об’єктів. Показано, що бібліотека комп’ютерного зору OpenCV може ефективно застосовуватися для попередньої обробки зображень. Проаналізовано можливості детектування зображень засобами хмарних інструментів Google Cloud Vision та Amazon Rekognition, які потребують надійного мережевого доступу.

Існуючі системи OpenPilot та Baidu Apollo, які призначені для аналізу дорожньої обстановки, володіють широкою функціональністю, проте не враховують специфіку конкретних дорожніх умов. Це підтверджує доцільність розробки власної спеціалізованої системи детектування зображень автомобілів, адаптованої до конкретних задач моніторингу дорожнього трафіку.

РОЗДІЛ 2
ДЕТЕКТУВАННЯ ЗОБРАЖЕНЬ АВТОМОБІЛІВ ЗАСОБАМИ НЕЙРОННОЇ МЕРЕЖІ YOLO З ПОПЕРЕДНЬОЮ ОБРОБКОЮ ЗОБРАЖЕНЬ

У даному розділі описано розробку інтелектуальної системи для детектування зображень автомобілів та інших учасників дорожнього руху, в якій детектування виконується засобами згорткової нейронної мережі з архітектурою YOLO. Розроблено методику попередньої обробки зображень для ЗНМ, яка полягає в оптимізації їх яскравості та контрасту. У результаті застосування такої методики підвищено точність детектування об’єктів на зображеннях.

2.1 Структура інтелектуальної системи для детектування зображень автомобілів

Розроблено структуру інтелектуальної системи для детектування зображень автомобілів та інших учасників дорожнього руху (рис. 2.1). Безпосередньо детектування об’єктів на зображеннях виконується засобами штучного інтелекту, а саме штучними нейронними мережами. Використано моделі ЗНМ з архітектурою YOLO, яка забезпечує задовільну точність і швидкодію детектування. Початкові зображення зчитуються з відеокамери або з графічних файлів, після чого надсилаються в модуль попередньої обробки, в якому виконується оптимізація яскравості та контрасту зображень (згідно з розробленою методикою). Оброблені зображення подаються на входи моделі ЗНМ YOLO, а на виходах якої отримуються результати детектування для кожного об’єкта на зображенні: координати центру (x, y) прямокутної рамки об’єкта, її ширина та висота (w, h), впевненість розпізнавання об’єкта (англ. confidence). На вихідному зображенні виконується візуалізація рамок об’єктів, які вказують на їх просторове розміщення.

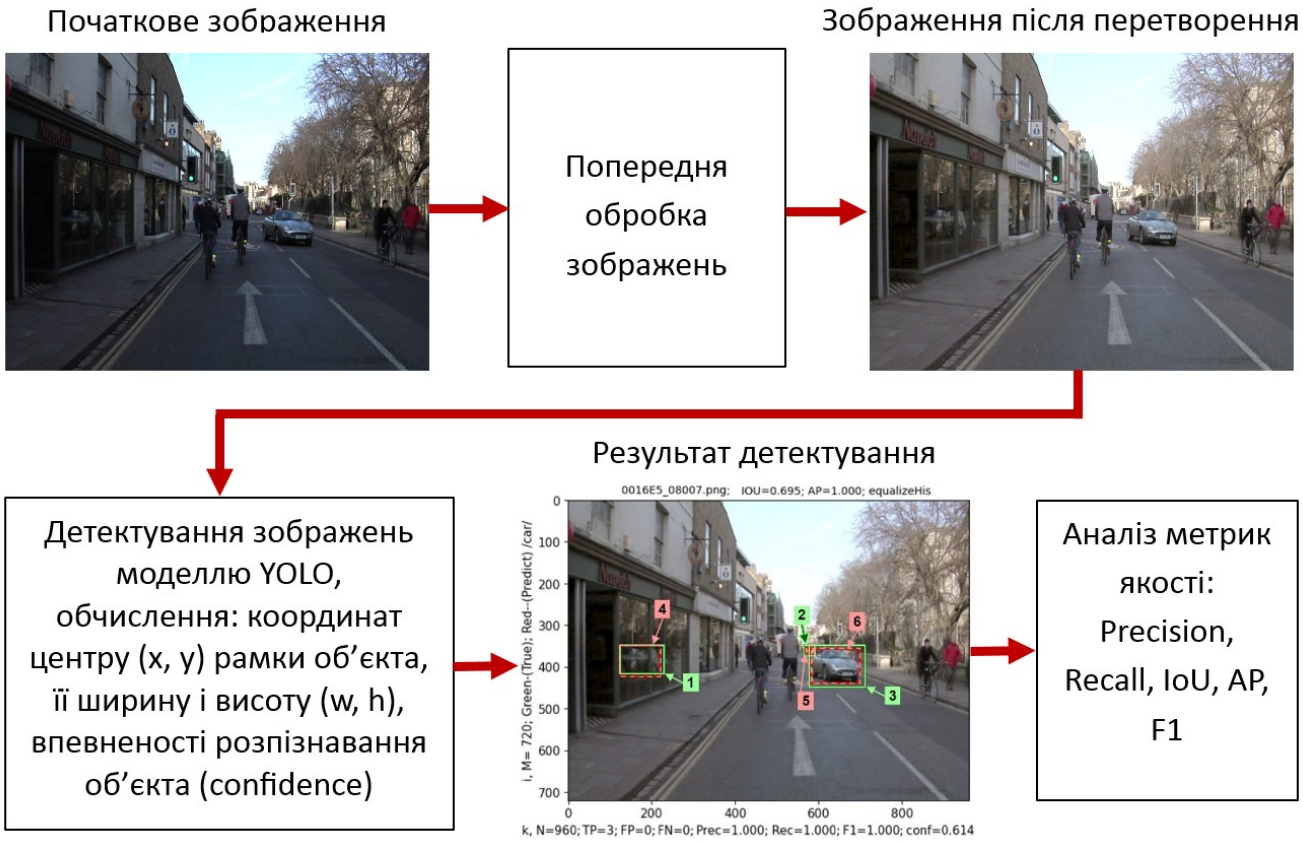


Рисунок 2.1 – Структура інтелектуальної системи для детектування зображень автомобілів із попередньою обробкою зображень

Виконано програмну реалізацію інтелектуальної системи детектування зображень на мові Python. Завдяки попередній обробці зображень забезпечується підвищення точності їх детектування навіть при застосуванні тих самих моделей ЗНМ YOLO. Точність детектування об'єктів оцінюється за такими метриками: *Precision, Recall, IoU, AP, F*1. Система передбачає застосування як попередньо навчених моделей YOLO, так й донавчання моделей ЗНМ на власних наборах даних.

Для опису принципів роботи та можливостей розробленої інтелектуальної системи розглянемо детальніше принципи функціонування ЗНМ у загальному та ЗНМ із архітектурою YOLO.

2.2 Базові принципи функціонування згорткових нейронних мереж

Основою функціонування ЗНМ є операція згортки [102, 103]. Для цифрового зображення I (у відтінках сірого) операція згортки полягає у ковзанні ядра фільтра K (розміром $m \times n$ елементів) по зображенню, множенню значень пікселів на елементи ядра та додаванню отриманих добутків (які є результатом згортки). У контексті глибокого навчання замість операції згортки часто реалізується взаємна кореляція (англ. cross-correlation), в якій ядро фільтра не повертається на 180 градусів, як у класичній згортці [76]. Кожен елемент ядра K є параметром (вагою), який змінюється у процесі навчання ЗНМ. Результатом операції згортки є карта ознак (англ. feature map), кожен елемент якої $S(i,j)$ обчислюється як сума добутків елементів ядра K та відповідних пікселів зображення I за формулою:

$$S(i,j) = (I * K)(i,j) = \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I(i+u, j+v) \cdot K(u,v) + b \qquad (2.1)$$

де: i, j – координати пікселя зображення;

u, v – координати елементів ядра фільтра;

$I(i+u, j+v)$ – значення пікселя вхідного зображення;

$K(u,v)$ – вага відповідного елемента ядра;

b – зміщення (bias).

Процес згортки дозволяє ЗНМ виявляти локальні патерни (зразки, шаблони) на зображенні. Наприклад, в перших шарах ЗНМ ядра часто навчаються детектувати прості геометричні примітиви, такі як вертикальні або горизонтальні лінії, незалежно від їх розташування на зображенні (властивість трансляційної інваріантності). У глибших шарах ЗНМ аналізуються складніші патерни (наприклад, фрагменти автомобіля).

Візуальний приклад згортки зображення показано на рис. 2.2.

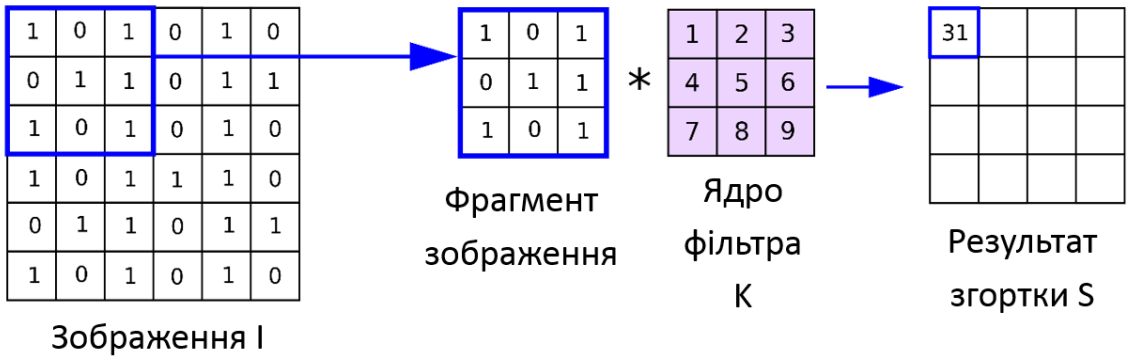


Рисунок 2.2 – Приклад операції згортки зображення I та ядра фільтра K розміром 3×3 елементів [104]

Окрім розміру ядра, робота згорткового шару регулюється двома основними гіперпараметрами, які визначають просторову роздільну здатність вихідної карти ознак: кроком згортки (англ. Stride) та доповненням (англ. Padding).

Крок згортки (англ. Stride, St) визначає крок, з яким вікно фільтра переміщується по вхідному зображенню.

- При $St = 1$ фільтр зміщується на один піксель за раз. Це стандартний режим, який максимально зберігає просторову інформацію.
- При $St > 1$ (наприклад, $St = 2$) фільтр «перестрибує» через пікселі. Це призводить до зменшення розмірності вихідної карти ознак (англ. downsampling) і використовується як альтернатива шарам підвибірки (пулінгу).

Застосування стандартної згортки без доповнення (англ. Padding, P) призводить до двох проблем:

1. Зменшення розмірності, оскільки у результаті згортки з кожним шаром карта ознак стає меншою на подвоєне значення від цілої частини половини розміру ядра (наприклад, згортка 3×3 зменшує зображення на 2 пікселі по кожній осі). У глибоких мережах це призвело б до зникнення зображення.

2. Втрата інформації на межі зображення, оскільки пікселі на краях зображення використовуються у згортці (без доповнення) рідше, ніж центральні.

Для вирішення цих задач застосовують доповнення (розширення) початкового зображення, наприклад, Zero Padding – додавання рамки з нулів навколо вхідного зображення (по периметру). Таке доповнення дозволяє контролювати розмір карт ознак ЗНМ. Часто використовують режим "Same Padding", при якому розмір входу і виходу згортки співпадає [76].

Якщо вхідне зображення I має розмір $W_{in} \times H_{in}$ пікселів, то для квадратного ядра розміром $K \times K$ елементів, доповнення на P пікселів та кроку St пікселів, розміри вихідної карти ознак ($W_{out} \times H_{out}$) розраховуються за формулами:

$$W_{out} = \left\lfloor \frac{W_{in} - K + 2P}{St} \right\rfloor + 1 \tag{2.2}$$

$$H_{out} = \left\lfloor \frac{H_{in} - K + 2P}{St} \right\rfloor + 1 \tag{2.3}$$

де $\lfloor \rfloor$ – операція округлення вниз (англ. floor).

Вищенаведені формули застосовуються для розрахунку архітектури ЗНМ, оскільки дозволяють спроектувати мережу так, щоб тензори на різних гілках мали сумісні розміри для операцій об'єднання або додавання.

Після операції згортки та активації карти ознак в ЗНМ часто мають велику розмірність, що призводить до високої обчислювальної складності. Для вирішення цієї задачі, а також для забезпечення інваріантності мережі до локальних зсувів та спотворень, використовуються шари пулінгу/підвибірки/(англ. subsampling або pooling). Тому в ЗНМ шари згортки та підвибірки часто чергуються.

Основна мета шару підвибірки – поступове зменшення просторової розмірності представлення даних при збереженні найбільш важливих структурних

елементів [105]. Це дозволяє збільшити рецептивне поле нейронів на наступних шарах без збільшення кількості параметрів.

Якщо карта ознак X має розміри $H \times W$ елементів, то операція пулінгу виконується ковзним вікном розміром $k \times k$ з кроком St. Існує два основні типи пулінгу: максимальний та середній. У більшості випадків застосовується максимальний пулінг (англ. Max Pooling), який обирає максимальне значення у вікні. Математично для області $\Omega_{i,j}$, що відповідає положенню вікна з координатами (i, j), вихідне значення $y_{i,j}$ визначається як:

$$y_{i,j} = max_{(u,v)\in\Omega_{i,j}} x_{u,v}, \tag{2.4}$$

де u, v – координати елементів у межах вікна.

Max Pooling працює як детектор найбільш виражених ознак [105]. Якщо певний патерн (наприклад, фара) був знайдений у будь-якій частині вікна 2×2, то він буде збережений. Для прихованих шарів ЗНМ Max Pooling є стандартом де-факто. Експерименти показують, що Max Pooling забезпечує високу збіжність навчання ЗНМ, оскільки передає на наступний шар тільки найсильнішу активацію [76]. Для фінальних шарів, замість розгортання карт ознак у вектор (англ. flattening), сучасні архітектури ЗНМ (ResNet, EfficientNet) часто використовують глобальний середній пулінг (Global Average Pooling – GAP), який усереднює всю карту ознак до одного числа. Завдяки цьому значно зменшується кількість параметрів перед класифікатором, що запобігає перенавчанню [105].

Після того, як згорткові та пулінгові шари виконали вилучення та ущільнення високорівневих ознак (англ. feature extraction) зображень, отримані карти ознак трансформуються у фінальний прогноз класу або координати рамки об’єкту. Таку задачу виконують повнозв’язні шари (англ. Fully Connected – FC), структура яких ідентична до шарів класичного багатошарового перцептрона.

Математично операція в повнозв'язному шарі ШНМ описується як матричне множення вхідного вектора ознак x на матрицю ваг W з додаванням вектора зміщення b:

$$z = W \cdot x + b \tag{2.5}$$

Для задач мультикласової класифікації (коли об'єкт може належати лише до одного з Q класів) вихід останнього повнозв'язного шару необхідно нормалізувати, щоб отримати розподіл ймовірностей класів. Як активаційна функція вихідного шару часто використовується функція *Softmax*. Якщо виходи нейронів останнього шару записати в масив $z = (z_1, \dots, z_Q)$, то ймовірність розпізнавання i-го класу обчислюється за формулою [85]:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{Q} e^{z_j}} \tag{2.6}$$

Функція *Softmax* має дві важливі властивості: 1) значення $\sigma(z_i)$ завжди знаходяться в діапазоні $[0, 1]$; 2) сума всіх значень дорівнює 1 ($\sum \sigma(z_i) = 1$). У контексті розпізнавання об’єктів це дозволяє системі видавати результат у форматі ймовірностей розпізнавання класів, наприклад: "Автомобіль: 0.97, Велосипед: 0.02, Фон: 0.01". Для навчання нейронної мережі з функцією *Softmax* зазвичай використовують функцію втрат перехресної ентропії (англ. Cross-Entropy Loss) [76].

На основі розглянутих шарів та функцій активації будуються складні спеціалізовані ЗНМ. Розглянемо далі ЗНМ з архітектурою YOLO, яка використана у даній роботі для розпізнавання та детектування зображень об’єктів.

2.3 Принципи функціонування згорткової нейронної мережі YOLO

Методологія детектування об'єктів пройшла значну еволюцію від методів ковзаючого вікна до одностадійних детекторів. Фундаментальний зсув у цій галузі відбувся з появою одностадійної архітектури YOLO (англ. You Only Look Once), вперше запропонованої у роботі [21]. На відміну від двостадійних методів, які спочатку генерують регіони інтересу, а потім класифікують їх, YOLO формулює задачу детектування як єдину задачу регресії, прогнозуючи координати рамок та ймовірності класів за один прохід нейронної мережі.

На даний час застосовується ряд версій ЗНМ YOLO. У даній роботі як базову обрано архітектуру 8-ї версії YOLOv8 (розробка Ultralytics), яка на сьогодні є поширеним рішенням за балансом швидкодії та точності [106-110]. Проте, напрацювання для версії YOLOv8, за потреби, можуть бути перенесеними на інші версії (наприклад, YOLOv10 або YOLOv11). Моделі YOLOv8 розрізняються за типами завдань, які вони виконують (англ. Classification, Detection, Segmentation – Розпізнавання, Детектування, Сегментація зображень) та розмірами (табл. 2.1).

Таблиця 2.1 – Типи (англ. Classification, Detection, Segmentation) та розміри (англ. Kind) моделей згорткових нейронних мереж YOLOv8

№	**Kind**	**Classification**	**Detection**	**Segmentation**
1	Nano	yolov8n-cls.pt	yolov8n.pt	yolov8n-seg.pt
2	Small	yolov8s-cls.pt	yolov8s.pt	yolov8s-seg.pt
3	Medium	yolov8m-cls.pt	yolov8m.pt	yolov8m-seg.pt
4	Large	yolov8l-cls.pt	yolov8l.pt	yolov8l-seg.pt
5	Huge	yolov8x-cls.pt	yolov8x.pt	yolov8x-seg.pt

Моделі нано (англ. Nano) розміру володіють найвищою швидкодією, але поступаються за точністю. Моделі величезного розміру (англ. Huge) відповідно володіють найвищою точністю, але поступаються за швидкодією. Таким чином, конкретна модель ЗНМ YOLO вибирається залежно від вимог до точності детектування, швидкодії та доступних апаратних ресурсів [111-114].

Структура моделі ЗНМ YOLOv8 складається з трьох основних компонентів: магістральної мережі (англ. Backbone), шиї (англ. Neck) та голови (англ. Head).

2.3.1 Еволюція сімейства нейронних мереж YOLO

Вибір архітектури нейромережі у даній роботі здійснено шляхом аналізу еволюції сімейства ЗНМ YOLO. Важливий прорив у розвитку нейромереж YOLO здійснено у 2016 році [21], коли модель YOLO досягла безпрецедентної на той час швидкості обробки (45 кадрів на секунду на GPU Titan X), що зробило можливим впровадження комп'ютерного зору в системи реального часу. На відміну від тогочасних двостадійних детекторів (таких як R-CNN), архітектура YOLO застосовує одну стадію обробки і накладає на зображення сітку. Кожна комірка цієї сітки одночасно прогнозує обмежувальні рамки об’єктів (англ. bounding boxes) та ймовірності розпізнавання класів за один прохід нейронної мережі.

Протягом наступних років архітектура YOLO зазнала значних змін, проходячи через версії v2–v5, де впроваджувалися якірні рамки (англ. anchor boxes), покращувалися магістральні мережі та функції втрат. Ключовою інновацією архітектури YOLOv7 [115] стала концепція «тренованого набору безкоштовних бонусів» (англ. Trainable bag-of-freebies) – методів оптимізації та аугментації, які покращують точність моделі ЗНМ під час навчання. Також було впроваджено розширену ефективну агрегацію мереж (E-ELAN), що дозволило моделі краще засвоювати ознаки на різних масштабах без руйнування градієнтів.

Наступним кроком еволюції стала архітектура YOLOv8 [106], розроблена компанією Ultralytics. Вона відійшла від використання якірних рамок на користь без якірного підходу (англ. anchor-free), що підвищило здатність ЗНМ узагальнювати об'єкти нестандартних форм [116]. YOLOv8 уніфікувала задачі детектування, сегментації та класифікації в межах одного фреймворку, що робить її найбільш придатною для інтеграції в системи комп'ютерного зору. Для задач детектування також є перспективним застосування новітніх архітектур YOLO (v10, v11).

2.3.2 Будова та принципи роботи моделі YOLOv8

Структура моделі YOLOv8 складається з входів (англ. Input), магістральної мережі (англ. Backbone) з модулем C2f, шиї (англ. Neck), шару прогнозу (англ. Prediction) та виходів (англ. Output) (рис. 2.3). Основною задачею магістральної мережі (англ. Backbone) є екстракція ознак із вхідного зображення [117].

У YOLOv8 використовується модифікована версія ЗНМ з крос-передачею даних через шари CSPDarknet (англ. Cross Stage Partial Darknet). Ключовою інновацією тут є заміна стандартних блоків C3 (використовуваних у YOLOv5) на модулі C2f (англ. Cross Stage Partial with 2 convolutions), який застосовує крос-передачу даних зі згорткою.

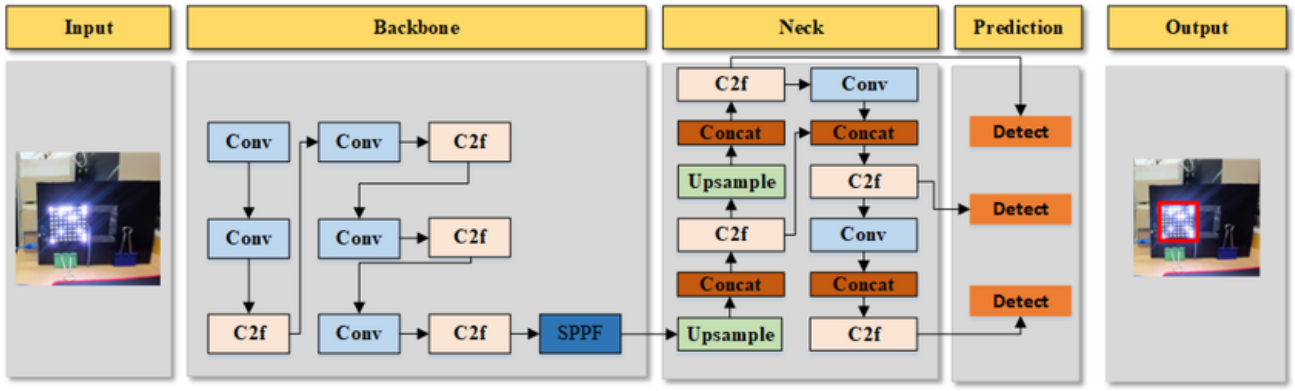


Рисунок 2.3 – Структура моделі YOLOv8, призначена для детектування об'єктів на зображені [117].

Архітектура CSPDarknet розроблена спеціально для вирішення задачі надлишковості градієнтної інформації під час навчання глибоких мереж [118]. Вона стала основою (англ. backbone) для детекторів реального часу серії YOLO (v4, v5, v8). В шиї (англ. Neck) моделі YOLOv8 реалізовано механізм агрегації ознак. Для якісного детектування об'єктів різного масштабу (наприклад, вантажівка на передньому плані та легковий автомобіль далеко на горизонті) використовується архітектура «шиї» типу PANet (англ. Path Aggregation Network) з агрегацією ознак.

Ця структура PANet реалізує двонаправлений потік інформації:

1. FPN (англ. Feature Pyramid Network) – передає семантично сильні ознаки з верхніх шарів ЗНМ до нижніх (англ. top-down).

2. PAN (англ. Path Aggregation) – передає точні локалізаційні ознаки з нижніх шарів до верхніх (англ. bottom-up).

Така комбінація дозволяє детектору "бачити" як семантичний зміст зображення (клас об'єкту), так і його точні геометричні контури, що підвищує точність детектування для об'єктів різних розмірів та ракурсів [119].

Найсуттєвішою відмінністю обраної архітектури YOLOv8 від попередників (YOLOv3-v5) є відмова від якірних рамок. У класичних версіях YOLO передбачала зміщення відносно наперед заданих шаблонів (якорів). Це створювало проблеми при детектуванні нестандартних об'єктів та вимагало ручного налаштування розмірів якорів під конкретний датасет. YOLOv8 безпосередньо передбачає центр об'єкта, що спрощує архітектуру та пришвидшує процес детектування. Крім цього, в YOLOv8 застосовується структура Decoupled Head (розділена голова). Замість того, щоб виконувати класифікацію об’єктів та регресійний аналіз координат рамок в одному шарі, ці задачі розділяються на дві паралельні гілки:

1. Гілка класифікації – визначає ймовірність належності об'єкта до певного класу (наприклад автомобіль, дерево, автомобіль) використовуючи функцію втрат Binary Cross Entropy (BCE).

2. Гілка регресії – уточнює координати рамки об’єкта, використовуючи функції втрат CIoU (англ. Complete IoU) та DFL (англ. Distribution Focal Loss).

Таке гілки дозволяють пришвидшити детектування зображень, оскільки задачі класифікації та локалізації часто конфліктують між собою в просторі ознак [38].

Функція втрат для методі YOLOv8 комплексна і має вигляд:

$$L_{total} = \lambda_{box}L_{box} + \lambda_{cls}L_{cls} + \lambda_{dfl}L_{dfl} \quad (2.7)$$

де L_{box} відповідає за точність детектування координат рамок (за метрикою IoU); L_{cls} – за точність класифікації; L_{dfl} допомагає ЗНМ фокусуватися на краях обмежувальних рамок, підвищуючи точність локалізації; значення вагових коефіцієнтів (λ_{box}, $\lambda_{cls}, \lambda_{dfl}$) встановлюються емпіричним шляхом [38].

Таким чином, вибір ЗНМ з архітектурою YOLO для детектування зображень зумовлений її здатністю навчатися на великих наборах даних, стійкістю до зміни масштабу об'єктів та високою швидкістю роботи на кінцевих пристроях.

2.4 Метрики оцінки якості детектування

Для об’єктивного порівняння ефективності різних архітектур ЗНМ необхідно використовувати стандартизований набір метрик [106, 120-122]. У задачах детектування об’єктів оцінка якості є складнішою, ніж у класифікації, оскільки вона вимагає одночасної перевірки правильності визначення класу та точності локалізації об’єкта.

Основними метриками детектування є точність (англ. Precision) та повнота (англ. Recall), які розраховуються на основі таких параметрів [123]:

- TP (англ. True Positive) – кількість правильно виявлених об'єктів на зображенні (об’єкти містяться на зображенні й правильно детектуються);

- FP (англ. False Positive) – кількість неправильно виявлених об'єктів на зображенні (об’єкти відсутні на зображенні, але ЗНМ їх виявила; помилкове детектування – "хибна тривога");
- FN (англ. False Negative) – кількість реальних об'єктів, які модель ЗНМ не змогла виявити (пропуск об’єктів, тобто об’єкти присутні на зображенні, але нейронна мережа їх не знайшла).

Метрика точності (англ. Precision) вимірює частку правильно виявлених об'єктів серед усіх об'єктів, які модель детектує на зображенні, згідно з формулою:

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

Точність відображає, наскільки точні передбачення моделі щодо виявлених об’єктів. Висока точність означає, що модель майже не робить помилок, приймаючи фонові об'єкти за цільові. Низька точність свідчить про велику кількість хибно виявлених об'єктів.

Метрика повноти (англ. Recall) визначає, яку частку всіх реальних об'єктів модель успішно виявила. Метрика розраховується за наступною формулою:

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

Повнота вимірює здатність моделі ЗНМ знаходити всі об'єкти на зображенні. Висока повнота означає, що модель виявляє майже всі шукані об'єкти, які містяться на зображенні. Низька повнота означає, що модель не виявляє значну частину об'єктів. Метрика повноти є важливою у випадках, коли пропуск важливого об'єкта є більш проблемним, ніж хибна детекція (наприклад, виявлення пішоходів у системах керування автономних автомобілів).

Для досягнення балансу між точністю (англ. Precision) та повнотою (англ. Recall) та одночасного їх врахування використовують єдину метрику з назвою F1-міра, яка обчислюється як середнє гармонійне Precision та Recall за формулою:

$$F1 = 2\frac{Precision \cdot Recall}{Precision + Recal}. \quad (2.10)$$

Метрика коефіцієнту перекриття *IoU* (англ. Intersection over Union – Перетин над Об'єднанням) описує міру збігу передбаченої моделлю та реальної прямокутної рамки для виявленого об’єкту. Метрика IoU розраховується як відношення площі перетину передбаченої рамки B_p та істинної рамки B_{gt} до площі їх об’єднання:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \quad (2.11)$$

Чим вище значення IoU, тим точніше модель визначила положення об'єкта на зображенні. Зазвичай пороговим значенням вважається $IoU \geq 0.5$. Якщо перекриття менше цього порогу, детектування вважається хибним (FP), навіть якщо клас об’єкта визначено вірно [123]. Показники Precision та Recall не є статичними – вони залежать від порогу впевненості (англ. confidence threshold) моделі ЗНМ. Зменшення порогу призводить до зростання повноти Recall (знаходиться більше об'єктів), але падіння точності Precision (з'являється більше хибних детектувань).

Метрика Mean Average Precision IoU = 0.5 (mAP50, яка позначається також mAP@0.5) є одною з основних, що використовуються для оцінки якості детектування об'єктів. Дана метрика обчислює середнє значення точності *AP* (англ. Average Precision) для всіх класів об'єктів при фіксованому порозі $IoU = 0.5$. Тобто, об'єкт вважається правильно детектованим (англ. True Positive, TP), якщо його передбачена рамка перекривається з реальною хоча б на 50%.

Значення середньої точності AP для одного класу обчислюється як площа під кривою залежності Precision (Recall) за формулою:

$$AP = \int_0^1 P(R)dR\,, \tag{2.12}$$

де $P(R)$ – значення метрики Precision (P) як функція від Recall (R), інтегрування здійснюється за всіма значеннями Recall від 0 до 1.

Значення метрики mAP50 визначаються за формулою:

$$mAP50 = \frac{1}{N}\sum_{i=1}^{N} AP_i\,, \tag{2.13}$$

де N – кількість класів об'єктів, AP_i – середня точність для i-го класу.

Однак високе значення mAP50 не завжди означає точну локалізацію об'єктів. Тому також використовують метрику mAP50-95, яка враховує більше рівнів IoU (від 0.5 до 0.95 з кроком 0.05), а тому краще відображає точність локалізації. Значення метрики обчислюється за формулою [123]:

$$mAP50-95 = \frac{mAP_{50} + mAP_{55} + \cdots + mAP_{95}}{10} \tag{2.14}$$

Такий підхід штрафує детектори за неточні рамки [124]. Високе значення метрики mAP50-95 (яка позначається також mAP@0.5:0.95) свідчить про те, що модель не лише знаходить об'єкт, а й точно окреслює його контур, що є важливим для задач автономного водіння, де помилка в кілька пікселів може впливати на оцінку дистанції.

Задача детектування автомобілів часто вирішується в контексті автономного водіння, тому важливим параметром є швидкість обробки, яка вимірюється у кадрах за секунду FPS. Для роботи в реальному часі система повинна забезпечувати обробку щонайменше 30 кадрів в секунду. Також для оцінки обчислювальної складності моделі ЗНМ використовують кількість операцій з плаваючою комою (англ. FLOPs – Floating Point Operations) та кількість параметрів мережі. Це дозволяє оцінити придатність моделі для розгортання на вбудованих пристроях [123].

2.5 Алгоритм детектування зображень автомобілів із попередньою обробкою зображень

Розроблено алгоритм детектування зображень автомобілів та інших учасників дорожнього руху, в якому передбачено попередню обробку зображень (перед надсиланням в ЗНМ) шляхом оптимізації їх яскравості та контрасту (за розробленою методикою) з метою максимізації точності детектування об’єктів (рис. 2.4). Відповідно до алгоритму спочатку зчитується ЗНМ з архітектурою YOLO (наприклад модель yolov8m середнього розміру версії YOLOv8), яка попередньо навчена на наборі даних COCO (англ. Common Objects in Context). Датасет COCO містить понад 100 тис. анотованих зображень, а об’єкти на зображеннях поділяються на 80 категорій (класів). Після цього зчитується початкове кольорове цифрове зображення *IRGB* (у моделі кольорів RGB) з відеокамери або з графічного файлу (розміром $M \times N$ пікселів). Програмно зображення обробляється як масив $IRGB = (IRGB(i, k, c))$, де $i = 0,..., M-1$; $k = 0, ..., N-1$; $c = 0,..., 2$; M – висота зображення (у пікселях); N – ширина зображення; c – номер каналу кольору (червоний, зелений, синій). Початкове зображення *IRGB* перетворюється у зображення ycrcb (у моделі кольорів YCrCb), що дозволяє окремо обробляти канал яскравості (зображення *Y*) та канали різниці кольору (зображення Cr та Cb).

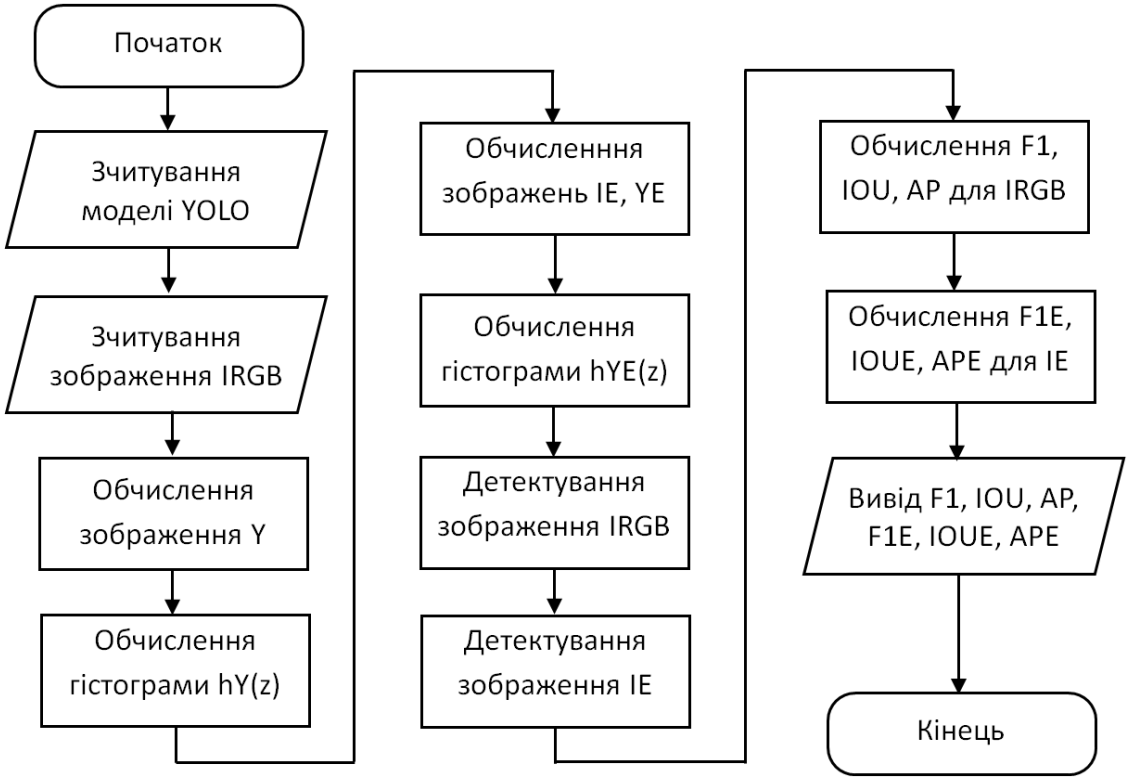


Рисунок 2.4 – Схема алгоритму детектування зображень автомобілів із попередньою обробкою зображень

З метою глибшого аналізу зображення *Y* обчислюється його гістограма *hY(z)*, де z – яскравість пікселів зображення. Сума значень всіх стовпців гістограми нормалізується до 1. Далі за розробленою методикою виконується попередня обробка зображень, а саме оптимізації їх яскравості та контрасту, у результаті чого обчислюється покращене зображення *IE* (у моделі кольорів RGB) та його канал яскравості *YE*. Для зображення *YE* обчислюється його гістограма *hYE (z)*.

Процес детектування полягає в тому, що зображення *IRGB* подається на входи ЗНМ YOLO, а на виходах ЗНМ отримуються координати центрів (*x, y*) рамок об’єктів, їх ширина та висота (*w, h)*, впевненості розпізнавання об’єктів (англ. confidence). Аналогічно відбувається детектування об’єктів на зображенні *IE* (після попередньої обробки).

Виявлені об’єкти локалізуються в прямокутних рамках, координати яких порівнюються з координатами еталонних (правильних) рамок, отриманих в ручному режимі або за допомогою сервісу Roboflow. Таке порівняння дозволяє об’єктивно і комплексно оцінювати точність детектування за метриками Precision, Recall, IoU, F1 та Average Precision (AP) для зображення *IRGB* без попередньої обробки [125], а також за відповідними метриками PrecisionE, RecallE, IoUE, F1E та APE для зображення *IE* після попередньої обробки.

2.6 Методика попередньої обробки зображень

При обробці зображень з низьким контрастом, надто високою або низькою яскравістю точність детектування значно знижується. Тому розроблено методику попередньої обробки зображень, яка полягає в оптимізації їх яскравості та контрасту з метою підвищення точності детектування зображень об’єктів [125]. Зміна яскравості та контрасту зображень виконується з урахуванням їх гістограм [126-129]. Методика базується на гіпотезі, що оптимізація яскравості та контрасту зображень здатна покращити інформативність ознак об’єктів для ЗНМ та підвищити точність детектування об'єктів у складних умовах освітлення.

Попередня обробка зображень виконувалася за допомогою фотометричних перетворень трьома способами:

1. Глобальне вирівнювання (еквалізація) гістограми [130].
2. Підвищення локального контрасту методом адаптивної еквалізації гістограми (англ. Adaptive Histogram Equalization) (метод CLAHE) [131-133].
3. Вирівнювання (еквалізація) та центрування гістограми (розроблений спосіб).

У результаті такої обробки покращилася інформативність ознак об’єктів на зображеннях як світлих, так й в темних ділянках. Розглянемо детальніше способи попередньої обробки зображень.

Спосіб №1, який реалізується методом глобального вирівнювання (еквалізації) гістограми, полягає в зміні яскравостей пікселів зображення таким чином, щоб функція розподілу значень стовпців гістограми наближалася до рівномірної. Математично таке перетворення реалізується через лінеаризацію кумулятивної функції розподілу для гістограми (англ. Cumulative Distribution Function – CDF).

Гістограма *hY(z)* зображення *Y* обчислювалася для *Qh* інтервалів, тому значення стовпців гістограми залежить не тільки від яскравості *z*, але й від номеру інтервала *b,* де *b* = 0,.., *Qh*-1. В такому випадку кумулятивна функція розподілу CDF визначається як сума стовпців гістограми, які не перевищують значення b:

$$\mathrm{CDF}(b) = \sum_{j=0}^{b} hY\,(j), \tag{2.15}$$

де $b = 0, \ldots, Qh - 1$ для зображення з Qh $(Qh = 255)$ можливими рівнями яскравості [25]. Саме CDF використовується для побудови функції перетворення, яка рівномірно розподіляє значення яскравості зображення по всьому доступному діапазону. Такий підхід дозволяє розширити динамічний діапазон зображення та підвищити глобальний контраст. Перетворення яскравостей полягає в тому, що кожному значенню *b* яскравості початкового зображення ставиться у відповідність нове значення яскравості *g(b):*

$$g(b) = round\left(\frac{\mathrm{CDF}(b) - \mathrm{CDF}_{\min}}{(M \times N) - \mathrm{CDF}_{\min}} \times (Qh - 1)\right), \tag{2.16}$$

де CDF_{min} – найменше ненульове значення CDF, $M \times N$ – кількості пікселів зображення за висотою та шириною відповідно [134].

Результатом такого перетворення є гістограма з більшим охопленням усіх яскравостей, що особливо корисно для зображень із низьким або вузьким діапазоном яскравостей. Це дозволяє розширити динамічний діапазон зображення, підвищуючи глобальний контраст, що є важливим для зображень із низькою експозицією. Однак глобальна еквалізація гістограми може призводити до надмірного підсилення шуму та втрати локальних деталей, оскільки одна глобальна функція перетворення застосовується до всього зображення. Це особливо проявляється при неоднорідному освітленні або наявності однорідних областей, де на гістограмі спостерігається пік (вузький діапазон розподілу), що спричинює перенасичення або надмірне контрастування шуму в цих зонах.

Щоб подолати такі обмеження, використано спосіб № 2 попередньої обробки зображень, а саме адаптивну еквалізацію гістограми (англ. Adaptive Histogram Equalization), яка розбиває зображення на ряд локальних прямокутних блоків (тайлів) і для кожного з них окремо обчислює гістограму та кумулятивну функцію розподілу CDF. Це дозволяє враховувати локальні статистики яскравості, що особливо ефективно при обробці сцен з неоднорідним освітленням [135].

Проте, базовий метод адаптивної еквалізації гістограми має недолік через надмірне підсилення шуму в однорідних регіонах, оскільки локальні гістограми там є вузькими та їхня *CDF* може мати великий нахил, що призводить до великого розширення певних ділянок значень яскравості [136]. Саме для вирішення цього питання розроблено метод адаптивної еквалізації гістограми з обмеженням контрасту (CLAHE – англ. Contrast Limited Adaptive Histogram Equalization). Метод CLAHE обмежує максимальну відносну кількість пікселів у кожному інтервалі (біні) гістограми, що запобігає надмірному підсиленню шуму [135-137].

Обробка зображення методом CLAHE містить такі кроки:

1. Розбиття зображення на прямокутні під тайли фіксованого розміру.

2. Для кожного тайла будується локальна гістограма яскравостей та обмежується відносна кількість пікселів у кожному біні відповідно до порога.

3. Обчислення локальної функції *CDF* на основі коректованої гістограми.

4. Перетворення яскравостей кожного пікселя за допомогою локальної *CDF*.

5. Білінійна інтерполяція між тайлами, яка дозволяє уникнути різких переходів на межах між тайлами (блоками) [137].

Такий підхід дозволяє усунути небажані артефакти глобальної еквалізації та досягти кращого балансу між підсиленням локального контрасту та контролем шуму, що підтверджують сучасні дослідження щодо застосування CLAHE [135]. Наприклад, так можливо покращити контраст зображення автомобіля у ділянці з низькою яскравістю без значного збільшення шуму [138-139].

Проте, після обробки зображень методом CLAHE у багатьох випадках максимум гістограми не завжди відповідає середині допустимого діапазону яскравостей, що призводить до зниження точності детектування. Тому для забезпечення високої точності детектування розроблено спосіб №3 попередньої обробки зображень, який полягає у вирівнюванні та центруванні їх гістограм. На гістограмі спочатку визначається центральне значення zC (за розподілом яскравості). Перетворення яскравості зображення виконується так, щоб значенню *zC* на гістограмі перетвореного зображення відповідав центр діапазону яскравості. Емпіричним шляхом встановлено, що центрування гістограми зображення забезпечує найвищу точність детектування об’єктів у випадку, якщо значення центру *zC* обчислювати як середнє значення за формулою:

$$\mathrm{zC} = \left(\frac{\mathrm{zC1} + \mathrm{zC2}}{2}\right), \tag{2.17}$$

де zC1 – центр ваги гістограми; zC2 – точка рівної площі (відносно якої площі лівої та правої частин гістограми є рівними).

Ефективність попередньої обробки зображень оцінювалася шляхом порівняльного аналізу їх гістограм до та після обробки, а також результатів детектування об’єктів на зображеннях. Така оцінка описана у наступних підрозділах. Вибір конкретного способу №1-№3 попередньої обробки зображень виконується користувачем з урахуванням особливостей оброблюваних зображень.

2.7 Програмна реалізація інтелектуальної системи для детектування зображень автомобілів із попередньою обробкою зображень

Програмна реалізація інтелектуальної системи виконана на мові Python у середовищі Google Colab у вигляді програми "YOLOContrast25" [140] (додаток Г) з урахуванням її структури (рис. 2.1), математичної моделі (2.1-2.17) та алгоритму (рис. 2.4). Точність детектування об’єктів оцінювалася за такими метриками: точність (Precision), повнота (Recall), F1-міра, IoU, AP (Average Precision). Початкові зображення IRGB зчитувалися з відеокамери або з графічних файлів, а при тестуванні системи зображення також зчитувалися з набору даних COCO.

Для програмної реалізації ЗНМ YOLOv8 використано фреймворк Ultralytics. На відміну від ранніх реалізацій YOLO, написаних на мові C (фреймворк Darknet), Ultralytics пропонує високорівневу надбудову над бібліотекою глибокого навчання PyTorch. Ключовою особливістю екосистеми Ultralytics є уніфікований API, який підтримує повний життєвий цикл розробки моделей комп’ютерного зору: від завантаження даних до розгортання на кінцевих пристроях. У даній роботі фреймворк Ultralytics застосовується для розпізнавання та детектування об'єктів [106]. Значною перевагою Ultralytics є простота експорту моделей ЗНМ для застосування на конкретному обладнанні у таких форматах [141]:

- ONNX (англ. Open Neural Network Exchange) – універсальний формат для кросплатформеного запуску.

- TensorRT – формат для максимальної швидкодії виконання на графічних процесорах NVIDIA.
- OpenVINO – оптимізація для процесорів Intel.
- CoreML та TFLite – для мобільних пристроїв iOS та Android відповідно.

Встановлення положення об’єктів (наприклад, автомобілів) на тестових зображеннях виконано в ручному режимі за допомогою інструменту Roboflow, який дозволяє точно встановити просторове положення кожного об’єкту (рис. 2.5). Координати правильних обмежувальних рамок, встановлених в Roboflow, зберігаються в спеціальному форматі і зчитуються програмою на мові Python.

Початкові зображення IRGB (у кольоровій моделі RGB з 256 рівнями для всіх каналів кольору) перетворювалися у зображення img_u8 (формат 'uint8') для подальшої обробки деякими методами бібліотеки OpenCV (зокрема, для еквалізації гістограм).

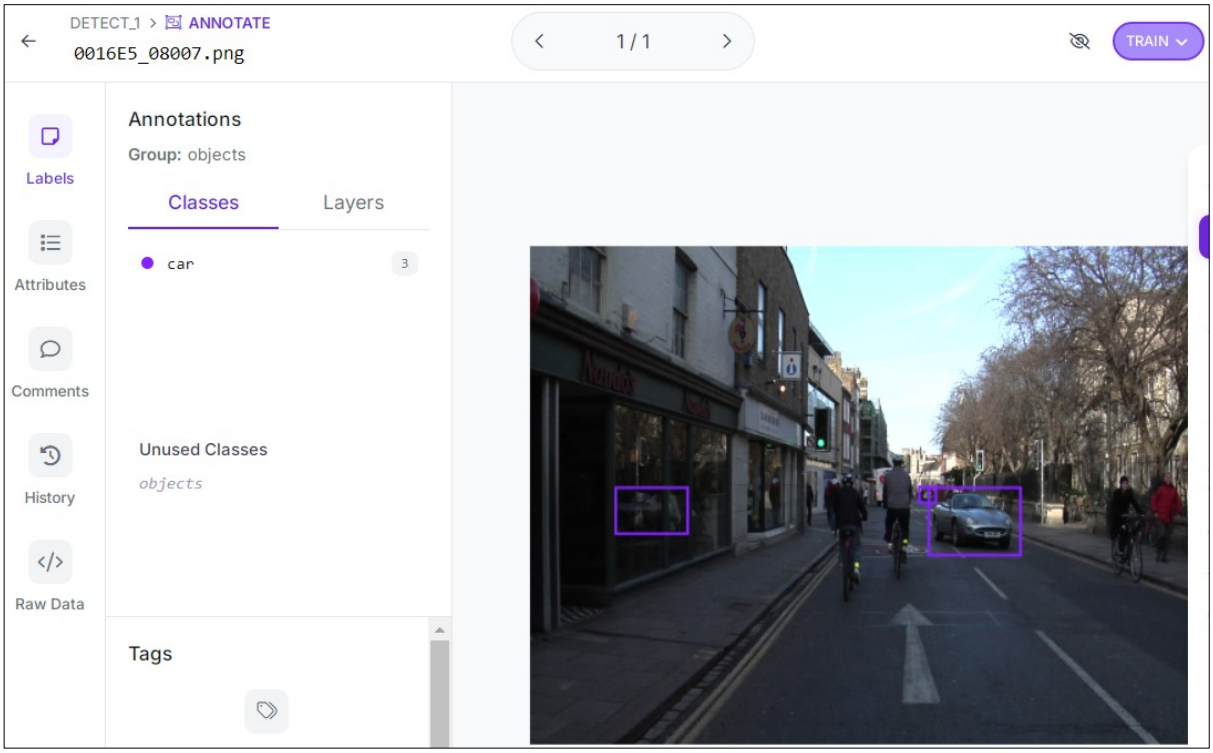


Рисунок 2.5 – Приклад виділення правильних положень автомобілів на зображенні засобами Roboflow (анотування зображення)

Початкове зображення *IRGB* перетворювалося у зображення ycrcb (у моделі кольорів YCrCb) функцією COLOR_RGB2YCrCb бібліотеки OpenCV, що дозволило окремо обробляти канал яскравості (зображення *Y*) та канали різниці кольору (зображення Cr та Cb). Гістограма *hY(z)* зображення *Y* та гістограма *hYE(z)* зображення після попередньої обробки *YE* обчислювалася для *Qh* інтервалів (наприклад, *Qh* = 32) функцією histogram бібліотеки numpy.

За розробленою методикою попередня обробка зображень виконувалася одним з трьох способів (за вибором): 1) глобальне вирівнювання (еквалізація) гістограми; 2) підвищення локального контрасту методом адаптивної еквалізації гістограми (метод CLAHE); 3) вирівнювання (еквалізація) та центрування гістограми.

Спосіб №1, а саме глобальне вирівнювання (еквалізація) гістограми, програмно реалізовано функцією «equalizeHist» бібліотеки OpenCV для кожного каналу кольору (RGB) окремо.

Спосіб №2, який полягає у підвищенні локального контрасту методом адаптивної еквалізації гістограми (англ. Adaptive Histogram Equalization), програмно реалізовано методом CLAHE бібліотеки OpenCV (імпортованої як cv2). Для цього спочатку створювався об’єкт «clahe» командою

clahe = cv2.createCLAHE(clipLimit=cLim, tileGridSize=(Tx, Yy)),

де cLim – максимальна відносна кількість пікселів у кожному інтервалі (біні) гістограми (наприклад, cLim = 1);

Tx, Yy – розміри тайлу (прямокутної ділянки) за шириною і висотою (наприклад, Tx = Yy = 2).

Підвищення локального контрасту для вхідного зображення img_u8 (перетвореного до типу 'uint8') методом CLAHE виконується командою

IE[:,:,c] = clahe.apply(img_u8[:,:,c]),

де c – номер каналу кольору в моделі RGB (c = 0, ..., 2).

Розроблений спосіб попередньої обробки зображень №3, який полягає у вирівнюванні (еквалізації) та центруванні гістограми, програмно реалізується розробленою функцією «img_hist_C», в якій спочатку обчислюється центр гістограми zC (2.17) для зображення Y (каналу яскравості початкового зображення).

Яскравість зображення Y перетворюється командами:

mask = Y <= zC # маска для зображення

YhC[mask] = Y[mask] * (127.5 / zC) # для Y <= zC

YhC[~mask] = 127.5 + 127.5*(Y[~mask] -zC) / (255.0 - zC) # для Y > zC

У результаті отримується зображення YhC, для якого значення центру гістограми зсувається до центру діапазону яскравості (центр діапазону яскравості рівний 127.5 для діапазону від 0 до 255), тобто виконується центрування гістограми.

Отримане зображення нового каналу яскравості YhC замінює канал яскравості Y в зображенні ycrcb_2 (копії ycrcb), а канали різниці кольору (Cr та Cb) залишаються без зміни. Тобто в процесі оптимізації яскравості та контрасту зображення його кольорова гамма практично не змінюється. Зображення після попередньої обробки IE (у моделі кольорів RGB) отримується з зображення ycrcb_2 функцією COLOR_YCrCb2RGB бібліотеки OpenCV, а саме командою:

IE = cv2.cvtColor(ycrcb_2, cv2.COLOR_YCrCb2RGB)

Розроблена програма дозволяє користувачу вибрати один із способів №1-3 для попередньої обробки зображень.

Детектування об’єктів на початковому зображенні виконано командою:

results = model.predict(image_BGR),

де results – результати детектування;

model – попередньо зчитана модель ЗНМ YOLO (наприклад, модель середнього розміру "yolov8m.pt");

image_BGR – початкове зображення IRGB, перетворене у кольорову модель BGR функцією COLOR_RGB2BGR бібліотеки OpenCV.

Аналогічно виконується детектування об’єктів на зображені після попередньої обробки ІЕ. З результатів детектування results зчитуються координати центрів (*x, y*) рамок всіх об’єктів, їх ширина та висота (*w, h)*, впевненості розпізнавання об’єктів (confidence). Для кожного об’єкта на зображенні з номером *nb* визначається номер його класу class_id = int(result.boxes[nb].cls[0].item()); при цьому в датасеті COCO автомобілям відповідає клас з номером 3 (якщо нумерація починається з 1). Координати рамок об’єктів шуканих класів отримуються командою cords0 = result.boxes[nb].xyxy[0].tolist(). Координати рамок отримуються у форматі (x1, y1, x2, y2), тобто вказуються координати лівого верхнього та правого нижнього кутів рамки.

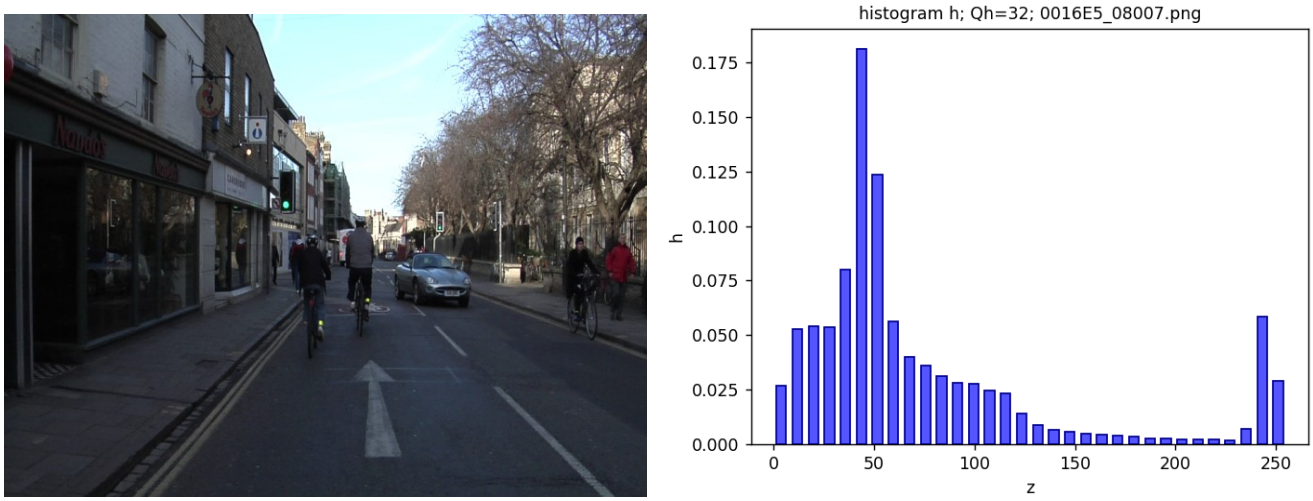
У режимі тестування зчитуються параметри правильних рамок об’єктів (з власних анотованих зображень або з датасету COCO), а шляхом порівняння передбачених ЗНМ та правильних рамок обчислюються значення метрик детектування: Precision, Recall, IoU, AP, F1. Алгоритм виконує "жадібний" пошук відповідностей правильних та передбачених об’єктів на зображенні: для кожного правильного (еталонного) об'єкта обирається прогнозований об’єкт за максимальним значенням IoU (за максимальним перекриванням їх рамок). Детектування вважається істинно позитивним (True Positive), якщо значення IoU перевищує встановлений поріг (наприклад, 0.5).

2.8 Результати детектування зображень об’єктів

Проведено перевірку розробленої інтелектуальної системи шляхом детектування автомобілів на тестових зображеннях. Метрики якості детектування обчислювалися окремо для початкових зображень та зображень після попередньої обробки.

2.8.1 Вплив попередньої обробки зображень об’єктів на точність їх детектування

Вплив попередньої обробки зображень об’єктів на точність їх детектування мережею YOLO (моделлю середнього розміру "yolov8m.pt") досліджено на прикладі обробки типового зображення дорожнього руху (рис. 2.6а) [148]. На зображенні значна частина ділянок затемнена, що видно на його гістограмі (рис. 2.6б).



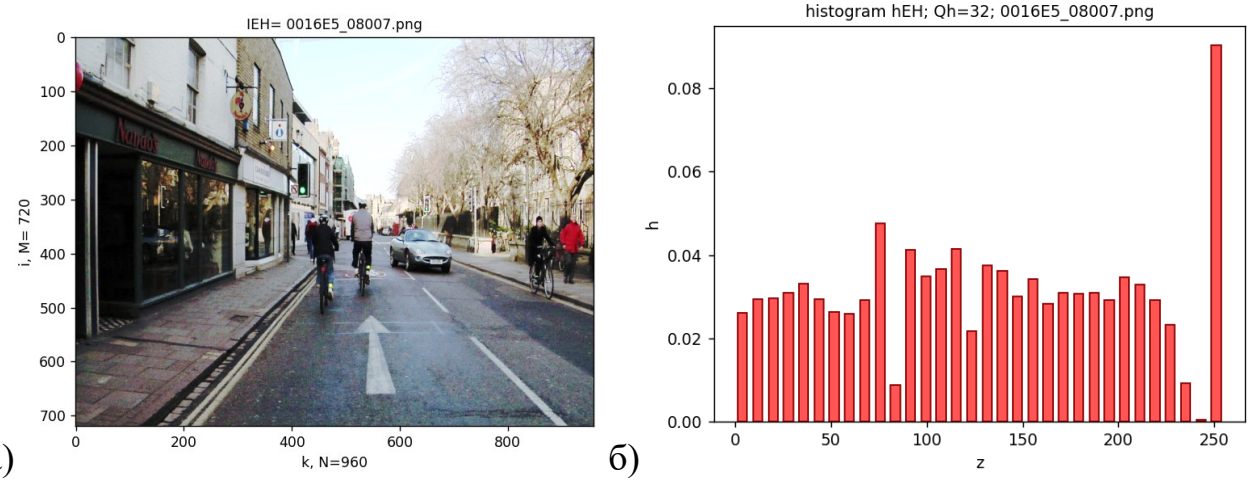
а) б)

Рисунок 2.6 – Початкове зображення дорожнього руху *IRGB* (а) та гістограма $h(z)$ його каналу яскравості Y (б), яка містить Qh інтервалів (бінів).

На досліджуваному зображенні із використанням сервісу Roboflow виділено правильні прямокутні ділянки автомобілів № 1-3 (зелені рамки) (рис. 2.7). У результаті детектування на зображенні виділяються передбачені ділянки автомобілів №4-6 (червоні рамки). Без попередньої обробки точність детектування виявилася низькою (рис. 2.7). Зокрема, на ділянці №1 відбитого у віконному склі зображення автомобіля детектується два зображення №4 та №5, а зображення автомобіля №2 не детектується [142].

Рисунок 2.7 – Детектування автомобілів на початковому зображенні

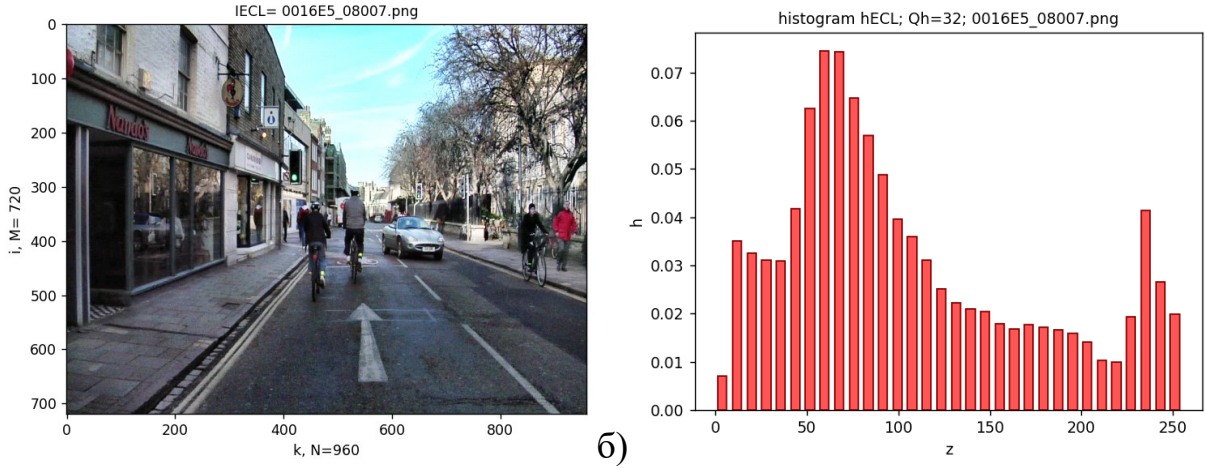
Після глобального вирівнювання гістограми способом №1 з використанням функції «equalizeHist» (рис. 2.8) отримано подібні результати детектування, як на початковому зображенні (рис. 2.6). Відсутність покращення детектування пояснюється перенасиченням зображення, оскільки частина його ділянок стала надто темною або надто світлою.



а) б)

Рисунок 2.8 – Зображення після глобального вирівнювання гістограми IEH (а) та гістограма hEH(z) його каналу яскравості (б)

Після обробки зображень способом №2, а саме підвищенні локального контрасту методом адаптивної еквалізації гістограми методом CLAHE, отримано незначне покращення візуальної якості зображення (спостерігається перенасичення), а гістограма залишилася зміщеною вліво (оскільки переважають темні ділянки) (рис. 2.9). За рахунок цього не отримано покращення точності детектування, порівно з початковим зображенням (рис. 2.6) [142].



а) б)

Рисунок 2.9 – Зображення IECL після вирівнювання гістограми методом CLAHE (clipLimit=3.0, tileGridSize=(8,8)) (а) та гісторама hECL(z) для каналу яскравості (б)

У певній мірі візуальну якість зображення можливо покращити підбором параметрів (clipLimit, tileGridSize) методу CLAHE, проте такий підбір параметрів у ручному режимі є досить трудомістким. Найкращі результати детектування отримано після еквалізації та центрування гістограми зображення запропонованим способом №3: точність детектування підвищилася (рис. 2.10), правильно детектуються всі автомобілі, навіть автомобіль №2 з малими розмірами. У розглянутому прикладі (рис. 2.6б) центр ваги гістограми zC1= 76.1, точка рівної площі (лівої і правої частини гістограми) zC2= 51.9, а значення центру гістограми zC = (zC1 + zC2)/2 = 63.0. Після еквалізації та центрування гістограми з таким параметром zC вона стає більш симетричною і рівномірною (рис. 2.10), що підвищує візуальну якість навіть темних та світлих ділянок.

а) б)

Рисунок 2.10 – Детектування автомобілів на зображенні IE після еквалізації та центрування гістограми способом №3 (а) та гістограма hE(z) його каналу яскравості (б)

За рахунок цього підвищено точність детектування об’єктів практично за всіма метриками (параметрами) (рис. 2.11).

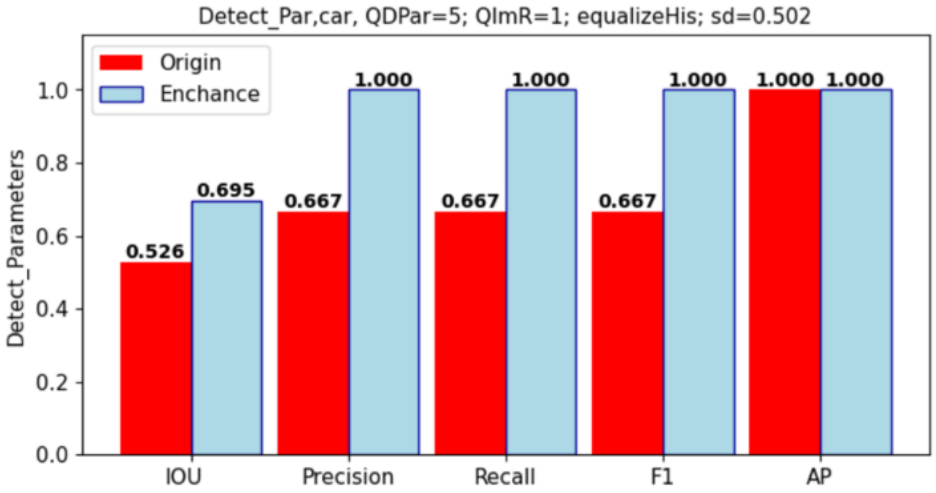


Рисунок 2.11 – Параметри точності детектування автомобілів; Origin – на початковому зображенні (рис. 2.7), Enchance – на зображенні після еквалізації та центрування гістограми (рис. 2.10)

Проведено також еквалізацію та центрування гістограми зі значенням центру гістограми zC1 та zC2, проте в обох випадках на зображеннях отримано нижчу точність детектування, порівняно з використанням значення zC як центру гістограми. Після попередньої обробки зображень запропонованим способом №3 підвищилася середня точність детекції об'єктів за IoU на 0,169, а сумарне покращення за показниками *IoU*, *F*1 та *AP* досягло $sd = 0.502$ [125]. Локальне підсилення контрасту також сприяло більш надійному виділенню малих або затемнених об'єктів, що особливо важливо у дорожніх сценах із різним рівнем освітлення. Детектування об'єктів на інших зображеннях з переважно темними або світлими ділянками (асиметричною гістограмою) показало подібне як у досліджуваному випадку (рис. 2.11) покращення точності детектування за метриками Precision, Recall, IoU, AP, F1. При дослідженні серії з 10-ти тестових зображень за метрикою *IoU* отримано покращення на 16 %.

2.8.2 Регресійний та кореляційний аналіз результатів детектування зображень

Проведено інтелектуальний аналіз даних, а саме регресійний та кореляційний аналіз [143-147] результатів детектування зображень для знаходження взаємозв'язків між метриками якості детектування IoU, Confidence, Precision, Recall, F1, AP та *Ar* – середньою відносною площею детектованих об'єктів на зображенні (площа об'єктів визначалася відносно площі всього зображення). Проведено аналіз результатів детектування 100 зображень з валідаційного набору даних COCO без попередньої обробки та з попередньою обробкою зображень. В обох випадках отримано подібні регресійні залежності. Під час аналізу значення однієї метрики використовувалися як параметр *X*, а значення іншої метрики – як параметр *Y*. Всі досліджувані дані розділено на навчальну ($N = 80$ значень) та контрольну ($NV = 20$ значень) вибірки.

Навчальна вибірка призначена для кореляційного та регресійного аналізу даних, а контрольна вибірка – для перевірки результатів такого аналізу. Значення параметрів навчальної вибірки записувалися в масиви *X* та *Y*, а значення параметрів контрольної вибірки записувалися в масиви *XV* та *YV*.

Шляхом кореляційного аналізу обчислено лінійний зв'язок між параметрами [148-149]. Такий зв'язок кількісно описано коефіцієнтом кореляції Пірсона *Corr*, який обчислюється для значень навчальної вибірки за формулою:

$$Corr = \frac{\sum_{i=0}^{N-1}(x_i - X_s)(y_i - Y_s)}{\sqrt{\sum_{i=0}^{N-1}(x_i - X_s)^2} \cdot \sqrt{\sum_{i=0}^{N-1}(y_i - Y_s)^2}}, \quad (2.18)$$

де x_i, y_i – значення параметрів *X* та *Y* відповідно; X_s, Y_s – середні значення *X* та *Y*.

Значення коефіцієнта *Corr* завжди знаходиться в діапазоні від -1 до +1. Значення, близькі до +1, вказують на сильну позитивну кореляцію (зі зростанням одного параметру другий також зростає), а значення, близькі до -1, вказують на сильну негативну кореляцію (зі зростанням одного параметру інший зменшується). Значення, близьке до 0, свідчить про відсутність лінійного зв'язку.

Рівняння регресії, яке описує зв'язок $y(x)$ між значеннями метрик якості, апроксимовано поліномом степені *p*, який описується формулою:

$$y_A(x) = a_0 + \sum_{n=1}^{p} a_n \cdot x^n . \quad (2.19)$$

Апроксимовані значення y_A записано в масиви *YAN*. Коефіцієнти полінома $a_0 ... a_p$ обчислено методом найменших квадратів. Для оцінки точності апроксимації використано корінь середньої квадратичної різниці RMSE (англ. Root mean square error) для навчальної вибірки та *RmseV* для контрольної вибірки, які обчислювалися за формулами:

$$Rmse = \cdot \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} \left(y(i) - y_A(i) \right)^2} \,. \tag{2.20}$$

$$RmseV = \cdot \sqrt{\frac{1}{N_V} \sum_{i=0}^{NV-1} \left(y_V(i) - y_{AV}(i) \right)^2} \,. \tag{2.21}$$

Найвище значення коефіцієнта кореляції Corr отримано для залежностей *conf* (*IoU*) та *Prec* (*IoU*) (рис. 2.12). Високе значення коефіцієнта кореляції $Corr = 0.9344$ між впевненістю розпізнавання об'єктів conf (confidence) та коефіцієнтом перекриття *IoU* підтверджує правильність результатів детектування, оскільки у випадку точного визначення меж об'єкта (високе значення *IoU*) його клас визначається з високою надійністю. Аналогічно високе значення коефіцієнта кореляції $Corr = 0.9826$ між точністю Precision та IoU підтверджує правильність результатів детектування, оскільки точна локалізація об'єкта на зображені сприяє високій точності його виявлення. Отримані залежності *conf* (*IoU*) та *Prec* (*IoU*) можуть застосовуватися для цілеспрямованого дослідження зображень, метрики якості детектування яких значно відхиляються від рівнянь регресії.

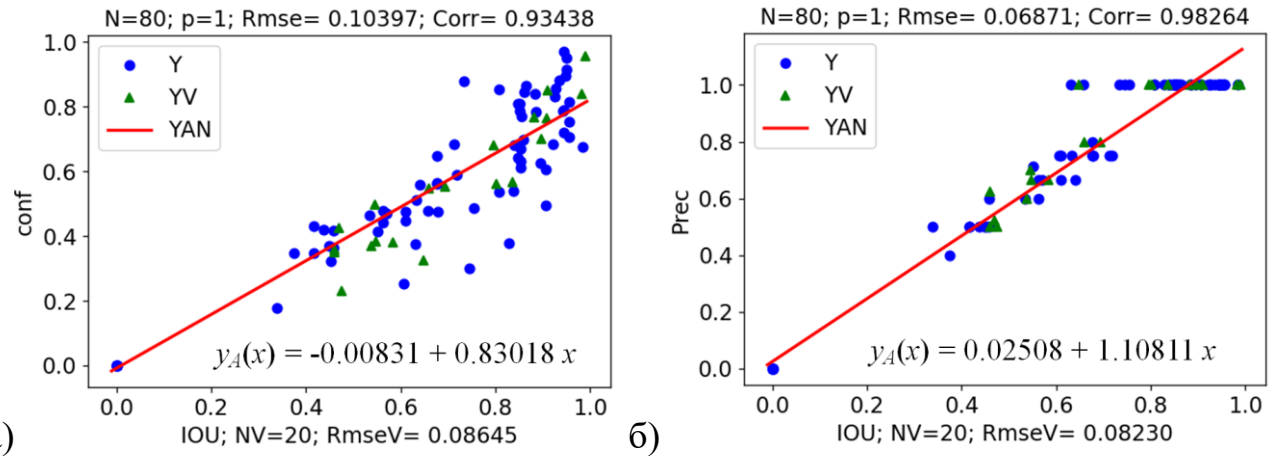
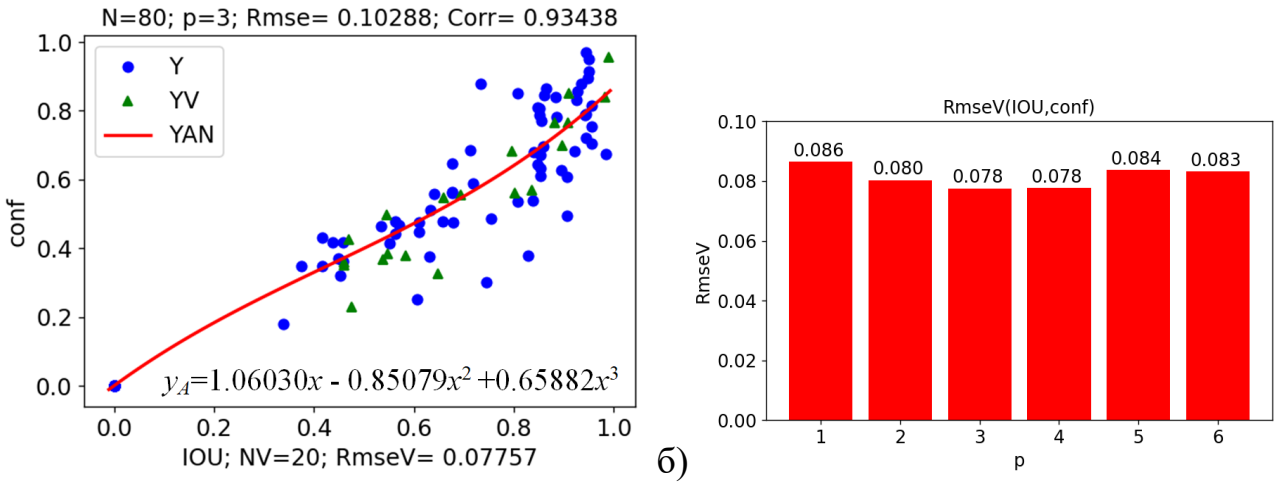


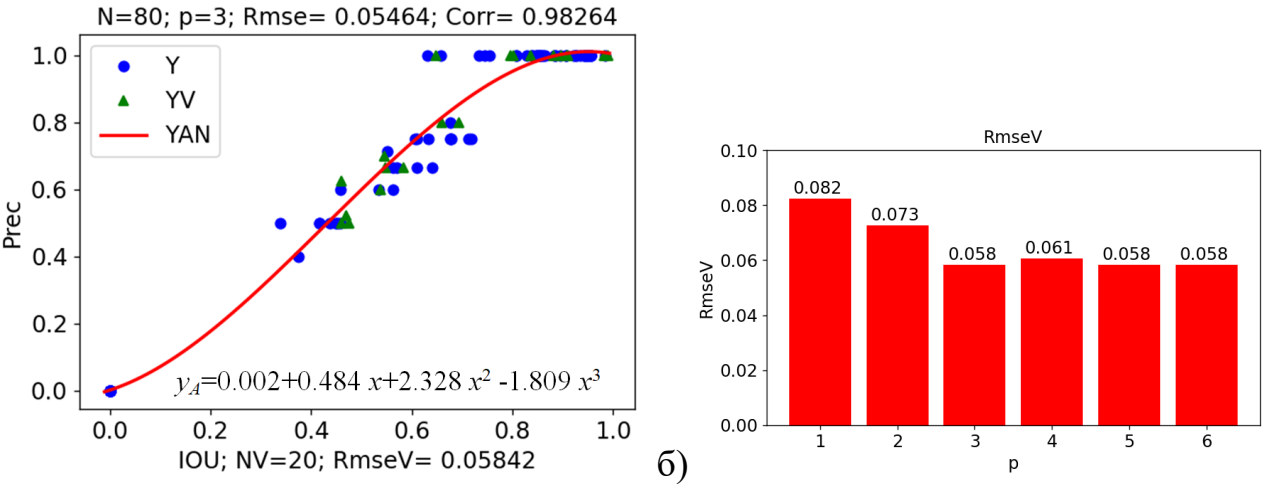
Рисунок 2.12 – Лінійна регресія між значеннями метрик якості та їх коефіцієнти кореляції *Corr*, досліджено взаємозв'язки: *conf* (*IoU*); б) *Prec* (*IoU*)

Автоматичне визначення оптимальної степені полінома *p* виконано шляхом перебору значень у заданому діапазоні від 1 до 6. Оптимальна степінь полінома *pA* обчислена як значення *p*, для якого отримано мінімальну похибку апроксимації *RmseV* для контрольної вибірки (рис. 2.13, рис. 2.14).



а) б)

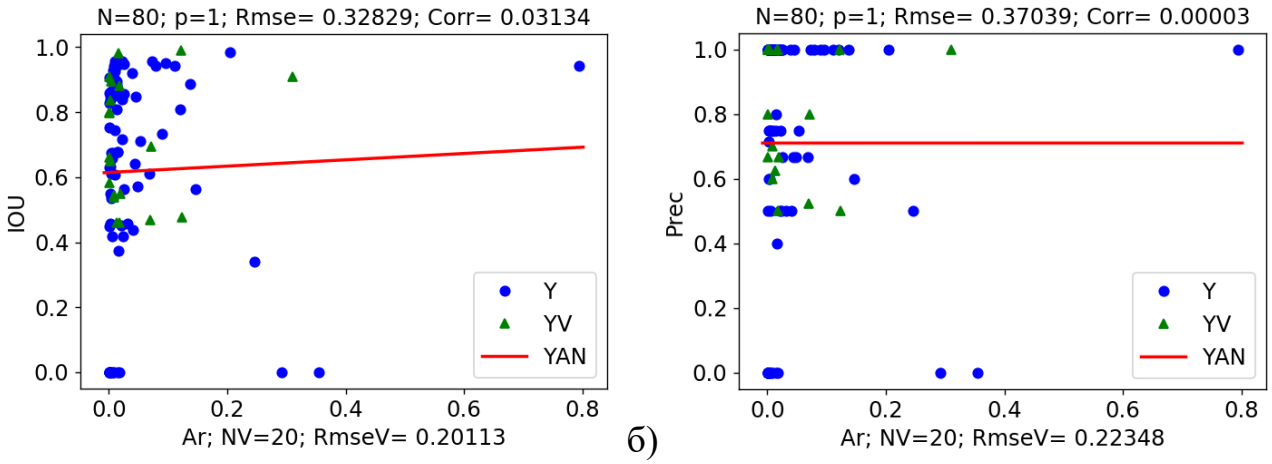
Рисунок 2.13 – Рівняння регресії між значеннями *conf* (*IoU*) у вигляді полінома степені 3 (а) та залежність похибок апроксимації для контрольної вибірки (*RmseV*) залежно від степені полінома *p* (б)



а) б)

Рисунок 2.14 – Рівняння регресії між значеннями *Prec* (*IoU*) у вигляді полінома степені 3 (а) та залежність похибок апроксимації для контрольної вибірки (*RmseV*) залежно від степені полінома *p* (б)

Отримані рівняння регресії для нелінійних залежностей *conf* (*IoU*) та *Prec* (*IoU*) можуть застосовуватися для прогнозу метрик якості детектування (*conf*, *Prec*) залежно від коефіцієнту перекриття рамок *IoU*. Аналіз зв'язків метрик якості (*IoU*, *Prec*) та середньої відносної площі *Ar* детектованих об'єктів на зображеннях показав практично відсутність кореляції (*Corr* ≈ 0) між їх значеннями (рис. 2.15).



а) б)

Рисунок 2.15 – Лінійна регресія між значеннями метрик якості та їх коефіцієнти кореляції *Corr*, досліджено взаємозв'язки: *IoU* (*Ar*); б) *Prec* (*Ar*)

Отриманий результат означає, що розроблена інтелектуальна система забезпечує високу точність детектування об'єктів не тільки великого, але й середнього та малого розміру (для низьких значень *Ar*).

Висновки до розділу 2

Розроблено структуру, математичну модель та програмне забезпечення інтелектуальної системи для детектування зображень автомобілів та інших учасників дорожнього руху, в якій детектування виконується засобами згорткової нейронної мережі з архітектурою YOLO. З метою підвищення точності детектування попередня обробка вхідних зображень для ЗНМ виконується за розробленою методикою, яка полягає в оптимізації їх яскравості та контрасту.

Розглянуто базові принципи функціонування згорткових нейронних мереж і принципи функціонування згорткової нейронної мережі YOLO, яка призначена для одностадійного детектування зображень об’єктів. Як базову обрано архітектуру 8-ї версії YOLOv8, описано її структурні особливості, зокрема, застосування безякірного підходу до локалізації рамок об’єктів. Розглянуто основні метрики оцінки якості детектування: Precision, Recall, IoU, F1 та Average Precision (*AP*).

Розроблено алгоритм детектування зображень автомобілів та методику попередньої обробки вхідних зображень для ЗНМ з використанням 3 способів: 1) глобального вирівнювання (еквалізація) гістограми; 2) підвищення локального контрасту методом адаптивної еквалізації гістограми (метод CLAHE); 3) запропонованим способом вирівнювання та центрування гістограми.

Виконано програмну реалізацію інтелектуальної системи для детектування зображень автомобілів ЗНМ YOLOv8 із попередньою обробкою зображень. Програму "YOLOContrast25" реалізовано на мові Python у середовищі Google Colab з використанням фреймворку Ultralytics. Анотування тестових зображень виконано в ручному режимі за допомогою інструменту Roboflow.

Досліджено вплив попередньої обробки зображень об’єктів на точність їх детектування за метриками Precision, Recall, IoU, AP та F1. Найкращі результати детектування отримано після попередньої обробки зображень запропонованим способом еквалізації та центрування гістограми зображення, при дослідженні серії тестових зображень за метрикою IoU отримано покращення на 16 %.

Проведено регресійний та кореляційний аналіз результатів детектування зображень для знаходження взаємозв’язків між метриками якості з використанням валідаційного набору даних COCO. Отримані рівняння регресії можуть застосовуватися для цілеспрямованого дослідження зображень, метрики якості яких значно відхиляються від загального тренду.

РОЗДІЛ 3
ДОНАВЧАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ YOLO ДЛЯ ДЕТЕКТУВАННЯ ЗОБРАЖЕНЬ АВТОМОБІЛІВ ІЗ РУЧНИМ ФОРМУВАННЯМ ДАТАСЕТУ

У даному розділі розроблено методику донавчання згорткових нейронних мереж сімейства YOLO різних версій для задачі детектування зображень автомобілів із ручним формування набору даних. У розділі описано етапи формування набору даних, включно з попередньою обробкою зображень, ручною анотацією, розподілом вибірок та застосуванням методів аугментації. Розроблену методику донавчання ЗНМ реалізовано програмно та інтегровано в інтелектуальну систему для детектування зображень автомобілів. Наведено особливості програмної реалізації процесу донавчання нейронних мереж YOLO та проведено аналіз результатів детектування для різних версій моделей на основі метрик якості. Отримані результати дозволяють оцінити точність детектування для різних версій YOLO та дослідити вплив донавчання на точність виявлення об’єктів.

3.1 Структура інтелектуальної системи для детектування зображень автомобілів із донавчанням нейронних мереж

Структуру інтелектуальної системи для детектування зображень автомобілів із попередньою обробкою зображень (рис. 2.1) доповнено модулем донавчання ЗНМ YOLO, у результаті чого отримано структуру інтелектуальної системи для детектування зображень автомобілів із донавчанням ЗНМ (рис. 3.1). Модуль донавчання додано до існуючих блоків попередньої обробки зображень (модуль №1), детектування зображень об’єктів засобами початкової (pre-trained) моделі YOLO та обчислення метрик якості для початкової моделі YOLO.

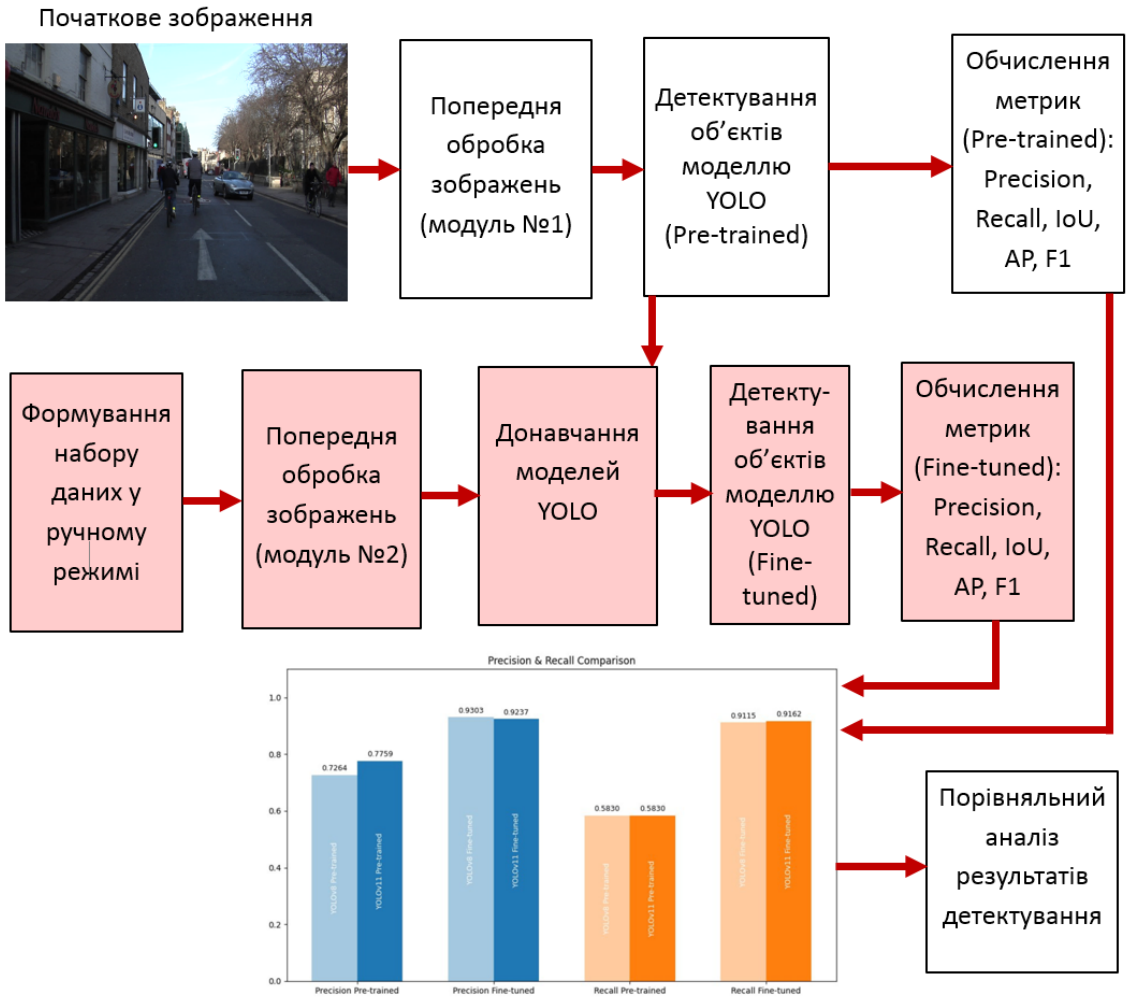


Рисунок 3.1 – Структура інтелектуальної системи для детектування зображень автомобілів із донавчанням ЗНМ

Модуль донавчання реалізує етапи розробленої методики донавчання ЗНМ:

1. Формування набору даних (датасету) у ручному режимі за допомогою платформи Roboflow.
2. Попередньої обробки зображень (модуль №2).
3. Донавчання моделей YOLO (fine-tuned) за допомогою сформованого датасету.
4. Детектування тестових зображень донавченою (fine-tuned) моделлю YOLO.
5. Обчислення метрик якості для результатів детектування донавченою моделлю YOLO.

На основі обчислених метрик якості (Precision, Recall, F1, IoU, confidence, AP) проводиться порівняльний аналіз результатів детектування зображень різними моделями YOLO:

1. Моделями до (pre-trained) та після (fine-tuned) донавчання.
2. Різними версіями та розмірами моделей (наприклад, YOLOv8m та YOLOv11m).

Такий порівняльний аналіз дає змогу перевірити ефективність донавчання моделей ЗНМ та вибрати моделі YOLO, характеристики яких (зокрема, точність та швидкодія) найбільш повно відповідають умовам конкретного завдання.

3.2 Методика формування набору даних у ручному режимі та донавчання нейронних мереж

3.2.1 Вимоги до набору даних

Детектування транспортних засобів є окремим класом задач комп'ютерного зору, який характеризується особливими вимогами до наборів даних, які застосовуються для навчання ШНМ. Ефективність системи детектування залежить не лише від архітектури використаної нейронної мережі, а й від процесу її навчання та релевантності датасету. До основних факторів, що ускладнюють детектування зображень автомобілів, належать:

1. Масштабна варіативність, оскільки автомобілі на зображенні можуть займати як декілька десятків пікселів за шириною та висотою (знаходячись на значній відстані від камери), так і все зображення (знаходячись безпосередньо перед камерою). Проте, поширені детектори зображень часто втрачають ефективність при виявленні дрібних об'єктів [150].

2. Оклюзія та усічення виникає через те, що у щільному дорожньому трафіку автомобілі часто частково перекривають один одного або інші об'єкти

інфраструктури (пішоходи, стовпи). Це призводить до втрати характерних ознак (коліс, фар), за якими мережа ідентифікує об’єкт. Усічення відбувається, коли об’єкт виходить за межі кадру камери, що вимагає від алгоритму здатності екстраполювати форму автомобіля за його видимою частиною.

3. Вплив погодних умов, освітлення та часу доби, оскільки дощ, сніг, туман та відблиски від мокрого асфальту вносять суттєві дефекти у зображення та спотворюють кольоропередачу. Адаптація детекторів зображень до складних погодних умов часто вимагає використання специфічних методів аугментації даних або мультиспектральних сенсорів (наприклад, поєднання RGB-камер та тепловізорів) [151].

Тому для навчання та тестування алгоритмів у сфері детектування зображень автомобілів використовуються спеціалізовані датасети, які значно відрізняються від загальних наборів даних (таких, як ImageNet). До спеціалізованих датасетів, орієнтованих на детектування автомобілів, належать:

- KITTI Vision Benchmark Suite – один із перших і найбільш авторитетних наборів даних для задач автономного водіння, що містить стереозображення та розмітку для 3D-детектування [152].
- MS COCO (англ. Microsoft Common Objects in Context) – даний загальний датасет є стандартом для оцінки метрик якості детектування автомобілів завдяки великій кількості екземплярів автомобілів різного масштабу [153].
- Cityscapes та BDD100K – набори даних великих розмірів, які фокусуються на семантичному розумінні дорожніх сцен, отриманих для різноманітних погодних умов і часу доби [154].

Проте, розглянуті датасети орієнтовані тільки на певні дорожні умови. Тому в даному дослідженні створено варіативний набір даних (датасет) з високим рівнем генералізації, що дозволяє донавчати й адаптувати сучасні моделі YOLO до задач детекції автомобілів у складних дорожніх сценах. Датасет сформовано шляхом

вибору та анотування 1542 зображень, серед яких 1002 містили автомобілі, а 540 – не містили. Сформований датасет зображень поділено на навчальний набір (1018 зображень), контрольний набір (420 зображень) та тестовий набір (104 зображення). Значна увага приділена різноманіттю сценаріїв для зображень дорожнього руху, що охоплюють зміни розмірів зображень, освітлення, погодних умов, дистанцій, напрямків руху автомобілів та інших факторів [155].

Формування набору даних виконувалося шляхом обробки та селекції «сирих» даних (зображень), отриманих з мережі Інтернет або з власних фотокамер, що дозволяє охопити різні ракурси та умови освітлення. З метою забезпечення повноти представленості різноманітних ситуацій встановлено такі вимоги до характеристик зображень датасету [155]:

- Відсоток заповнення кадру транспортним засобом: від повно кадрових зображень автомобіля до кадрів мінімальними розмірами автомобіля.
- Якість зображення: містить як високоякісні чіткі зображення, так і кадри з поганою роздільною здатністю або високим рівнем шуму.
- Інтенсивність трафіку: сцени з низькою, середньою та високою щільністю дорожнього руху.
- Час доби: денні, вранішні, вечірні та нічні умови освітлення для моделювання різних сценаріїв експлуатації.
- Погодні умови: сонячна погода, дощ, сніг, а також інші атмосферні ефекти, які можуть впливати на видимість.
- Відстань до відеокамери та деталізація: зображення з добре видимими деталями автомобіля та такі, на яких деталі розпізнаються погано.
- Основний напрямок руху транспортного засобу: рух у напрямку камери, від камери або під кутом до оптичної осі камери.
- Кут нахилу камери: зйомка зверху вниз, паралельно до земної поверхні або під кутом.

Доповнюючи вищенаведений перелік, також враховано додаткові параметри для покращення варіативності датасету:

- Оклюзії: часткове перекриття транспортного засобу іншими об'єктами.
- Фон сцени: урбаністичне середовище, заміські дороги, паркування, тунелі тощо.
- Артефакти зображення: відблиски від сонця чи фар, тіні, засвітки.
- Стан транспортного засобу: чистий або забруднений автомобіль, наявність пошкоджень кузова.
- Швидкість руху транспортного засобу: нерухомий, повільно рухається або рухається на високій швидкості, що може вливати на чіткість зображення автомобіля.

Такий підхід до формування датасету дозволяє забезпечити максимальну варіативність навчальних даних, підвищує здатність моделі ЗНМ до генералізації та забезпечує високу точність детектування в умовах реального середовища.

3.2.2 Формування власного набору зображень

З урахуванням вищенаведених вимог виконано формування власного датасету зображень автомобілів на основі відео з датасету «Highway Traffic Videos Dataset» [155]. Для автоматизації створення датасету розроблено програму на мові Python, яка зчитувала та зберігала кадри автомобільного трафіку з відеопотоку (рис. 3.2). Такі зображення стали основною власного датасету. Далі до них було додано зображення з інших відкритих датасетів, широко доступних відео та фото, а також власні зображень дорожнього руху, отримані за допомогою відеокамер. У реальних умовах зображення, отримані з відеопотоку або камер спостереження, можуть мати різну роздільну здатність, пропорції, рівні освітлення, шум та інші артефакти, що знижують точність моделей машинного навчання без відповідної обробки даних.

Рисунок 3.2 – Приклади зображень з різною інтенсивностю автомобільного трафіку (низька, середня, висока) на відео датасету «Highway Traffic Videos Dataset»

Дослідження показують, що методи попередньої обробки зображень, які полягають, зокрема, в масштабуванні, нормалізації яскравості та підвищенні контрасту зображень, сприяють меншій похибці навчання моделей та вищій точності детектування об’єктів засобами ЗНМ [156].

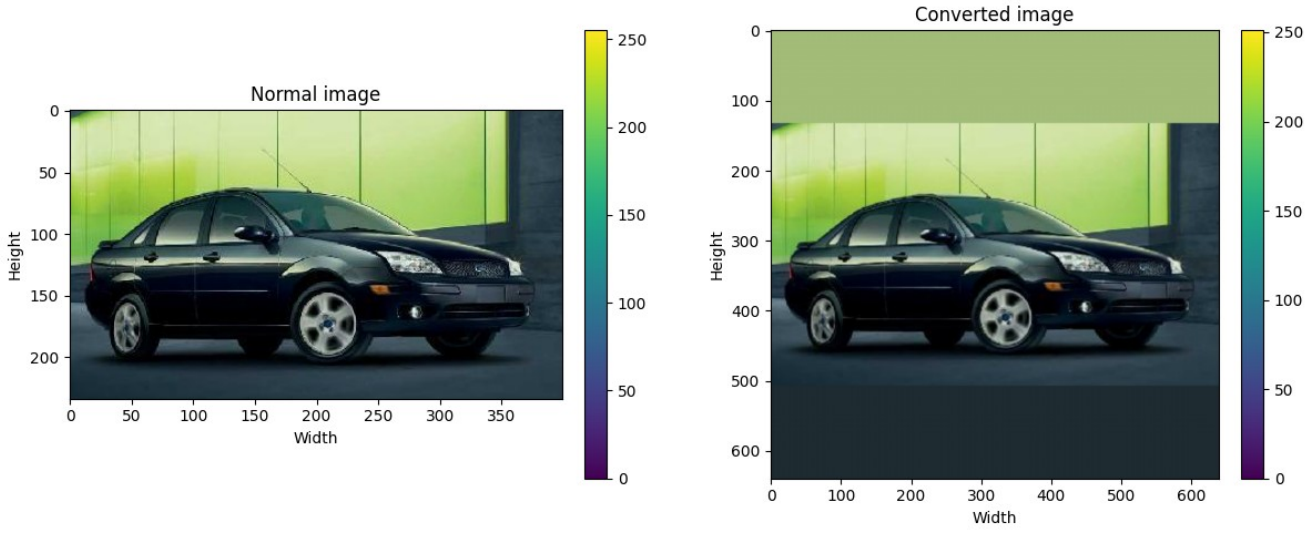
Тому перед обробкою зображень датасету в ЗНМ виконано їх попередню обробку. Така обробка зображень є важливим етапом підготовки даних для згорткових нейронних мереж, оскільки якість вхідних зображень безпосередньо впливає на здатність моделей виявляти ознаки, узагальнювати інформацію та уникати перенавчання [157]. Розроблено програмне забезпечення на мові Python з використанням бібліотеки OpenCV для попередньої обробки зображень, які завантажуються з каталогів (використовуються окремі каталоги для зображень з автомобілями та без них).

Перший етап попередньої обробки зображень полягав у їх масштабуванні методом reshaped_image, який перетворював зображення до стандартного розміру M_S × N_S пікселів (наприклад, до розміру 640 × 640 пікселів). Якщо оригінальне зображення не квадратне, то до нього додаються додаткові смуги пікселів або видаляються непотрібні (для збереження пропорцій). З цією метою розроблено алгоритм, який перетворює прямокутні зображення в квадратні. Колір доданих смуг пікселів обчислювався як середнє значення найближчої половини зображення, що

дозволило зменшити спотворення на межах смуг масштабованих зображеннях (рис. 3.3) [158]. Таке масштабування та доповнення зображень до квадратних дозволяє моделі ЗНМ обробляти вхідні зображення з різними пропорціями без втрати структурної інформації, що забезпечує однорідність представлення даних на вході нейромережі [156].

Крім масштабування, виконано нормалізацію яскравостей зображень (кожного колірного каналу) до діапазону від 0 до 1. Нормалізація піксельних значень також може полягати в приведенні їх до нульового середнього та одиничної дисперсії, що базується на статистичних характеристиках тренувального набору.

Такий процес нормалізації яскравостей прискорює збіжність алгоритмів навчання ЗНМ, оскільки узгоджує масштаби вхідних даних і зменшує внутрішню коваріаційну зміну (англ. internal covariate shift) під час проходження даних через шари мережі. Це сприяє як стабільнішому навчанню (за критерієм похибки навчання), так і кращій узагальнювальній здатності ЗНМ на нових даних [159].



а) б)

Рисунок 3.3 – Приклад перетворення початкового прямокутного зображення (а) в масштабоване квадратне зображення (б) з додаванням нижньої та верхньої смуг

Окрім масштабування та нормалізації, попередня обробка зображень може включати ідентифікацію та усунення шумів, підвищення контрасту, а також видалення непотрібних фрагментів фону [159]. Вищеописана попередня обробка зображень датасету забезпечує кращу збіжність та узагальнюваність моделі ЗНМ, зменшує вплив випадкових артефактів та підвищує якість детектування транспортних засобів на нових зображеннях [158].

3.2.3 Анотування зображень та розподіл датасету за допомогою платформи Roboflow

Проведено анотування зображень датасету, що дозволило їх застосовувати для донавчання ЗНМ YOLO. Анотування виконано в ручному режимі з використанням платформи Roboflow [155], яка автоматизує процеси анотації, аугментації та версіонування датасетів для їх застосування в системах комп'ютерного зору. До переваг платформи належить автоматична конвертація датасетів у формати YOLO (TXT), Pascal VOC (XML), COCO (JSON), TFRecord, що проблему сумісності між різними архітектурами ШНМ [160].

Формат даних YOLO застосовується для навчання ЗНМ з архітектурою YOLO. У такому форматі кожному зображенню відповідає текстовий файл із нормалізованими координатами обмежувальних рамок об’єктів (x, y, w, h), що є ефективним для навчання одностадійних детекторів у режимі реального часу [113].

Pascal VOC (англ. Visual Object Classes) є форматом даних, який використовує структуровані XML-файли для детального опису метаданих об’єктів, включаючи складні атрибути (difficult, truncated, occluded), і є фундаментальним стандартом для валідації алгоритмів у межах щорічних візуальних викликів [161].

Формат COCO (англ. Common Objects in Context) базується на єдиній ієрархічній структурі JSON, яка інтегрує детекцію об'єктів, сегментацію екземплярів

та розпізнавання ключових точок, що робить його придатним для оцінки складної контекстуальної взаємодії об'єктів [161].

TFRecord – бінарний формат серіалізації даних на основі протоколу Protocol Buffers, розроблений для екосистеми TensorFlow, що забезпечує лінійне зчитування та максимальну пропускну здатність каналу "диск-процесор" при роботі з надвеликими масивами даних [162].

Також перевагою Roboflow є можливість швидко знаходити помилкові детектування та додавати їх у навчальну вибірку для перенавчання моделі. Проте, у платформи є залежність від зовнішньої інфраструктури.

Ефективне навчання ЗНМ потребує високої точності розмітки обмежувальних рамок. З цією метою в Roboflow використано інструменти Smart Polygon (Розумні полігони), що дозволяють щільно охоплювати контури автомобілів, виключаючи зайвий фон (асфальт, будинки), який може вносити похибки у навчання. Особливо корисною для великих масивів дорожнього трафіку є функція Label Assist (Допомога з мітками). Вона дозволяє використати попередньо навчену модель ЗНМ для автоматичного детектування автомобілів на нових зображеннях, що скорочує час ручної розмітки. Перед подачею даних у нейронну мережу Roboflow дозволяє стандартизувати вхідні зображення, що є обов'язковим для архітектур типу YOLO або SSD. Стандартизація зображень в Roboflow полягає в наступному:

1. Auto-Orient (Автоматична орієнтація) – автоматичне виправлення орієнтації зображень, що особливо важливо при обробці зображень, отриманих з камер відеоспостереження або дронів (БПЛА). У процесі такої корекції враховуються значення метаданих для цифрових зображень EXIF (англ. Exchangeable Image File Format), в яких зберігається інформація про параметри камери, дату й час зйомки, геолокацію, орієнтацію зображення та ін. [163].

2. Resize (Зміна розміру) – приведення всіх кадрів до єдиного розміру без спотворення пропорцій автомобілів, використовуючи методи заповнення (англ.

padding) або розтягування (англ. stretching). В даній роботі зображення було перетворені до розміру 640 × 640 пікселів.

Roboflow генерує аналітичні звіти щодо розподілу класів (англ. Class Balance). Це дозволяє виявити дисбаланс, наприклад, коли зображень легкових авто значно більше, ніж вантажівок чи автобусів. Інструмент також показує "теплову карту" (англ. Heatmap) розташування об'єктів у кадрі, що допомагає уникнути перенавчання моделі на детектування автомобілів лише в центрі зображення.

У Roboflow реалізовано механізм версіонування (англ. Dataset Versioning), що дозволяє створювати знімки даних (v1, v2, v3) з різними параметрами аугментації для порівняльного аналізу ефективності моделей. Є можливість швидко знаходити помилкові детектування та додавати їх у навчальну вибірку (англ. Active Learning).

Анотацію зображень у ручному режимі за допомогою платформи Roboflow виконано у такій послідовності. Першим кроком роботи з платформою є наповнення проєкту даними. У меню проєкту користувач обирає опцію “Upload Data” (Завантажити дані). Після завантаження файлів система автоматично перевіряє їх на наявність дублікатів та пошкоджених файлів. У випадку успішної перевірки виконується імпорт зображень у поточний проєкт.

Другим кроком у системі Roboflow є етап ручної анотації (меню “Annotate”). В процесі анотації на кожному зображенні в ручному режимі виділено рамки автомобілів за допомогою інструмента “Bounding Box Tool” та присвоєно їм клас «cars» (рис. 3.4). Аналогічно виконується розмітка для об’єктів інших класів. На цьому кроці Roboflow автоматично генерує метадані для кожного анотованого об'єкта: ідентифікатор класу, координати центру рамки $(x,\ y)$ відносно розмірів зображення, а також її ширину та висоту. Для коректного навчання ЗНМ є важливим правильний розподіл даних між навчальною, контрольною та тестовою вибірками, що забезпечує об’єктивність оцінки та запобігає перенавчанню. З цією метою в Roboflow застосовано інструмент для автоматичного балансування датасету.

Рисунок 3.4 – Приклад виділення рамок автомобілів на зображенні в Roboflow за допомогою інструмента Bounding Box Tool

Засобами Roboflow всі зображення розподіллено між трьома вибірками:

- Train (навчальна) – основна частина даних для тренування моделі ЗНМ (1018 зображень, 70 %).
- Valid (валідаційна або контрольна) – використовується для перевірки моделі під час навчання (420 зображень, 20 %).
- Test (тестова) – використовується для фінальної оцінки якості навчання ЗНМ (104 зображень, 10 %).

3.2.4 Аугментація даних за допомогою платформи Roboflow

Засобами платформи Roboflow виконано автоматичну аугментацію (англ. Augmentation) даних для створеного датасету, оскільки така аугментація (повороти зображень, зміни яскравості, додавання шуму) дозволяє збільшити точність детектування Mean Average Precision (mAP) на 15-20% при роботі з невеликими

наборами даних [141]. Аугментація зображень автомобілів на платформі Roboflow виконується такими методами:

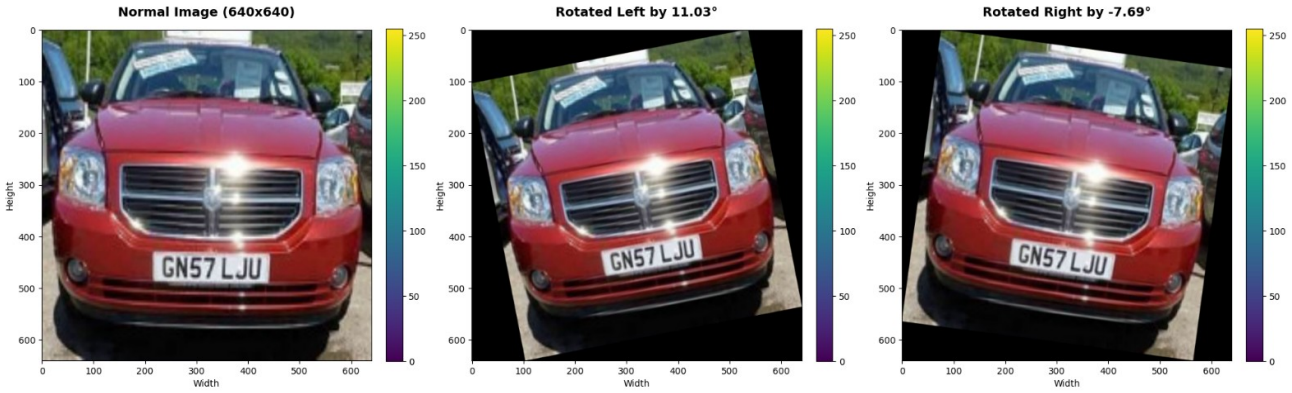
• Geometric Augmentations (Геометричні доповнення) – використовується Horizontal Flip (горизонтальне віддзеркалення), оскільки автомобіль може мати різні напрями руху. Водночас Vertical Flip (вертикальне віддзеркалення) слід вимкнути, оскільки автомобілі рідко знаходяться в перевернутому стані.

• Environmental Simulation (Моделювання навколишнього середовища) – зміна яскравості та контрасту імітує різні умови освітлення (день, ніч, тіні від будівель). Додавання Gaussian noise (Гаусівський шум) або Blur (розмиття) допомагає моделі краще розпізнавати рухомі об'єкти, що емулює ефект швидкості.

• Mosaic Augmentation (Мозаїчна аугментація) – одна з ключових функцій для навчання детекторів YOLO. Вона поєднує 4 різні зображення в одне, змушуючи модель вчитися знаходити об'єкти в несподіваних контекстах та різних масштабах. Це важливо для детектування віддалених автомобілів на загальних планах.

Одним із підходів до збільшення обсягу даних (аугментації) є застосування випадкового обертання зображень на невеликі кути. У створеному датасеті кожне вхідне зображення з набору даних оберталося спочатку вправо, а потім вліво на випадковий кут (у діапазоні -10 до 10 градусів). Таким чином, кожне оригінальне зображення доповнено двома повернутими зображеннями (рис. 3.5). Це дозволяє моделі ЗНМ навчитися розпізнавати об'єкти на зображеннях з різних кутів огляду.

Створений датасет аугментовано шляхом застосування геометричних та фотометричних перетворень, зокрема Flip (горизонтальне віддзеркалення), Rotation (поворот у діапазоні від −10° до +10°), Shear (зсув до ±5° по горизонталі та вертикалі), Brightness (зміна яскравості в межах від −15 % до +15 %), Blur (розмиття з радіусом ядра фільтра 1 піксель) та Noise (додавання імпульсного шуму до 1,01 % всіх пікселів) (рис. 3.6). За рахунок аугментації ЗНМ стає більш адаптованою до різних умов зйомки (кутів огляду, освітлення тощо).



а) б) в)

Рисунок 3.5 – Приклад аугментації даних шляхом повороту зображення: а) початкове зображення; б) повернуте вліво; в) повернуте вправо.

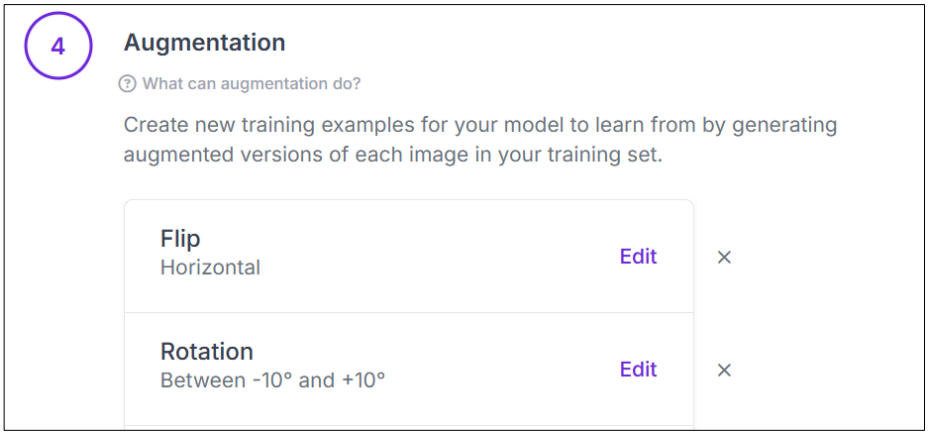


Рисунок 3.6 – Фрагмент форми для налаштування параметрів аугментації (Flip, Rotation) у Roboflow

Це також допомагає уникнути перенавчання, оскільки модель ЗНМ навчається на більш різноманітних даних [155]. Збільшення різноманітності даних зменшує чутливість ЗНМ до шуму та інших артефактів у даних [158]. Після аугментації розмір датасету збільшився у 3 рази (до 3578 зображень). Однак аугментацію слід обмежувати, оскільки збільшення розміру датасету призводить до збільшення часу навчання моделі ЗНМ.

Платформа Roboflow зберігає новий датасет в архіві (формат “YOLOv8”), який складається з трьох основних директорій: «train» (навчальна вибірка), «test» (тестова вибірка) та «valid» (валідаційна вибірка), а також конфігураційного файлу «data.yaml». Файл «data.yaml» містить метадані проєкту: шляхи до відповідних директорій з даними, кількість класів об’єктів та їхні назви. У датасеті кожна з директорій вибірок («train», «test», «valid») містить два підкаталоги: «images» та «labels». У підкаталозі «images» розміщено зображення, тоді як підкаталог «labels» містить відповідні файли анотацій у текстовому форматі (.txt). Зв’язок між зображенням та його анотацією забезпечується ідентичністю назв файлів (наприклад, зображенню image_01.jpg відповідає файл розмітки image_01.txt).

Структура текстових файлів підпапки «labels» має строго регламентований вигляд: кожен рядок (запис) файлу описує один детектований об’єкт і складається з п’яти значень, розділених пробілом. Формат запису наступний:

$$< class\_id > \ < x_{center} > \ < y_{center} > \ < width > \ < height >$$

де: $class\_id$ – ціле число, що позначає ідентифікатор (номер) класу об’єкта;

$x\_\{center\}, y\_\{center\}$ – нормалізовані координати центру обмежувальної рамки відносно ширини та висоти зображення (значення у діапазоні від 0 до 1);

$width, height$ – нормалізовані ширина та висота обмежувальної рамки відносно розмірів усього зображення.

Таким чином, створений датасет у форматі “YOLOv8” може застосовуватися для навчання моделі ЗНМ YOLO версії 8.

3.2.5 Алгоритм донавчання нейронних мереж YOLO та аналізу результатів донавчання

Основні етапи розробленої методики донавчання нейронних мереж YOLO та аналізу результатів донавчання описано у вигляді алгоритму (рис. 3.7), згідно з яким

спочатку зчитується початкова модель YOLO №1 (наприклад, YOLOv8m або YOLOv11m) та створений набір даних, розділений на навчальну, контрольну та тестові вибірки. Далі встановлюються гіперпараметри навчання (кількість епох, розмір пакету, умова зупинки навчання та ін.) і виконується донавчання моделі ЗНМ, у результаті чого отримується донавчена модель №2. Виконується трансферне навчання нейромережі, тобто попередньо навчені ваги моделей були уточнюються на розробленому наборі зображень. Це дозволило значно прискорити конвергенцію моделей та досягти високої точності детектування зображень транспортних засобів [35]. Зображення тестового набору детектуються моделями №1 та №2, після чого обчислюються метрики якості детектування, які використовуються для аналізу результатів донавчання ЗНМ.

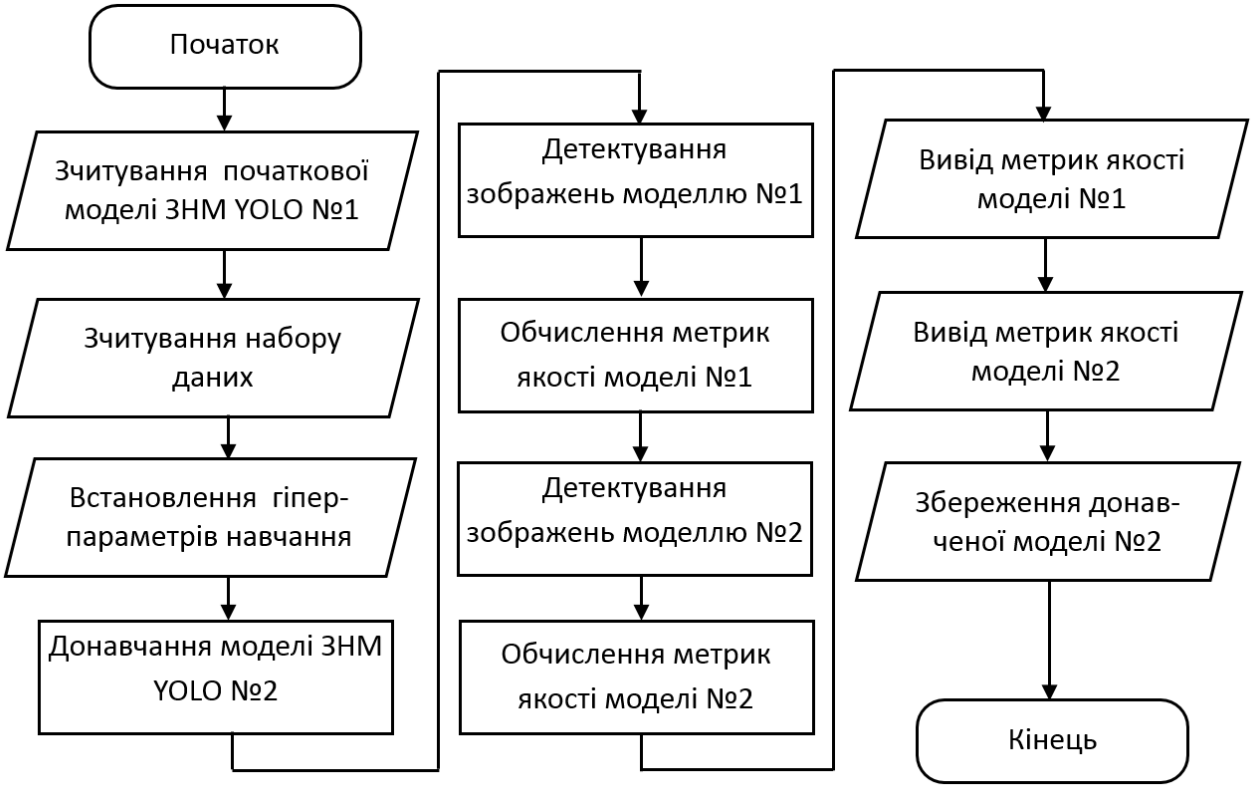


Рисунок 3.7 – Схема алгоритму донавчання моделі ЗНМ YOLO та обчислення метрик якості детектування зображень

3.3 Програмна реалізація донавчання нейронних мереж

Програму «FineTunedYOLOCarDetection» (додаток Д) для донавчання нейромережі та детектування об’єктів розроблено на основі алгоритму (рис. 3.7) та реалізовано на мові Python у хмарному сервісі Google Colab [164] за допомогою записника Jupyter Notebook [165]. Однією переваг Google Colab є доступ до обчислювальних ресурсів, зокрема GPU та TPU, без необхідності налаштування локального середовища, що дозволяє зменшити час навчання ЗНМ. Поєднання коду з поясненням в Jupyter Notebook сприяє кращій реплікації результатів, оскільки послідовність обробки даних, архітектура моделі, гіперпараметри та результати зберігаються в одному документі [165, 166]. В програмі використано ряд бібліотек, зокрема opencv, ultralytics, numpy, matplotlib та ін.

Для навчання ЗНМ YOLO використано алгоритм стохастичного градієнтного спуску SGD (Stochastic Gradient Descent) та модифікацію алгоритму адаптивної оцінки моменту (Adaptive Moment Estimation), а саме алгоритм AdamW. Алгоритм SGD забезпечує високу узагальнюючу здатність моделі та стабільність збіжності на великих наборах даних. Застосування SGD дозволяє досягати вищої точності (mAP) на фінальних етапах навчання, що є важливим для архітектур сімейства YOLO при детекції об'єктів у складних умовах. Алгоритм AdamW добре справляється з пропусками в даних, проте може призводити до перенавчання. Тому для остаточного навчання ЗНМ використано алгоритм SGD. Донавчену модель ЗНМ №2 збережено у файл. За допомогою моделей ЗНМ №1 та №2 виконано детектування тестових зображень, а на основі результатів детектування обчислено метрики якості.

Реалізацію моделей YOLO виконано з використанням бібліотеки Ultralytics [16], яка забезпечує широкий вибір параметрів та гіперпараметрів навчання нейромережі. У програмі використано моделі YOLO середнього розміру (medium), а саме моделі YOLOv8m та YOLOv11m. Процес навчання для обох моделей було

уніфіковано: встановлено ліміт у 75 епох, базову швидкість навчання (англ. learning rate) на рівні 0.001 та розмір вхідних зображень 640 × 640 пікселів.

Програмна реалізація навчання побудована на принципах автоматизації конвеєра даних та уніфікації середовища тестування. На початковому етапі виконується конфігурація робочого простору: за допомогою бібліотеки os динамічно формуються шляхи до компонентів датасету (навчальної, валідаційної та тестової вибірок), що структуровані у форматі Roboflow. Для забезпечення відтворюваності результатів здійснюється попередній аналіз складу даних через зчитування файлу data.yaml, що дозволяє програмі ідентифікувати кількість категорій об'єктів та обсяг вхідних зображень перед початком ітераційного процесу.

Ключовим аспектом реалізації є використання розширеного словника гіперпараметрів hyperparams, який інтегрується в метод навчання обох моделей. Для оптимізації збіжності ваг застосовано режим змішаної точності (amp: True) та механізм ранньої зупинки (patience: 20), що припиняє процес навчання при відсутності покращення метрик якості, запобігаючи перенавчанню [106].

Безпосередній цикл донавчання (англ. fine-tuning) реалізовано через виклик методу .train() для кожної моделі окремо з фіксацією часових витрат за допомогою бібліотеки time. Програма обробляє дані з розміром пакета (батча) 16, що забезпечує баланс між швидкістю обчислень та стабільністю градієнта. Після завершення навчання виконується автоматичне збереження фінальних ваг моделей у форматі .pt до директорії models_save_path, а також експорт повної статистики навчання (графіків втрат та матриць помилок) у каталог результатів для подальшого компаративного аналізу ефективності архітектур YOLOv8m та YOLO11m.

На початковому етапі донавчання моделей YOLO виконувалося на CPU (тривалість 1 епохи навчання – 1 година 9 хвилин) [155]. Після цього навчання виконувалося на графічному процесорі NVIDIA Tesla T4, що дозволило скоротити тривалість 1 епохи навчання до 1 хвилини 37 секунд. Використання TPU

забезпечило майже 40-кратне прискорення навчання, що пояснюється розпаралелюванням обчислень [155]. Експериментальні дослідження проводилися з використанням хмарного середовища Google Colab на базі графічного прискорювача NVIDIA Tesla T4 (16 ГБ VRAM).

Розроблена програма включає розгалужену систему моніторингу, яка дозволяє в режимі реального часу та постфактум аналізувати ефективність моделей. Для цього розроблено комплекс інструментів візуалізації на базі бібліотек matplotlib та seaborn, які зчитують результати з лог-файлів results.csv. Система автоматично генерує порівняльні графіки метрик точності (Precision, Recall) та середньої точності (mAP50, mAP50-95), що дозволяє наочно оцінити динаміку збіжності процесу навчання моделей YOLOv8 та YOLOv11 на кожній епосі. Окремий модуль відповідає за моніторинг часових витрат, порівнюючи тривалість навчання однієї епохи для обох архітектур, що є важливим для оцінки обчислювальної складності. За допомогою бібліотек pandas та matplotlib реалізовано автоматичну побудову графіків функцій втрат (Box Loss, Classification Loss, DFL Loss) та метрик якості (Precision, Recall, mAP). Це дозволило в реальному часі контролювати стабільність градієнтного спуску та своєчасно діагностувати ефекти перенавчання.

У даній роботі початкові (оригінальні) моделі YOLO позначено як Pre-trained, а моделі YOLO після донавчання позначено Fine-tuned. В розробленому програмному забезпеченні реалізовано можливість виконувати детектування зображень автомобілів за допомогою чотирьох конфігурацій моделей YOLO:

1. Початкова (Pre-trained) YOLOv8m.
2. Початкова (Pre-trained) YOLOv11m.
3. Донавчена (Fine-tuned) YOLOv8m.
4. Донавчена (Fine-tuned) YOLOv11m.

За результатами детектування зображень тестової вибірки обчислено метрики якості для всіх чотирьох конфігурацій моделей YOLO. Для комплексного

порівняльного аналізу ефективності розроблених моделей реалізовано програмний модуль верифікації на незалежній тестовій вибірці. Програмний алгоритм вибирає випадкове зображення з набору даних та послідовно виконує операцію інференсу (детектування) для чотирьох конфігурацій ЗНМ: базових моделей YOLOv8m та YOLOv11m, а також моделей після донавчання. Як видно з результатів детектування, базові моделі (Pre-trained) демонструють певну нестабільність у класифікації зображень об'єктів (рис. 3.8).

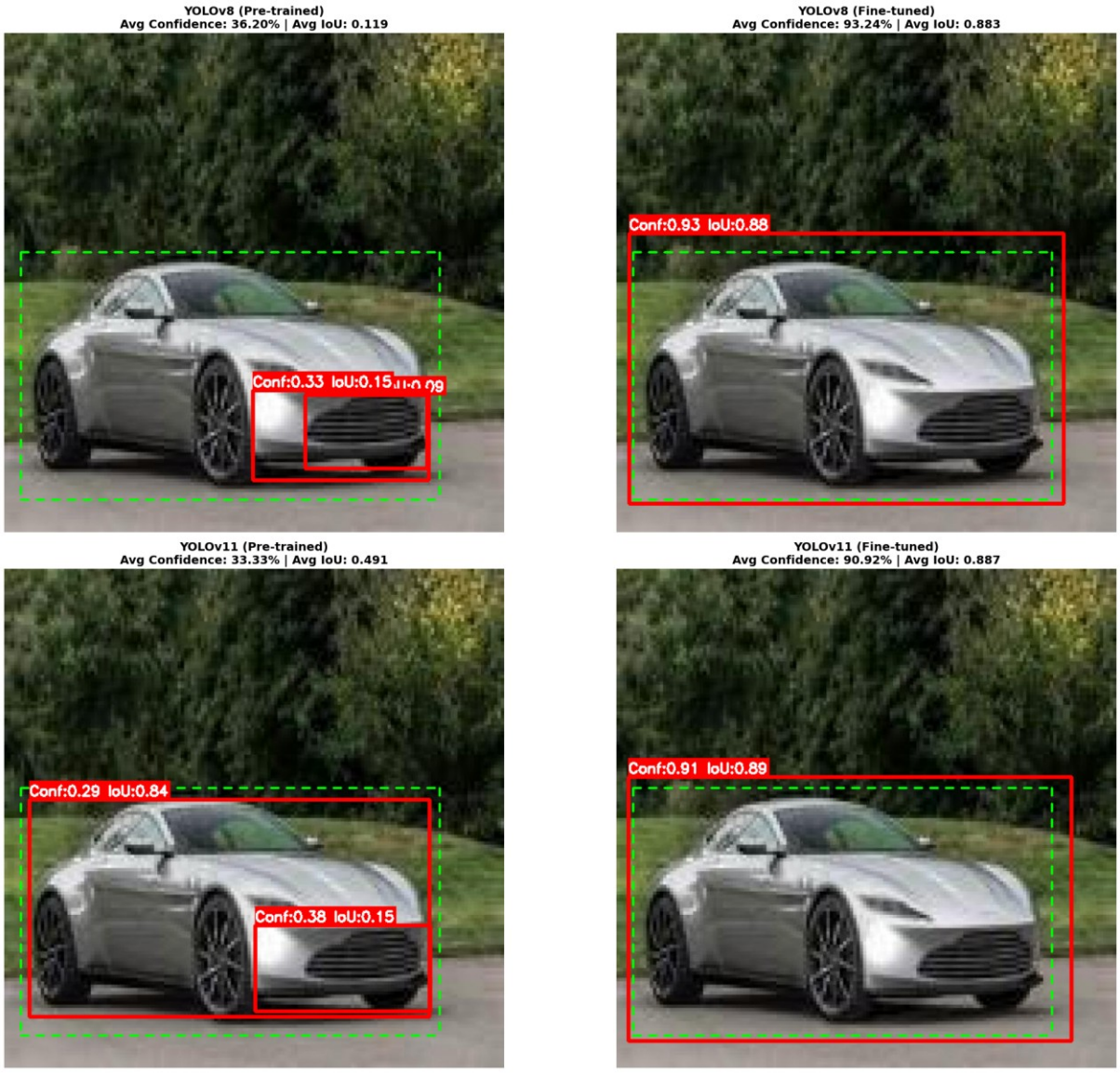


Рисунок 3.8 – Результати детектування автомобілів чотирма конфігураціями моделей YOLO на одному тестовому кадрі розміром 640 × 640 пікселів

Спостерігається значний розкид у впевненості розпізнавання малих об’єктів на задньому плані, де показники коливаються в межах 39 %–51 %. Натомість, донавчена моделі (Fine-tuned) демонструють суттєве покращення метрик впевненості. Для більшості детектованих автомобілів середнього та переднього плану показник впевненості розпізнавання зріс до рівня 88 %–92 %.

Донавчені моделі успішно ідентифікували об'єкти на краях кадру та в умовах щільного трафіку, які були пропущені або низько оцінені базовими моделями. Донавчання на специфічному датасеті дозволило нівелювати помилкові спрацьовування класу «bus» (автобус), зосередивши увагу моделі на цільовому класі «car» (автомобіль), що підтверджується коректним детектуванням фрагментів транспортних засобів навіть на периферії зображення.

Порівняння між моделями YOLOv8m та YOLOv11m після донавчання показує майже ідентичні результати за точністю локалізації, проте YOLOv8m демонструє дещо вищі показники впевненості розпізнавання для об'єктів, які знаходяться у тіні або мають низьку контрастність.

3.4 Аналіз результатів донавчання для різних версій YOLO

За допомогою розробленого програмного забезпечення «FineTunedYOLOCarDetection» проведено донавчання моделей YOLOv8m та YOLOv11m на створеному датасеті, поділеному на навчальну, контрольну (валідаційну) та тестову вибірки.

За результатами навчання модель YOLOv8m порівняно з YOLOv11m показала вищу швидкодію. У середньому одна епоха в YOLOv8m тривала 1 хвилину 14 секунд, а в YOLOv11m одна епоха тривала 1 хвилину 18 секунд (із використанням TPU). Загалом донавчання моделі YOLOv11m тривало 1 годину 21 хвилин, а донавчання YOLOv8m тривало 1 годину 10 хвилин [155].

Після завершення процесу донавчання моделі YOLOv8m (Fine-tuned) та YOLOv11m (Fine-tuned) збережено у файли формату .pt, які містять параметри навченої нейромережі. Розмір цих файлів служить важливим індикатором складності та ефективності моделі. Розміри донавчених моделей такі [155]:

- YOLOv11m (Fine-tuned) – 38,7 МБ.
- YOLOv8m (Fine-tuned) – 49,6 МБ.

Менший розмір файлу дотренованої моделі YOLOv11m (Fine-tuned) свідчить про її оптимізовану архітектуру, яка використовує меншу кількість параметрів та більш ефективні структури шарів. Компактніші моделі потребують менше пам’яті та володіють вищою швидкодією на пристроях із обмеженими ресурсами (наприклад, у вбудованих системах). Висока швидкодія є особливо важливою для детектування зображень у режимі реального часу. Зменшення розміру файлу моделі YOLOv11m до 38.7 МБ при збереженні високої точності є результатом архітектурної еволюції Ultralytics. У новій версії було замінено базові блоки C2f на більш ефективні C3k2, а також оптимізовано структуру просторової піраміди пулінгу [35]. Це робить YOLOv11m більш придатною для розгортання на Edge-пристроях.

З метою оцінки процесу навчання ЗНМ проаналізовано динаміку зміни функцій похибок (англ. Loss functions) у логарифмічному масштабі, що дозволяє краще аналізувати збіжність навчання моделей ЗНМ на етапі тонкого налаштування ваг для навчальної (Train) та валідаційної (Val) вибірок (рис. 3.9 - рис. 3.11).

Аналіз динаміки похибок виявив наступні закономірності. Графік похибок локалізації (англ. Box Loss), який визначає точність побудови обмежувальної рамки (рис. 3.9), показав, що на останній епосі YOLOv11 забезпечила вищу точність детектування зображень контрольної вибірки, досягнувши значення val/box_loss 0.73706 проти 0.74318 у YOLOv8.

Також проведено аналіз значень похибки класифікації (англ. Classification Loss), яка описує точність розпізнавання класу об'єкта (рис. 3.10).

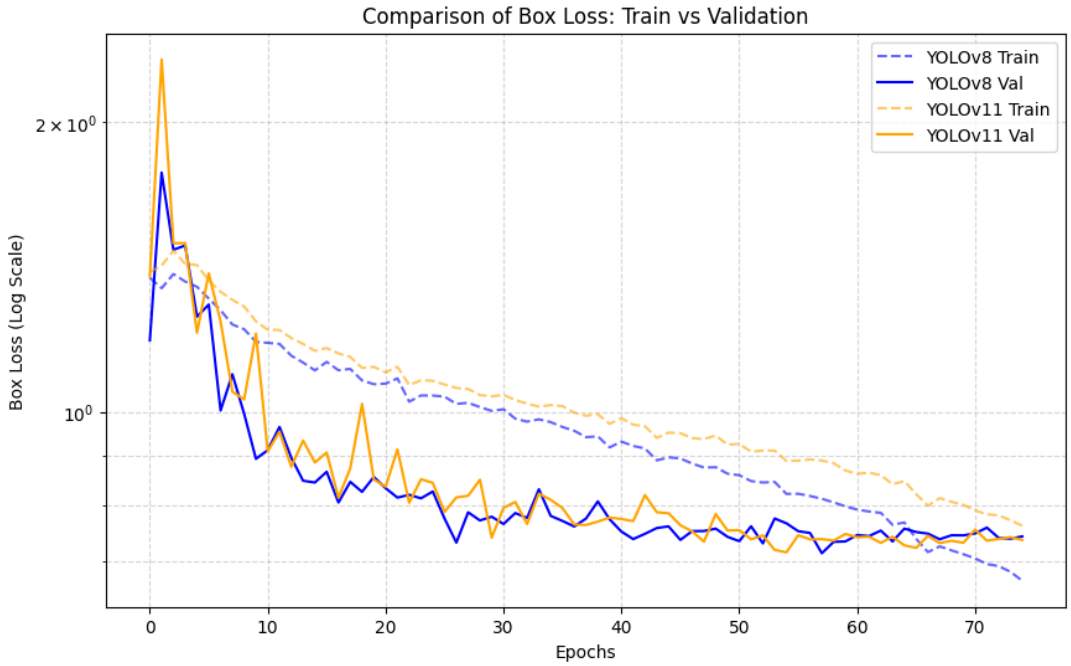


Рисунок 3.9 – Значення функцій похибок Box Loss для донавчених моделей YOLOv8m та YOLOv11m, отримані для валідаційної та навчальної вибірок

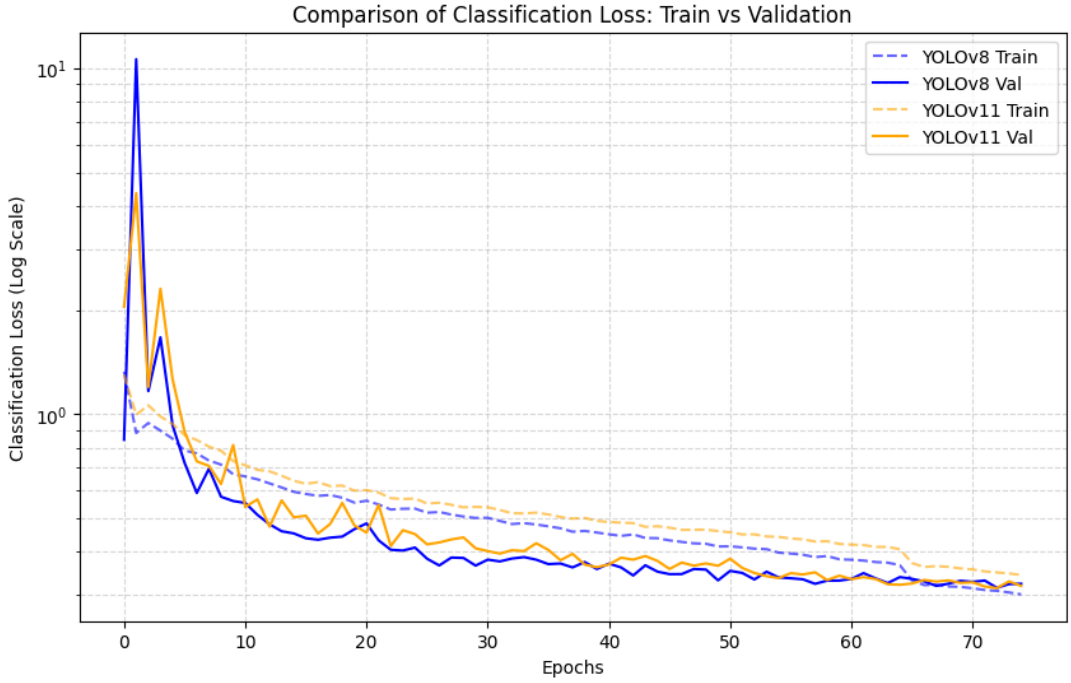


Рисунок 3.10 – Значення функцій похибок Classification Loss для донавчених моделей YOLOv8m та YOLOv11m (валідаційна та навчальна вибірки)

Проведено аналіз похибки DFL Loss (англ. Distribution Focal Loss), яка використовується для уточнення нечітких меж об'єкта (рис. 3.11). Значення val/dfl_loss на рівні 1.28342 для YOLOv11 та 1.31882 для YOLOv8 підтверджують високу точність детектування об’єктив донавченими моделями YOLO.

Обидві моделі YOLO показали стабільне зниження похибок, при цьому YOLOv11 досягла меншого значення для валідаційної вибірки (0.31874), що свідчить про вищу точність розпізнавання саме автомобілів серед інших об'єктів.

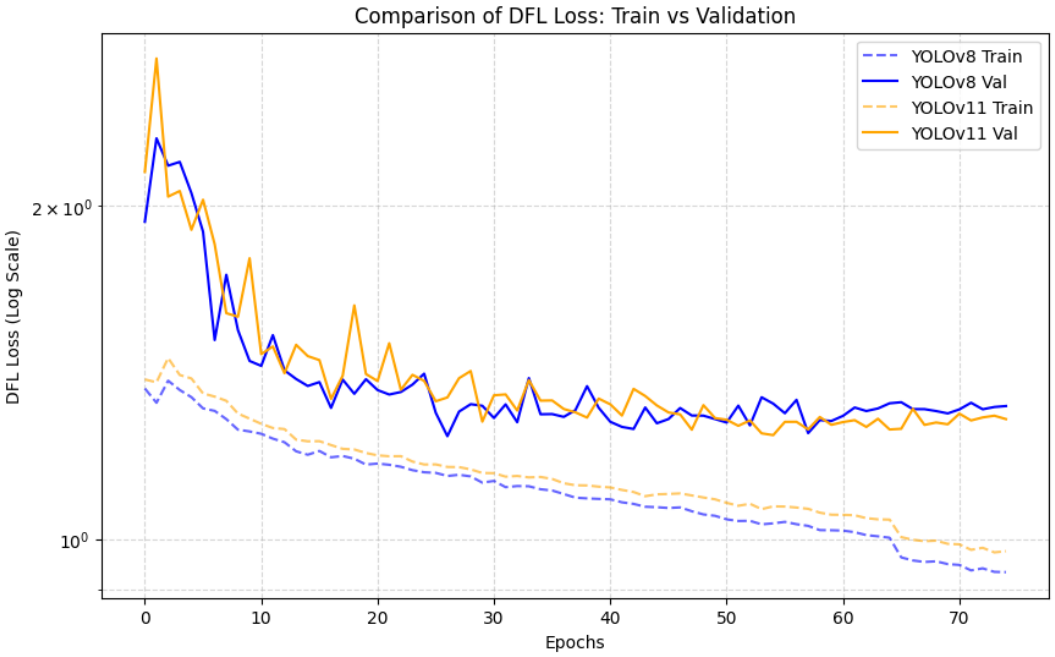


Рисунок 3.11 – Значення функцій похибок DFL Loss для донавчених моделей YOLOv8m та YOLOv11m, отримані для валідаційної та навчальної вибірок

Графік метрики mAP5-95 (рис. 3.12), яка описує точність детектування меж об’єктів, показує спочатку швидше, а потім більш повільне зростання. Кінцеві результати за метрикою mAP5-95 для різних моделей отримано практично однаковими, що свідчить про стабільність навчання.

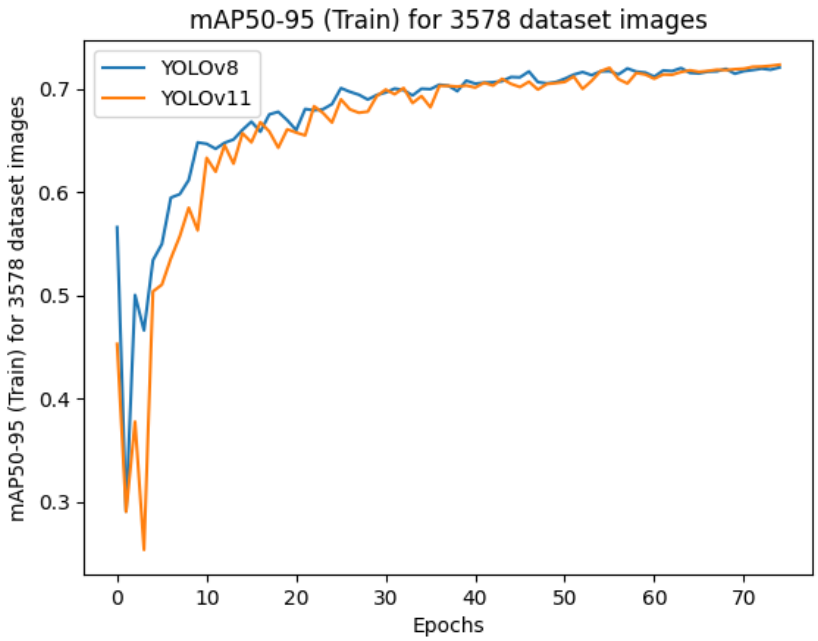


Рисунок 3.12 – Значення метрик mAP90-95 для донавчених моделей YOLOv8m та YOLOv11m, отримані для тренувальної вибірки

У всіх випадках (рис. 3.9 – рис. 3.11) похибки для контрольної (валідаційної) вибірки практично не зменшуються після 50-ї епохи, а значення метрики точності детектування (рис. 3.12) практично не збільшуються після 50-ї епохи, тому вибрана кількість епох (75) є достатньою для донавчання моделей ЗНМ.

3.5 Аналіз результатів детектування для різних версій YOLO

Після донавчання моделей YOLOv8m та YOLOv11m проведено їх тестування на незалежній тестовій вибірці (рис. 3.13). Для об'єктивного порівняння також було протестовано початкові (не донавчені) версії цих моделей. Точність детектування здійснювалася за основними метриками: Precision, Recall, mAP50 та mAP50-95 [155]. Донавчені моделі показали суттєво вищі значення всіх метрик (вищу точність детектування) завдяки адаптації до розробленого датасету (рис. 3.14, табл. 3.1).

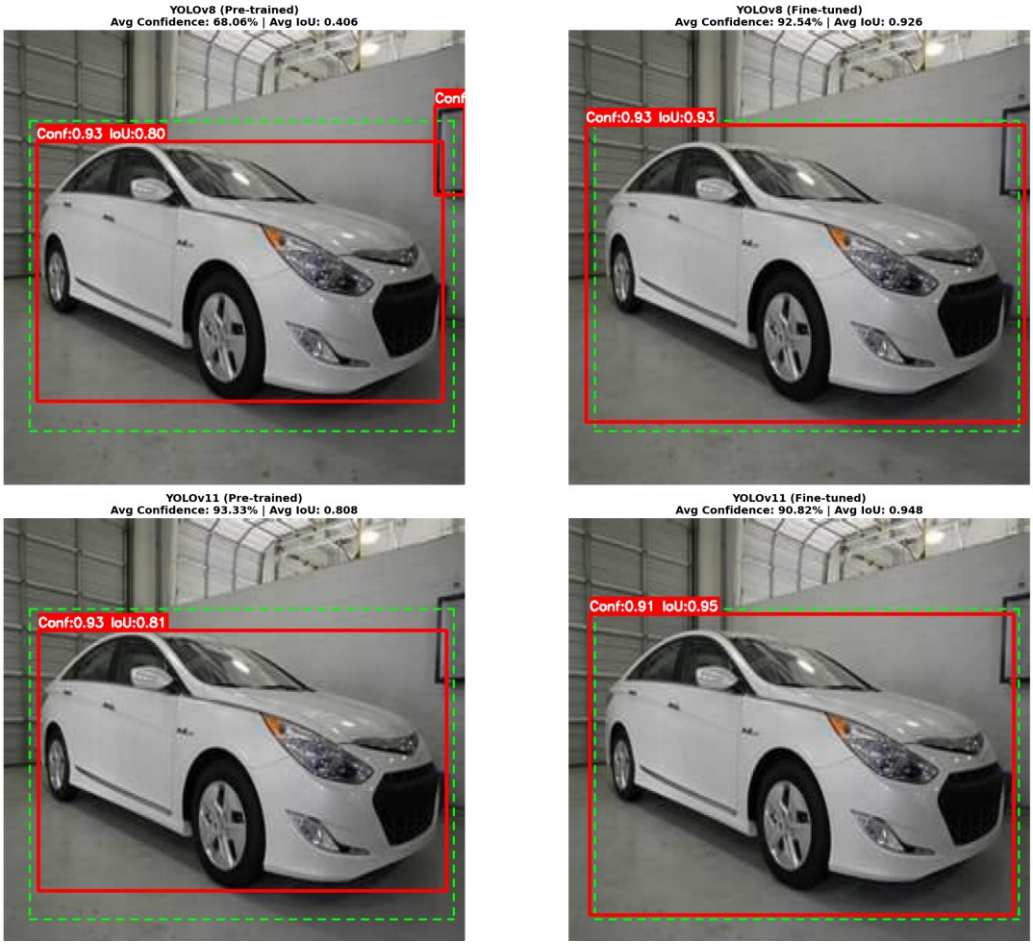


Рисунок 3.13 – Приклад детектування зображення автомобіля базовими (Pre-trained) та донавченними (Fine-tuned) моделями версій YOLOv8m та YOLOv11m

Підвищення значень метрики Recall після донавчання означає, що модель стала краще виявляти всі об’єкти (автомобілі) на зображеннях. Найбільше покращення отримано для метрики mAP50-95 (майже вдвічі з 0.37 до 0.72), що підтверджує значне покращення точності локалізації об’єктів різних розмірів (координат їх обмежувальних рамок) [155].

У контексті систем допомоги водієві це важливо, оскільки висока точність виявлення рамки (IoU > 0.5...0.95) безпосередньо впливає на розрахунок дистанції до автомобіля. Початкові моделі часто детектували об'єкт, але рамка захоплювала зайвий фон, тоді як донавчені моделі (Fine-tuned) навчилися точніше охоплювати контури автомобілів, що мінімізує похибку при подальшій обробці.

Модель YOLOv11m (Fine-tuned) в середньому обробляла зображення швидше, ніж модель YOLOv8m (Fine-tuned). Для YOLOv11m (Fine-tuned) середній час детекції 1 кадру рівний 0.047 с, а для YOLOv8m (Fine-tuned) рівний 0.050 с. Підвищення швидкодії детектування пов'язано з розміром моделі, оскільки дотренована модель YOLOv11m (Fine-tuned) потрібує майже на 10 МБ менше пам'яті, ніж YOLOv8m (Fine-tuned) [155].

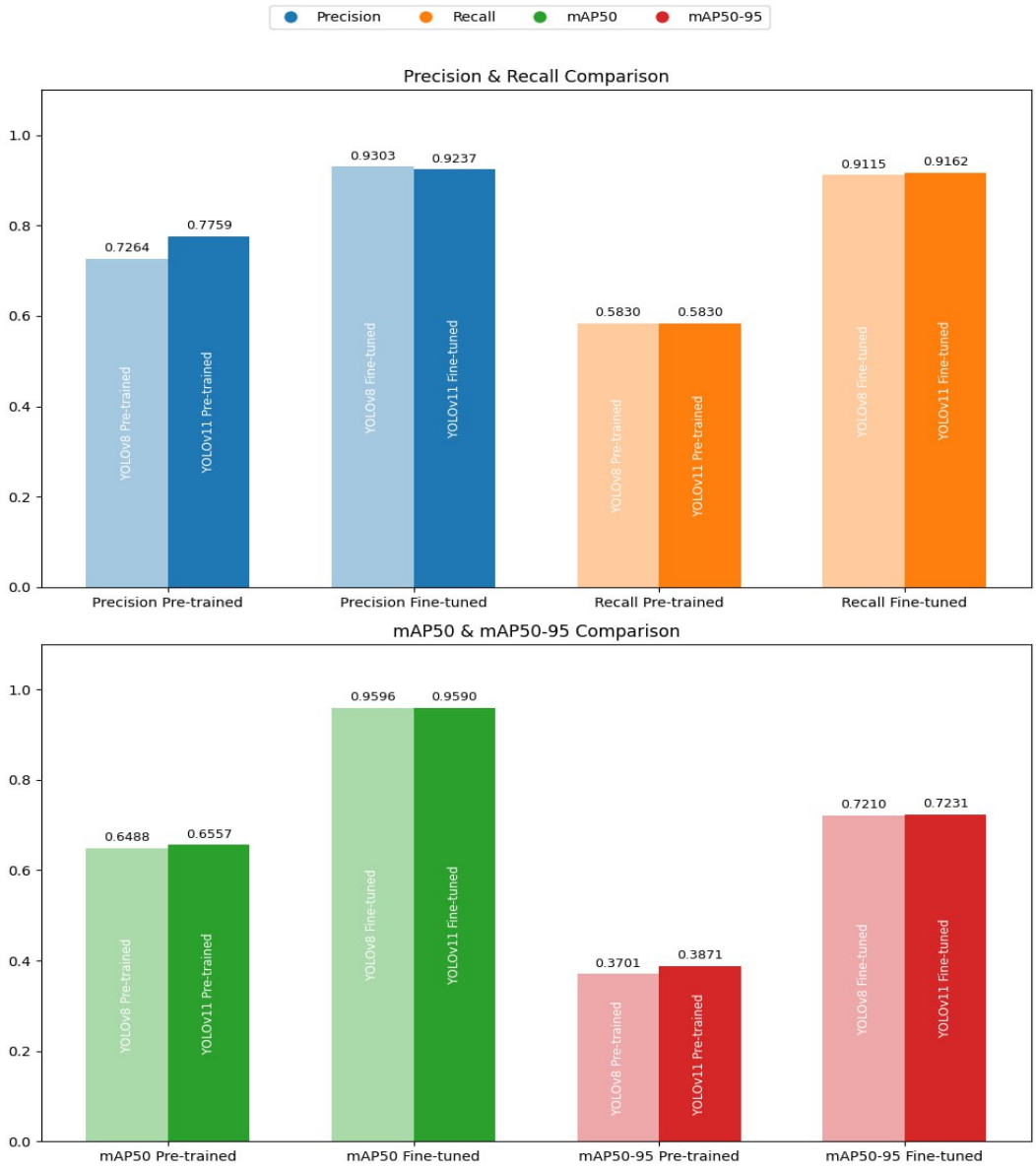


Рисунок 3.14 – Значення метрик Precision, Recall, mAP50 та mAP50-95 для початкових (Pre-trained) та дотренованих (Fine-tuned) моделей YOLO8m та YOLO11m, отримані на тестовому наборі зображень

Таблиця 3.1 – Значення метрик Precision, Recall, mAP50 та mAP50-95 для початкових (Pre-trained) та дотренованих (Fine-tuned) моделей YOLO8m та YOLO11m, отримані на тестовому наборі зображень

Метрика	YOLOv8m (Pre-trained)	YOLOv11m (Pre-trained)	YOLOv8m (Fine-tuned)	YOLOv11m (Fine-tuned)
Precision	0.7264	0.7759	0.9303	0.9237
Recall	0.5830	0.5830	0.9115	0.9162
mAP50	0.6488	0.6557	0.9596	0.9590
mAP50-95	0.3701	0.3871	0.7210	0.7231

Значення метрик точності для моделей YOLOv8m (Fine-tuned) та YOLOv11m (Fine-tuned) після донавчання відрізняються незначно, що в значній мірі зумовлено особливістю використаного датасету (який характеризується високою різноманітністю використаних зображень). Вибір між YOLOv8m та YOLOv11m залежить від конкретного випадку використання. Якщо критичною є висока точність, то доцільно використовувати YOLOv8m. Якщо ж пріоритетом є швидкість та оптимальне використання ресурсів, то має перевагу модель YOLOv11m [155].

Стрімке зростання після донавчання метрики Recall (з 0.58 до 0.91) пояснюється ефектом предметної адаптації. Початкові ваги моделей були сформовані на датасеті COCO, який містить узагальнені зображення транспортних засобів. Однак, сформований навчальний набір з 3054 зображень, що пройшов процедуру аугментації в Roboflow, містить специфічні умови освітлення, ракурси та типи дорожнього покриття, характерні для поставленої задачі. У процесі навчання не виникав ефект перенавчання, що підтверджується стабільністю метрик якості на валідаційній вибірці (420 зображень).

3.6 Трекінг зображень автомобілів засобами нейронних мереж YOLO

Для оцінки ефективності виявлення об'єктів у реальних умовах було протестовано початкові та донавчені моделі YOLOv8m та YOLOv11m шляхом детектування автомобілів на відеозаписах. Крім точності виявлення автомобілів, також аналізувалася якість трекінгу їхнього руху за допомогою бібліотеки OpenCV. Дотреновані моделі чітко відокремлюють автомобілі навіть у складних сценаріях, коли автомобілі знаходяться поруч (рис. 3.15, рис. 3.16). Точність детектування автомобілів на відео дотренованими моделями YOLOv8m (Fine-tuned) та YOLOv11m (Fine-tuned) за використаними метриками (зокрема, за метрикою mAP50-95) практично співпадає [155].

Результати детектування (рис. 3.15, рис. 3.16) показають, що дотренована модель YOLOv8m показує вищу точність виявлення автомобілів у порівнянні з початковою моделлю YOLOv8m. Середня впевненість розпізнавання автомобілів (для всіх кадрів) для донавченої моделі YOLOv8m рівна 0.81, а для початкової моделі YOLOv8m рівна 0.62. Візуально помітно, що дотренована модель (рис. 3.16) забезпечує більш гладкі траєкторії руху автомобілів («плавний трекінг»), які точніше відповідають реальним траєкторіям.

У даному дослідженні плавність трекінгу оцінювалася як показник Jitter [167], який визнався через зміщення центру об'єкта між кадрами. Для кожного автомобіля визначалися координати центру обмежувальної рамки (x, y) у кожному кадрі. Швидкість переміщення об'єкта (пікселі/кадр) з ідентифікатором *id* на кадрі з номером *k* визначалася через відстань, яку подолав об'єкт між послідовними кадрами з номерами *k* та *k*-1, за формулою:

$$v_k = \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}, \tag{3.1}$$

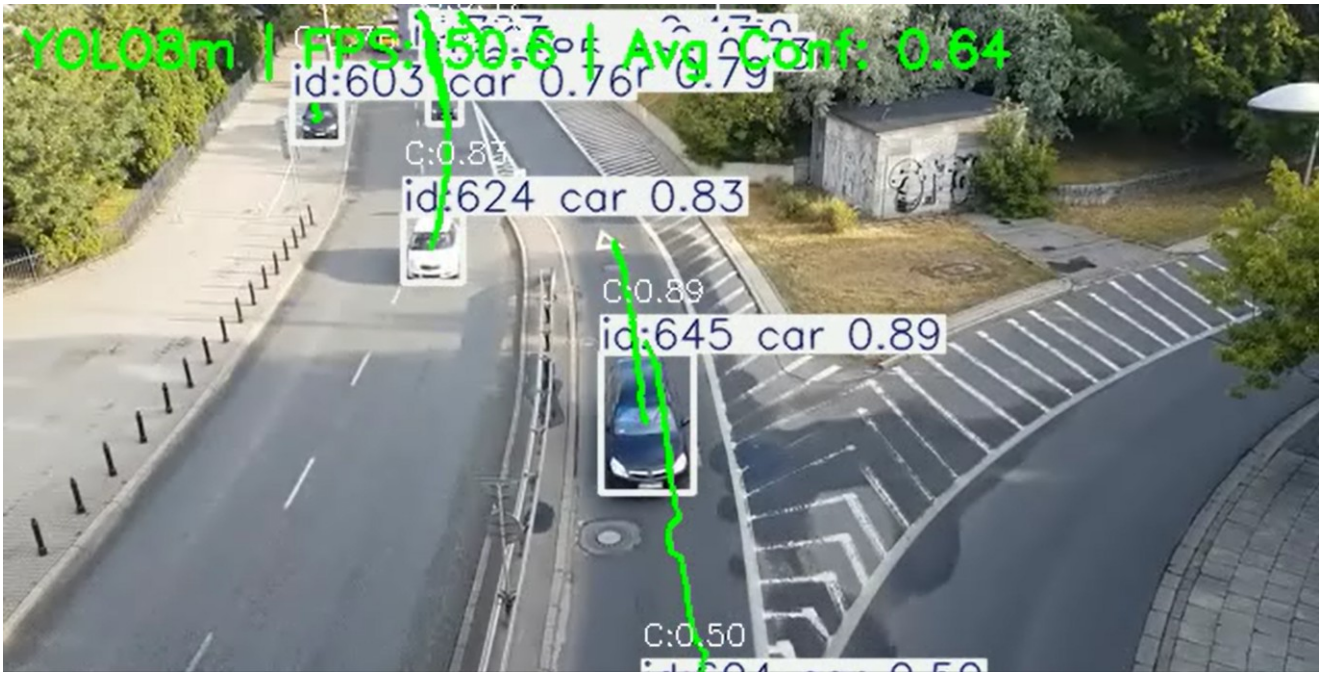


Рисунок 3.15 – Кадр відео з детекцією та трекінгом автомобілів за допомогою початкової моделі YOLOv8m [167]

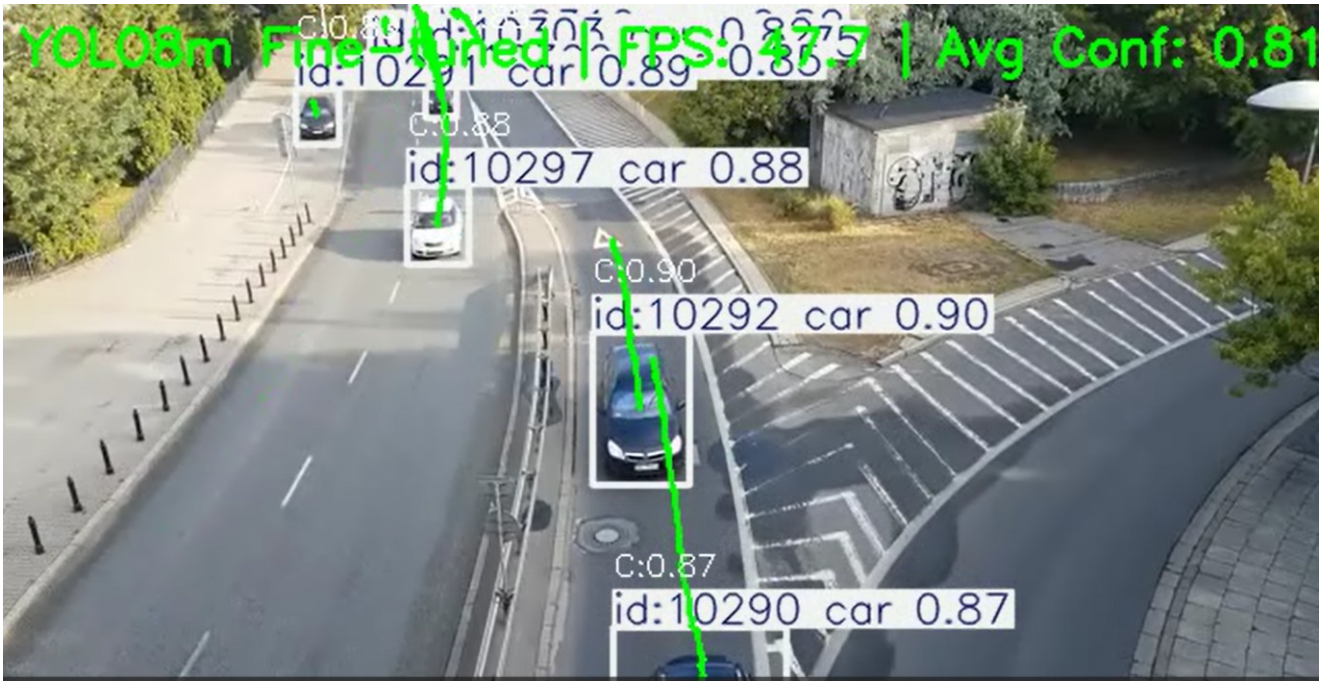


Рисунок 3.16 – Кадр відео з детекцією та трекінгом автомобілів за допомогою дотренованої моделі YOLOv8m (Fine-tuned) [167]

Об'єкти, що перебували у кадрі менше 10 кадрів, виключалися з аналізу для усунення впливу випадкових шумів та короткочасних детекцій. Показник Jitter (рис. 3.17) розраховувався як середнє квадратичне відхилення швидкості для кожного детектованого об’єкту з ідентифікатором *id* за формулою:

$$Jitter = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(v_k - \bar{v})^2} \ , \tag{3.2}$$

де $\bar{v}$ – середня швидкість об'єкта за весь час спостереження; N – кількість кадрів.

Горизонтальна вісь (рис. 3.17) відображає чотири досліджувані конфігурації нейромереж, вертикальна вісь – значення показника плавності трекінгу (Jitter).

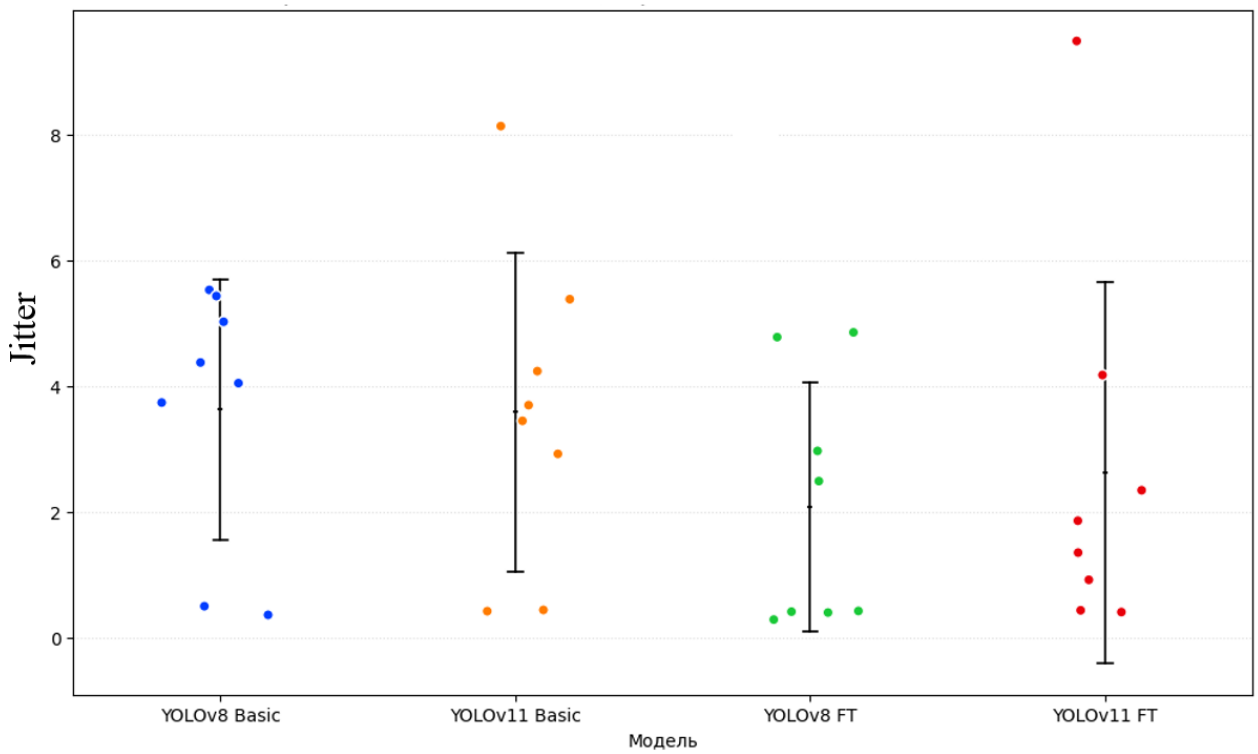


Рисунок 3.17 – Порівняльний аналіз плавності трекінгу (показник Jitter) для початкових (Basic) та донавчених (FT) моделей YOLOv8m та YOLOv11m

Кожна точка (маркер) показує значення Jitter для окремого транспортного засобу (досліджено $n = 8$ транспортних засобів для кожної моделі). З метою уникнення перекривання такі точки випадково зміщені вздовж горизонтальної осі. Для кожної моделі ЗНМ обчислено середнє арифметичне JitterA для показника Jitter та його середнє квадратичне відхилення σ_J. Інтервал значень для показника Jitter визначено через його нижню (JitterA - σ_J) та верхню (JitterA + σ_J) межі. Такий інтервал дозволяє аналізувати індивідуальну поведінку трекера на різних об'єктах.

Високе значення Jitter вказує на різкі стрибки рамки навколо детектованого об'єкта, що свідчить про низьку стабільність моделі. Чим менше значення Jitter, тим плавнішою є траєкторія об’єкту. Моделі після донавчання (Fine-tuned) демонструють тенденцію до зниження Jitter порівняно з базовими. Зокрема, у YOLOv8 (Fine-Tuned) спостерігається найнижче положення середнього значення JitterA, що свідчить про найбільш стабільне утримання об'єктів. Менше значення σ_J у моделі YOLOv8 (Fine-Tuned) вказують на високу повторюваність результату детектування. Натомість YOLOv11 (Basic) має значний розкид Jitter, що говорить про нестабільну роботу на різних типах транспортних засобів. Поодинокі високі точки (наприклад, у YOLOv11 Fine-Tuned) вказують на специфічні сценарії, де трекер тимчасово втрачав об'єкт або некоректно змінював розмір рамки. Найбільш ефективною моделлю за критерієм плавності трекінгу є YOLOv8 (Fine-Tuned). Таким чином, завдяки донавчанню моделей YOLO вдалося підвищити точність виявлення та відстеження автомобілів у порівнянні з початковими моделями [155].

Висновки до розділу 3

У третьому розділі розроблено методику донавчання моделей згорткових нейронних мереж сімейства YOLO різних версій, призначених для детектування

зображень автомобілів, із ручним формування набору даних. За результатами виконаних робіт сформульовано наступні висновки:

1. Розроблено структуру інтелектуальної системи для детектування зображень автомобілів із донавчанням нейронних мереж YOLO на основі набору даних, створеного в ручному режимі. Засобами платформи Roboflow створено спеціалізований датасет зображень автомобілів, який складається з навчальної вибірки (70 % датасету), контрольної вибірки (20 %) та тестової вибірки (10 %). Шляхом аугментації зображень, використовуючи геометричні та колористичні перетворення, обсяг датасету збільшено до 3578 зображень. Аугментація забезпечила достатню варіативність для навчання моделі ЗНМ та мінімізацію ризиків перенавчання.

2. Розроблено програму «FineTunedYOLOCarDetection» на мові Python для трансферного навчання (донавчання) нейромереж YOLO, призначених для детекції об’єктів. Як початкові використано моделі YOLOv8m та YOLOv11m, попередньо навчені на датасеті COCO. Використання хмарного середовища Google Colab з графічним процесором NVIDIA Tesla T4 забезпечило прискорення навчання майже у 40 разів порівняно з використанням центрального процесора CPU (з 1 год. 9 хв. до 1 хв. 37 с за епоху). Реалізована стратегія динамічного розміру пакета (batch size) дозволила використовувати 80–90% відео пам’яті, що є важливим для роботи з моделями ЗНМ середнього розміру.

3. Експериментально підтверджено, що донавчання (англ. fine-tuning) базових моделей на розробленому спеціалізованому датасеті автомобілів дозволяє значно підвищити точність детектування об’єктів. У результаті донавчання отримано збільшення метрик точності детектування, зокрема, для моделі YOLOv8m значення метрики Recall зросли з 0.58 до 0.91, а метрики mAP50-95 з 0.37 до 0.72. Для моделі YOLOv11m значення метрики Recall зросли з 0.58 до 0.91, а метрики mAP50-95 з

0.39 до 0.72. Це свідчить про успішну адаптацію моделей ЗНМ до конкретних умов освітлення та ракурсів зйомки.

4. Встановлено, що новітня модель YOLOv11m, порівняно з YOLOv8m, є більш ефективною для розгортання на мобільних пристроях, оскільки при зіставній точності детектування (mAP50 ≈ 0.96 для обох моделей), YOLOv11m має на 22% менший розмір файлу ваг (38.7 МБ проти 49.6 МБ) та вищу швидкість детектування (0.047 с проти 0.050 с на кадр).

5. Підтверджено придатність моделей YOLOv8m та YOLOv11m для детектування об’єктів у реальному часі. Результати детектування автомобілів у відеопотоці продемонстрували стабільність трекінгу та високу точність обмежувальних рамок об’єктів, що є важливим для коректної оцінки дорожньої ситуації. Вибір між моделями ЗНМ залежить від пріоритетів системи: YOLOv8m забезпечує незначну перевагу в точності детектування, а YOLOv11m потребує менше ресурсів і рекомендується для Edge-обчислень.

РОЗДІЛ 4
ДОНАВЧАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ YOLO ДЛЯ ДЕТЕКТУВАННЯ ЗОБРАЖЕНЬ АВТОМОБІЛІВ ІЗ АВТОМАТИЗОВАНИМ ФОРМУВАННЯМ ДАТАСЕТУ

Створення якісного датасету є важливим етапом при донавчанні нейронних мереж для задач комп'ютерного зору, зокрема для детектування зображень автомобілів. Вимоги для якісного датасету передбачають, що зображення повинні були унікальними, представляти різні дорожні ситуації, погодні умови, освітлення, трафік та ракурси. Проте, створення набору даних значного розміру в ручному режимі є надто трудомістким. Автоматизоване отримання знімків з відеокамер створює ризик накопичення подібних кадрів («дублів»), на яких автомобілі зберігають своє положення протягом багатьох кадрів, що може призвести до перенавчання моделі та зниження її здатності до узагальнення. Тому розроблено методику автоматизованого формування набору зображень, яка дозволяє шляхом селекції видаляти надлишкові кадри [168]. Також реалізовано методику автоматизованого донавчання згорткової нейронної мережі YOLO малого розміру (Учень) на основі спеціалізованого датасету зображень, створеного ЗНМ YOLO середнього або великого розміру (Вчитель). Методики автоматизованого формування датасету та донавчання ЗНМ реалізовано як складову частину інтелектуальної системи, призначеної для детектування зображень автомобілів.

4.1 Структура інтелектуальної системи для детектування зображень автомобілів із автоматизованим донавчанням нейронних мереж

Структуру інтелектуальної системи для детектування зображень автомобілів із ручним формуванням датасету (рис. 3.1) доповнено модулем автоматизованого

створення (генерування) датасету, у результаті чого отримано структуру інтелектуальної системи для автоматизованого створення датасету та донавчання ЗНМ YOLO (рис. 4.1). Автоматизоване створення датасету виконується шляхом відбору з відеопотоку унікальних кадрів із використанням мережі-вчителя YOLOv8m (середнього розміру).

Рисунок 4.1 – Структура інтелектуальної системи для автоматизованого створення датасету та донавчання нано мережі YOLOv8n за допомогою мережі-вчителя YOLOv8m

На основі створеного датасету виконується донавчання мережі-учня YOLOv8n (нано розміру), яка споживає значно менше ресурсів (порівняно з мережею-вчителем). Модуль автоматизованого створення датасету складається з блоків глобальної селекції кадрів, детектування автомобілів мережею-вчителем, селекції кадрів шляхом порівняння детектованих рамок автомобілів з існуючими рамками датасету за параметрами їх подібності (IoU, RMSE) та локальної селекція кадрів шляхом порівняння перцептивних хешів рамок автомобілів з існуючими. Відібрані таким чином кадри додаються до датасету.

4.2 Методика автоматизованого формування набору зображень

Розроблена методика автоматизованого формування набору зображень описується таким алгоритмом (рис. 4.2).

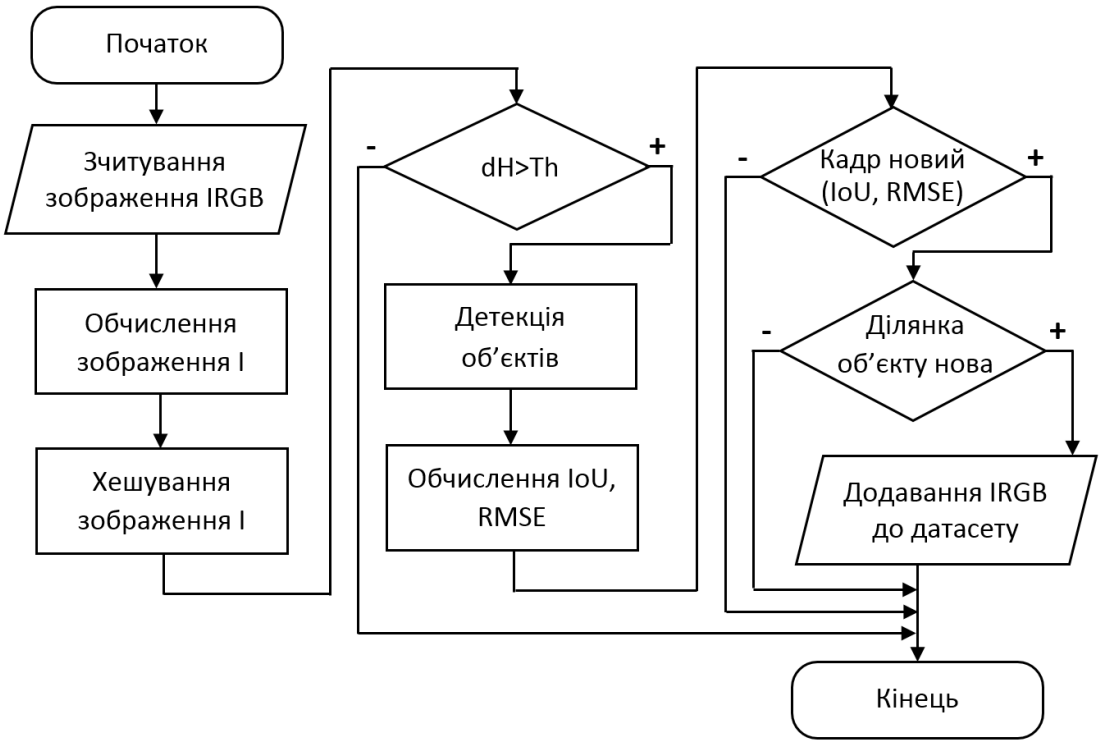


Рисунок 4.2 – Схема алгоритму створення спеціалізованого датасету для зображень автомобілів [30]

Початкові зображення I_{RGB} отримуються з відеокамери як кадри відеопотоку через заданий інтервал часу (наприклад, кожні 5 секунд). У відеопотоці багато кадрів є подібними, тому початкові зображення додаються до датасету тільки після проходження багатоетапної перевірки на унікальність. Згідно з алгоритмом кольорове зображення I_{RGB} (розміром $M_0 \times N_0$ пікселів) перетворюється у зображення у відтінках сірого I, яке масштабується до розміру $N \times N$ пікселів. Після цього виконується перцептивне хешування зображення I і його перевірка на унікальність. На першому етапі зображення вважається новим, якщо різниця d_H його хешу та хешу зображення з датасету перевищує поріг T_h. На другому етапі перевірки виконується детекція автомобілів на зображенні за допомогою ШНМ YOLO (середнього розміру). Для рамок детектованих автомобілів на початковому зображенні та на зображенні датасету обчислюється метрика IoU їх перекривання та корінь середньої квадратичної похибки (RMSE) між ділянками автомобілів. Кадр вважається новим, якщо значення IoU менше заданого порогу або значення RMSE перевищує поріг. На третьому етапі зображення вважається новим, якщо його ділянки автомобілів є новими з урахуванням їх хешів. У випадку успішного проходження всіх перевірок зображення I_{RGB} додається до датасету.

Результатом роботи алгоритму є унікальні зображення, на яких положення детектованих автомобілів вказується прямокутними рамками. Виявлення об’єктів на зображеннях виконується попередньо навченою моделлю №1 ШНМ yolov8m (середнього розміру), оскільки така модель забезпечує високу точність детекції об’єктів навіть у складних сценах. Окремі зображення датасету, за потреби, коректуються в ручному режимі. Отриманий спеціалізований датасет використовується для донавчання (fine-tuning) моделі №2 ШНМ yolov8n (розміру нано). У результаті після донавчання модель ШНМ малого (нано) розміру повинна не тільки забезпечувати точність на рівні моделі середнього розміру (при детектуванні автомобілів), але й споживати менше обчислювальних ресурсів і

демонструвати при цьому вищу швидкодію. Селекція зображень зменшує розмір датасету, що сприяє зменшенню часу навчання ШНМ малого розміру. Такий датасет зображень з детектованими автомобілями може використовуватися для навчання моделей ШНМ YOLO різних версій та розмірів за допомогою бібліотеки Ultralytics. Таким чином, в значній мірі автоматизується не тільки процес зчитування зображень з відеокамери та формування датасету, але й донавчання ШНМ. Ручне втручання дослідника потрібно тільки для періодичної перевірки точності детектування засобами донавченої ЗНМ [168].

4.2.1 Глобальна селекція кадрів з використанням перцептивного хешування

Розглянемо детальніше етап №1 відбору кадрів. На реальних зображеннях дорожнього руху, отриманих з відеокамери, багато кадрів є подібними і містять зображення тих самих автомобілів. Тому з метою усунення надлишковості даних виконується глобальна селекція зображень з використанням перцептивного хешування (perceptual hashing – pHash) [168-170]. На відміну від криптографічних хешів (SHA-256, MD5), перцептивні хеші нечутливі до незначних змін освітлення, масштабу, шумів, компресії або невеликих геометричних трансформацій зображень. Перцептивні хеші мають суттєву перевагу при виявленні унікальних зображень у порівнянні з безпосереднім порівнянням зображень, оскільки зміни освітлення та шуми призводять до певної різниці яскравості для різних зображень навіть тих самих об’єктів. У той же час перцептивні хеші будуть змінюватися при суттєвих змінах на зображеннях, зокрема, при переміщенні автомобілів. Крім цього, розмір хешу значно менший за розмір зображення, тому використання хешів підвищує швидкість селекції зображень.

Глобальна селекція зображень за допомогою перцептивного хешування полягає в наступному. Для кожного нового зображення обчислюється його хеш (глобальний для всього зображення) і порівнюється з хешами зображень, які вже містяться в датасеті. До датасету додаються тільки зображення з унікальними хешами, тобто зі значними змінами відносно зображень датасету (наприклад, зображення з новим автомобілем).

Перцептивне хешування pHash [169, 170] базується на двовимірному дискретному косинусному перетворенні (Discrete Cosine Transform – DCT). Перед таким перетворенням початкове кольорове зображення перетворюється у відтінки сірого і масштабується до розміру $N \times N$ пікселів, у результаті чого отримується зображення $I(x, y)$, де x, y – координати пікселів зображення за шириною та висотою відповідно. DCT виконується над зображенням $I(x, y)$ за формулою [168]:

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x,y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right], \qquad (4.1)$$

де C(u, v) – частотна матриця;

u, v – значення частот за шириною та висотою відповідно;

$\alpha(u) = \sqrt{\frac{1}{N}}$ для $u = 0$ і $\alpha(u) = \sqrt{\frac{2}{N}}$ для $u > 0$.

В матриці $C(u, v)$ аналізується низькочастотний блок розміром 8 × 8 елементів, який відображає загальну структуру сцени. Кожен коефіцієнт низьких частот $C(u, v)$ порівнюється з медіанним значенням блоку, у результаті чого формується бінарний хеш $h \in 0{,}1^{64}$. Таким чином, хеш зберігає загальну структуру зображення, а не абсолютні значення яскравості пікселів. Для порівняння двох зображень використовується різниця між 64-бітними хешами h_1 та h_2 зображень, яка обчислюється як відстань Хеммінга (Hamming distance):

$$d_{H(h_1,h_2)} = \sum_{\{i=1\}}^{\{64\}} 1\left[h_1^{\{(i)\}} \neq h_2^{\{(i)\}}\right] \quad (4.2)$$

де i – номер елемента хеша; $d_H = 0$ означає ідентичні кадри;

$d_H = 64$ означає повністю різні зображення.

Ключовим параметром селекції зображень є поріг відстані Хеммінга T_h, який визначає подібність двох зображень. Нове зображення з хешом h_{new} вважається відмінним від зображення в датасеті з хешем h_{stored}, якщо

$$d_H(h_{new}, h_{stored}) > T_h. \quad (4.3)$$

Якщо умова (4.3) виконується для всіх зображень датасету, то нове зображення успішно проходить глобальну селекцію з використанням перцептивного хешування. У практичній реалізації поріг T_h обрано експериментально шляхом перебору його значень. Для кожного значення T_h визначалася кількість кадрів, які відкидалися як дублікати або оцінювалися як унікальні. Найкращий результат отримано для порогу $T_h = 2$, при якому кадри з однаковою сценою та автомобілями відкидаються, а унікальні кадри з новими або зміщеними автомобілями оцінюються як унікальні.

Для верифікації ефективності обраного методу селекції проведено візуальний аналіз результатів роботи алгоритму pHash на реальних даних з відеопотоку. Метою експерименту було продемонструвати здатність алгоритму ігнорувати незначні шуми та зміни, зберігаючи при цьому чутливість до суттєвих структурних змін сцени (зміна часу доби, освітлення, поява нових об'єктів). Візуалізація базується на зворотному перетворенні 64-бітного хеш-рядка у бінарну матрицю розмірністю 8×8. Кожна клітинка цієї матриці відповідає певному частотному компоненту зображення після дискретного косинусного перетворення (DCT): чорний колір позначає біт "0", а білий – біт "1".

Розглянемо результати перцептивного хешування для трьох послідовних кадрів, які візуально сприймаються як ідентичні (однакова сцена, статичне положення камери, незмінне освітлення) (верхній ряд на рисунку 4.3). Незважаючи на те, що на піксельному рівні ці зображення можуть мати незначні відмінності (через шум сенсора камери, мікро-рухи або затінення), згенеровані бінарні матриці (чорні та білі блоки) хеш-рядків є майже ідентичними (нижній ряд на рисунку 4.3). Це підтверджує, що алгоритм pHash працює як фільтр низьких частот: відкидає високочастотний шум і фокусується на глобальній структурі сцени (автомобілі, контури будівель, лінія дороги та ін.).

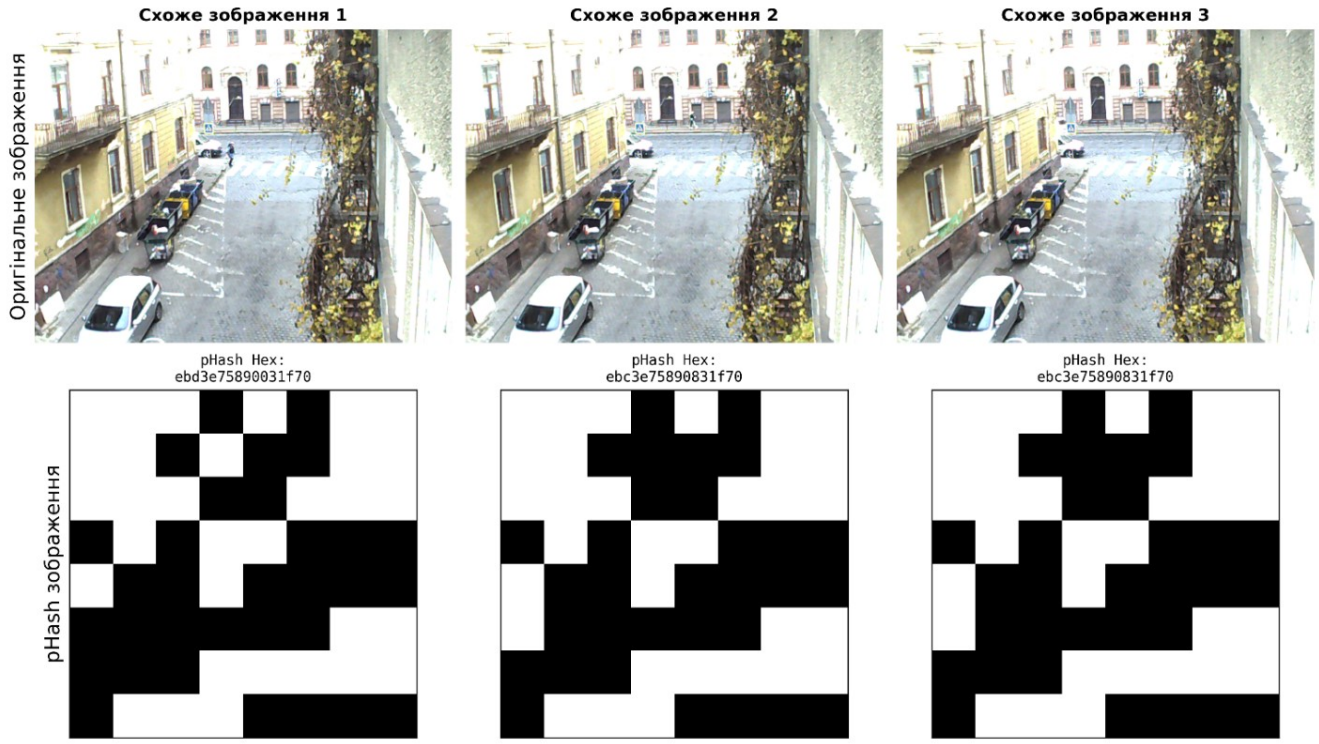


Рисунок 4.3 – Приклад результатів перцептивного хешування pHash для схожих зображень

У випадку суттєво відмінних кадрів, отриманих в різний час доби ("Вечір", "День" та "Сутінки"), результати перцептивного хешування також є різними

(рис. 4.4). Хоча фізичне положення відеокамери залишалося незмінним, проте на зображеннях виникли значні зміни (верхній ряд на рисунку 4.4), які спричинені:

1. Зміною освітлення, оскільки перехід від денного світла до штучного вуличного освітлення змінює контрастність ключових об'єктів. Зокрема вікна, які були темними вдень, стають яскравими джерелами світла вночі.

2. Тінями, оскільки довгі тіні в сутінках створюють нові геометричні форми, які фіксуються алгоритмом pHash.

Як наслідок, бінарні матриці хеш-рядків (нижній ряд на рисунку 4.4) демонструють суттєву відмінність.

У роботі застосовано різні значення порогу відстані Хеммінга $T_h = 1,2,3,\ldots$ і проаналізовано вплив Th на отримані хеші. На рисунку 4.3, наприклад, хеші відрізняються на 1 елемент.

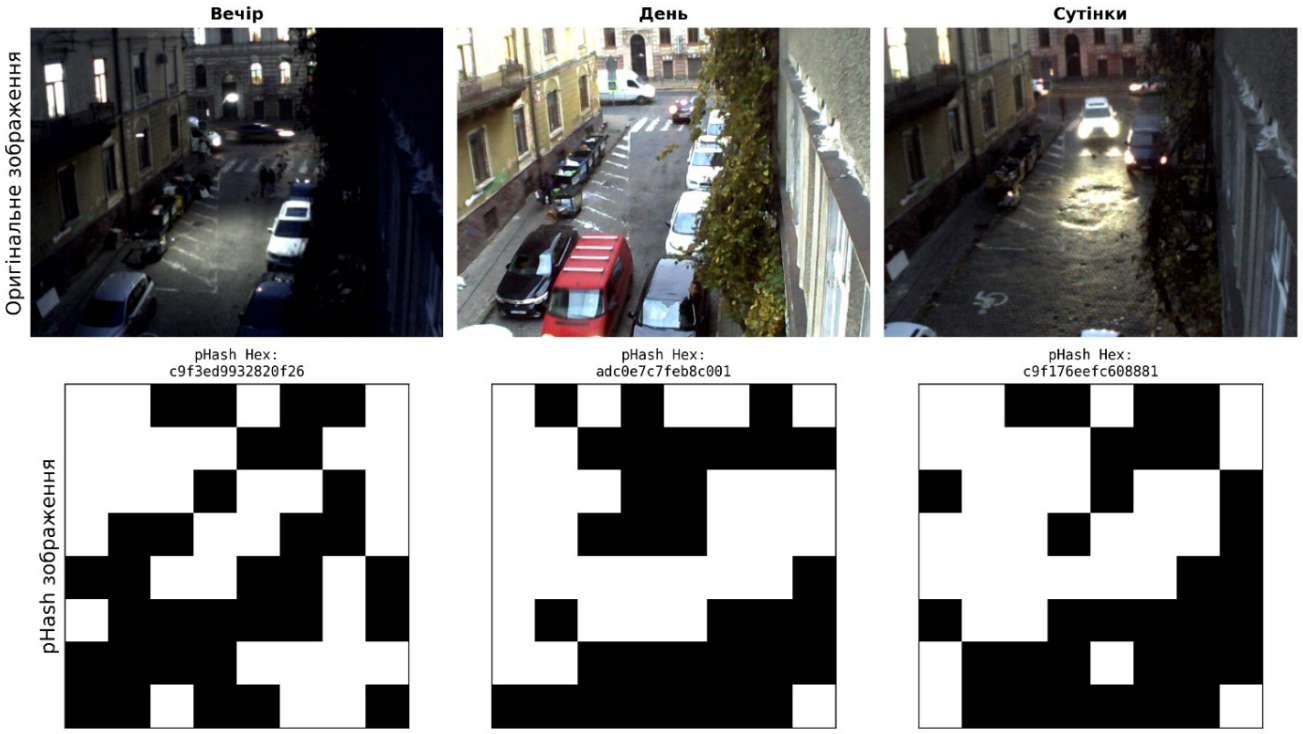


Рисунок 4.4 – Приклад результатів перцептивного хешування pHash для суттєво різних зображень

Найкращі результати селекції кадрів отримано для порогу відстані Хеммінга $T_h = 2$ [168]: кадри з подібними сценами та автомобілями відкидаються, а кадри з новими або зміщеними автомобілями залишаються у датасеті. Алгоритм pHash забезпечує необхідний баланс між ігноруванням незначних флуктуацій (стабільність) та детекцією суттєвих змін середовища (чутливість).

У процесі глобальної селекції кожне нове зображення перетворювалося в хеш та порівнювалося з уже збереженими хешами зображень у базі даних Redis. Використання Redis забезпечує швидкий доступ до даних та масштабованість, оскільки в такому випадку дані зберігаються в оперативній пам’яті. Також, якщо датасет вже має велику кількість збережених унікальних хешів, то такі кеші можна аналізувати по частинам у межах ковзного вікна порівняння (sliding window), наприклад, розміром 200 кешів. Після швидкодіючої глобальної селекції з використанням перцептивного хешування, під час якої відкидається більшість кадрів-дублікатів, відібрані зображенні проходять додаткові перевірки на унікальність. Інші етапи селекції зображень потребують більше обчислювальних ресурсів (порівняно з етапом №1), тому застосування глобальної селекції забезпечує високу швидкодію процесу відбору унікальних зображень у датасет.

4.2.2 Селекція кадрів із використанням детектування та аналізу зображень автомобілів

Етап №2 відбору зображень, який полягає в детектуванні та аналізі автомобілів на зображеннях, дає змогу залишити тільки ті зображення, на яких спостерігаються зміни досліджуваних об’єктів, а не фону. Така перевірка потрібна, оскільки частина зображень після глобальної селекції відрізняються від зображень датасету тільки фоном, що може бути зумовлено змінами освітлення, погодними умовами, рухом сторонніх об’єктів (не автомобілів).

Етап №2 відбору зображень містить такі кроки:

1. Детекція автомобілів на зображенні за допомогою моделі ЗНМ YOLO, а результаті чого обчислюються координати прямокутних обмежувальних рамок (bounding box) автомобілів.
2. Порівняння координат рамок автомобілів на новому зображенні з координатами рамок автомобілів для зображень, які містяться в датасеті, за допомогою угорського алгоритму та з урахуванням метрики IoU (Intersection over Union – Перетин над Об'єднанням) перекривання рамок [171].
3. Порівняння значень кольору ділянок автомобілів на новому зображенні зі значеннями кольору ділянок автомобілів для зображень, які містяться в датасеті; зображення автомобіля в межах рамки вважається новим, корінь середньої квадратичної похибки (Root Mean Square Error – RMSE) значень кольору двох ділянок перевищує заданий поріг T_{RMSE}. Така перевірка дозволяє виявити на кадрах нові автомобілі, навіть якщо їхні рамки просторово збігаються з рамками автомобілів для зображень датасету.

Кадр вважається унікальним і переходить на наступний етап селекції, якщо відсоток унікальних рамок автомобілів (за їх координатами або за зображеннями автомобілів у межах рамок) перевищує заданий поріг (наприклад, 30%).

Угорський алгоритм, також відомий як алгоритм Куна-Мункреса, є ефективним для вирішення задачі оптимального співставлення в двочастковому графі [172-175]. У контексті порівняння зображень, цей алгоритм використовується для асоціації виявлених об'єктів між двома кадрами, мінімізуючи або максимізуючи значення метрики *IoU* (2.11). Значення IoU змінюється від 0 до 1, де 1 означає повне перекривання рамок. У задачі порівняння нового зображення з попереднім кадром значення IoU використовується для оцінки подібності між просторовими рамками об'єктів. Угорський алгоритм застосовується для оптимального співставлення рамок об’єктів між двома кадрами, мінімізуючи їх сумарне значення IoU. Якщо сумарне

значення IoU двох кадрів перевищує певний поріг T_{IoU}, то об'єкти на ньому вважаються ідентичними, а такий кадр не вважається новим. Експериментальним методом встановлено значення порогу $T_{IoU} = 0.5$. Унікальними вважаються кадри, для яких сумарне значення IoU не перевищує поріг T_{IoU}.

Проте, у деяких випадках координати рамок автомобілів співпадають на двох зображеннях, але автомобілі є різними. У таких випадках виконується перевірка співпадіння значень кольору ділянок зображень об’єктів (розміром $M_b \times N_b$ пікселів) за критерієм RMSE, який обчислюється за формулою:

$$RMSE = \sqrt{\frac{1}{M_b \cdot N_b \cdot Q_C} \sum_{i=0}^{Mb-1} \sum_{k=0}^{Nb-1} \sum_{c=0}^{Q_C-1} \left(I(i,k,c) - I_D(i,k,c)\right)^2}, \qquad (4.4)$$

де $I(i, k, c)$ – значення каналу кольору c для пікселя нового зображення з координатами (i, k) за висотою та шириною відповідно;

$I_D(i, k, c)$ – значення каналу кольору c для пікселя (i, k) зображення датасету;

Q_C – кількість каналів кольору (RGB – червоний, зелений, синій).

Якщо RMSE між рамками двох автомобілів перевищує визначений поріг T_{RMSE}, та зображення автомобіля в рамці вважається новий (не дублікатом). Запропонований етап №2 відбору зображень дозволяє точно виявити кадри з новими досліджуваними об’єктами (автомобілями), що продемонстровано на рисунку 4.5 [168]. На рисунку 4.5 лівий кадр А належить до датасету, а правий кадр В є новим (проходить перевірку на унікальність). На лівому кадрі модель YOLO детектувала 3 автомобілі (№ 1-№ 3), а на правому кадрі також детектовано 3 автомобілі (№ 4-№ 6). У межах обмежувальної рамки кожного автомобіля вказано ймовірність його розпізнавання. За угорським алгоритмом перевірено перекривання рамок лівого та правого зображень, у результаті чого отримано значне просторове перекривання рамок: № 1 з № 4, та № 2 з № 5.

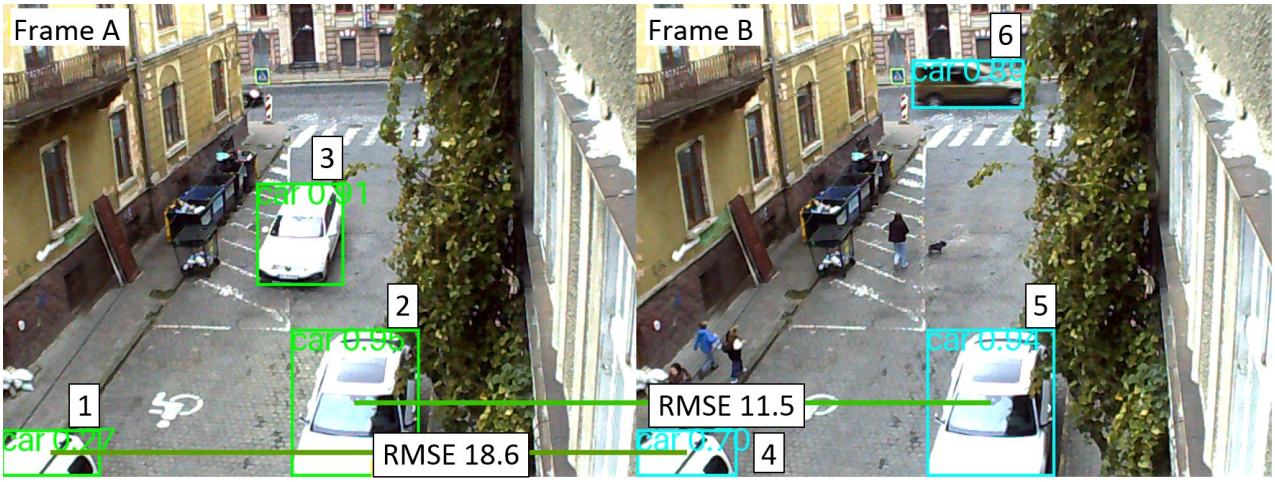


Рисунок 4.5 – Приклад селекції кадрів із використанням детекції та аналізу зображень автомобілів

Для таких пар рамок обчислено їх RMSE, емпірично обрано значення порогу T_{RMSE} = 30. Для двох нерухових автомобілів (рамки № 4 та № 5) отримано значення RMSE 18.6 та 11.5 відповідно (менше за поріг T_{RMSE}), тому такі рамки не вважаються новими. Проте, на правому кадрі є унікальний автомобіль (рамка № 6). Новий кадр проходить перевірку на унікальність на етапі № 2 у тому випадку, якщо відсоток унікальних рамок (з урахуванням їх перекривання та кольору ділянок об'єктів) перевищує поріг T_S (наприклад, T_S = 10%). Розглянутий правий кадр містить 1/3 унікальних рамок, тому такий кадр успішно проходить перевірку і надсилається на наступний крок селекції.

4.2.3 Локальна селекція ділянок автомобілів із використанням перцептивного хешування

На етапі № 3 локальної селекції ділянок автомобілів (обмежених рамками) виконується порівняння детектованих ділянок автомобілів на новому зображенні з ділянками автомобілів для зображень датасету (збережених у базі даних). Така перевірка виконується шляхом порівняння хешів ділянок зображень аналогічно як

для етапу №1, але на етапі №3 порівнюються хеші не всього зображення, а тільки локальних ділянок. Новий кадр вважається унікальним, якщо відсоток його унікальних рамок перевищує поріг T_h. Кадри, які пройшли всі етапи селекції №1-3, вважаються повністю унікальними і додаються до датасету [30].

Перевірка на рівні ділянок автомобілів дозволяє відсіювати повторювані сцени, але водночас не втрачати цінні дані у випадках, коли змінюється склад транспортних засобів на зображенні. Це суттєво підвищує точність відбору унікальних кадрів у порівнянні з підходами, що спираються лише на міжкадрову подібність. Унікальні зображення та координати рамок їх автомобілів зберігаються в датасеті у форматі, який підтримує навчання моделей YOLO. Таким чином формується спеціалізований датасет зображень автомобілів.

Завдяки збереженню кешів у базі даних Redis забезпечується висока швидкодія роботи системи, оскільки порівняння навіть тисячі хешів виконується менш ніж за секунду. Повна перевірка унікальності зображення займає менше ніж 5 секунд, при цьому значна частина цього часу припадає на етап селекції №2 (під час якого виконується детекція зображень об’єктів моделями YOLO).

У розробленій системі датасет формується з використанням потужної моделі нейронної мережі YOLOv8m або YOLOv8l (середнього або великого розміру), оскільки такі моделі забезпечують високу точність виявлення об’єктів навіть у складних сценах. Натомість донавчання на сформованому датасеті виконується для моделі меншого розміру (YOLOv8s або YOLOv8n), що дозволяє зменшити обчислювальні витрати при збереженні достатньої точності детектування [168].

4.3 Методика автоматизованого донавчання нейронної мережі

Традиційний підхід до формування датасетів для детектування об'єктів характеризується високими трудовитратами та дискретністю процесу: після збору

даних слідує етап анотації, а потім – навчання моделі. Такий підхід не дозволяє оперативно реагувати на зміни в умовах експлуатації та накопичувати знання про нові об'єкти в режимі реального часу. Тому розроблено методику донавчання згорткової нейронної мережі YOLO малого розміру (Учень) на основі спеціалізованого датасету зображень, який формується мережею YOLO середнього або великого розміру (Вчитель) за методикою автоматизованого створення набору даних із зображень автомобілів.

Застосування методики обґрунтовується тим, що попередньо навчена модель середнього розміру YOLOv8m потребує багато обчислювальних ресурсів і не враховує специфіку зображень автомобілів у певних дорожніх умовах [168]. Тому спочатку виконується автоматизоване створення спеціалізованого датасету зображень автомобілів за допомогою моделі середнього розміру YOLOv8m, після чого датасет застосовується для донавчання (fine-tuning) моделі найменшого розміру YOLOv8n. Донавчання моделі YOLOv8n на спеціалізованому наборі даних дозволяє суттєво підвищити точність виявлення автомобілів.

Розроблена методика реалізує концепцію безперервного навчання, де процеси зчитування зображень, їх селекції та підготовки до донавчання виконуються автоматизовано. Це дозволяє адаптувати засоби детектування зображень об’єктів до специфічних умов експлуатації та накопичувати репрезентативні датасети, які відображають реальний розподіл об'єктів у цільовому середовищі.

4.4 Програмна реалізація інтелектуальної системи для детектування зображень автомобілів із автоматизованим донавчанням нейронних мереж

Програму «YOLOAutoDatasetCreation» (додаток Е), призначену для автоматизованого створення датасету зображень автомобілів, донавчання моделі нейронної мережі та детектування зображень автомобілів засобами ЗНМ розроблено

на мові Python [176]. Автоматизоване створення датасету зображень автомобілів виконується з використанням моделі ЗНМ середнього розміру YOLOv8m, а донавчання виконується для моделі ЗНМ нано-розміру YOLOv8n. Після донавчання модель YOLOv8n, яка характеризується високою швидкодією, застосовується для практичного детектування зображень транспортних засобів. В програмі використано ряд бібліотек, зокрема opencv, ultralytics, numpy, matplotlib та ін. Реалізацію моделей YOLO виконано з використанням бібліотеки Ultralytics, яка забезпечує широкий вибір параметрів та гіперпараметрів навчання нейромережі.

Автоматизоване формування набору зображень виконується за розробленою методикою (рис. 4.2) шляхом багатоетапної селекції кадрів відеопотоку з використанням хешування зображень. Перцептивне хешування зображень виконано з використанням бібліотеки imagehash. Для реалізації серверу Redis з базою даних використано бібліотеку redis. Сервер Redis розміщено у Docker-контейнері на мікрокомп’ютері Raspberry Pi5. Використання контейнера спростило розгортання сервісу і збереження даних, гарантувало ізоляцію середовища [168]. Після формування набору зображень виконується автоматизоване донавчання ЗНМ малого розміру за розробленою методикою.

Надійність функціонування програми забезпечується розгалуженою системою збереження даних. Модуль RedisHashStorageManager забезпечує персистентне зберігання хешів кадрів у оперативній пам'яті з періодичним скиданням на диск (RDB snapshot), що дозволяє системі зберігати унікальні кадри навіть після перезапуску програми. Логування ключових подій (кількість захоплених, відкинутих та збережених кадрів) дозволяє в реальному часі оцінювати ефективність кожного етапу формування датасету [168].

Основну логіку функціонування програми та оркестрацію всіх компонентів інкапсульовано в класі WebcamNoveltyRunner. Архітектура цього класу побудована

за принципом каскадної фільтрації, що дозволяє мінімізувати обчислювальне навантаження на процесор Raspberry Pi.

Ініціалізація інтелектуальної системи відбувається в конструкторі класу, де задаються ключові параметри роботи: інтервал опитування камери (interval_sec), поріг чутливості хешування (hash_threshold) та коефіцієнт дублювання об'єктів (per_car_duplicate_ratio). Метод run() відповідає за життєвий цикл програми: він встановлює з'єднання з відео пристроєм через інтерфейс cv2.CAP_V4L2, виконує процедуру підготовки відеокамери (пропуск перших кадрів для стабілізації автоекспозиції та балансу білого) та запускає основний цикл обробки.

Розроблена програма описується такою діаграмою класів (рис. 4.6).

Найважливішим класом програми є WebcamNoveltyRunner (src/webcam/runner.py), який реалізує конвеєр обробки відеопотоку з трьома послідовними етапами селекції кадрів.

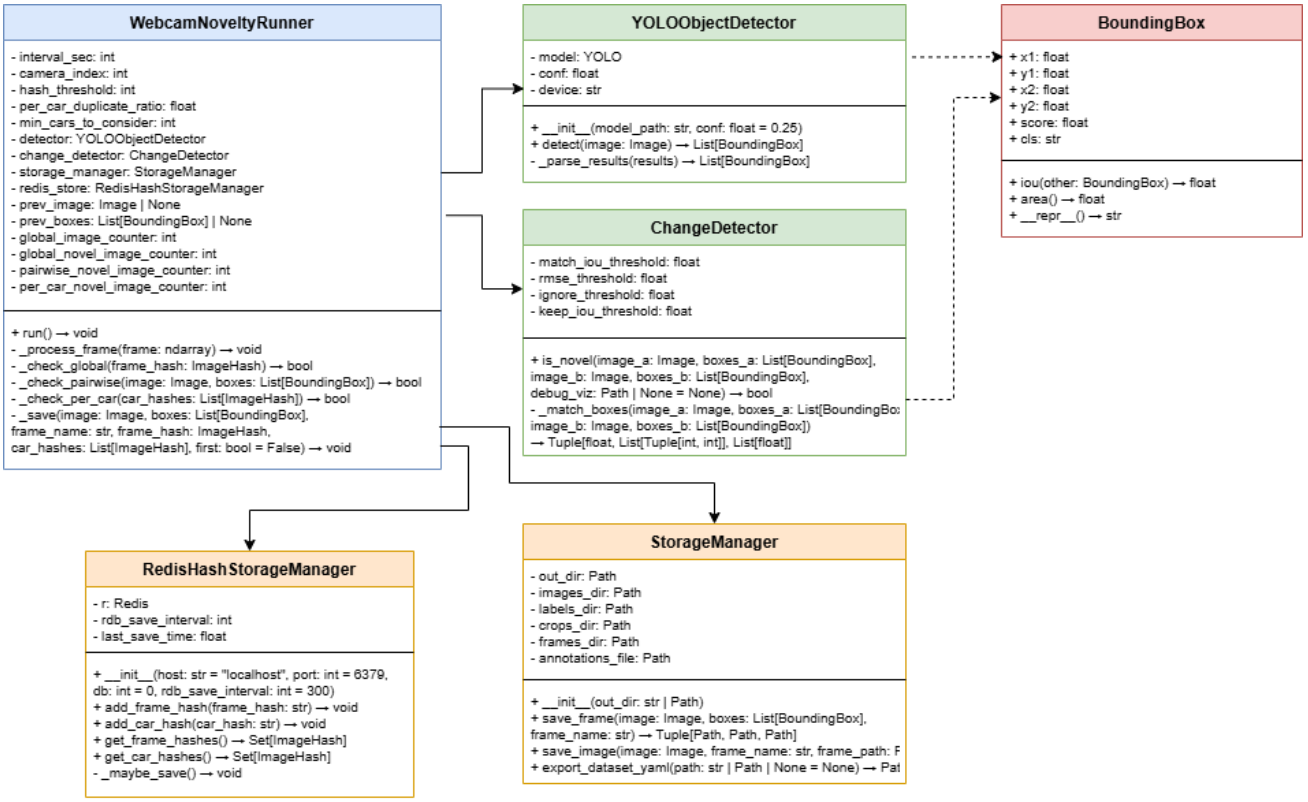


Рисунок 4.9 – Діаграма класів програми "YOLOAutoDatasetCreation"

У класі WebcamNoveltyRunner реалізовано модуль захоплення та обробки відеопотоку з використанням бібліотеки OpenCV. Система використовує Video4Linux2 (V4L2) бекенд для ефективного доступу до апаратного відеозахоплення на ARM64 архітектурі Raspberry Pi 5. Захоплення кадрів відбувається з конфігурованим інтервалом interval_sec (за замовчуванням 5 секунд, оскільки за цей час відбуваються суттєві зміни в конфігурації транспортних засобів у полі зору камери). Така частота зчитування кадрів забезпечує їх достатню різноманітність при незначному обчислювальному навантаженні.

Клас YOLOObjectDetercor призначений для детектування зображень об’єктів ЗНМ YOLO. Клас BoundingBox використовується для роботи з рамками детектованих об’єктів. Клас ChangeDetector застосовується для зміни процесу детектування. Клас RedisHashStorageManager застосовується для збереження хешів у базі даних Redis. Клас StorageManager призначений для збереження зображень та їх параметрів.

4.4.1 Програмна реалізація автоматизованого формування датасету

Процес селекції кадрів для формування датасету реалізовано в методі обробки кадру _process_frame(), в якому, відповідно до розробленої методики, селекція розділена на три етапи зі зростаючою обчислювальною складністю:

1. Глобальна селекція кадрів з використанням перцептивного хешування. На даному етапі для всього кадру обчислюється перцептивний хеш, який порівнюється зі збереженими хешами бази Redis методом _check_global(). Якщо значення хешів виявляються подібними, то зображення відкидається.
2. Селекція кадрів із використанням детектування автомобілів моделлю YOLOv8m. Порівняння кадрів виконується з використанням угорського алгоритму.

Зображення вважаються унікальними, якщо вони містять нові автомобілі, порівняно з кадрами датасету [168].

3. Локальна селекція ділянок автомобілів із використанням перцептивного хешування. Кадр вважається дублікатом у тому випадку, якщо частка автомобілів-дублікатів перевищує заданий поріг (за замовчуванням 0.8).

Процес зчитування та селекції кадрів описується такими параметрами:

1. global_image_counter – загальна кількість захоплених кадрів.
2. global_novel_image_counter – кадри, що пройшли глобальну селекцію хешів.
3. pairwise_novel_image_counter – кадри, що пройшли перевірку з використанням детектування автомобілів.
4. per_car_novel_image_counter — кадри, що пройшли локальну селекцію.

Такі параметри дозволяють оцінити ефективність кожного етапу селекції та налаштувати порогові значення для оптимізації якості датасету.

Кадри, які успішно пройшли всі 3 етапи селекції, додаються в датасет разом із координатами автомобілів. Завдяки такій селекції кадрів формується датасет з унікальними зображеннями автомобілів, що відрізняються умовами освітлення, кутами огляду, марками транспортних засобів, погодними умовами та часом доби [30]. Така багатоетапна селекція дозволяє значно зменшити обсяг збережених даних при забезпеченні високого рівня інформативності датасету. За результатами експериментальних досліджень отримано, що з кожних 100 захоплених кадрів у фінальний датасет потрапляє 5-15 кадрів (залежно від динаміки сцени).

4.4.2 Зберігання перцептивних хешів

Для зберігання перцептивних хешів в оперативній пам’яті використовується база даних Redis, що забезпечує високу швидкість доступу до даних (латентність на рівні мікросекунд) та можливість атомарних операцій над множинами.

Хеші зберігаються у двох окремих множинах (Redis Sets):

1. frame_hashes – множина 64-бітних перцептивних хешів повних кадрів.
2. car_hashes – множина 64-бітних перцептивних хешів для ділянок автомобілів.

Використання структури даних Set забезпечує:

1. Автоматичну дедуплікацію (множина не містить повторень).
2. O(1) складність додавання та перевірки наявності елемента.
3. Підтримку операцій над множинами (об'єднання, перетин, різниця).

Redis налаштовано на створення RDB (англ. Redis Database) снапшотів за двома критеріями:

1. Кожні 60 секунд за наявності хоча б однієї зміни.
2. При коректному завершенні роботи системи.

Конфігурація rdb_save_interval=300 у RedisHashStorageManager забезпечує додаткову гарантію збереження даних кожні 5 хвилин під час активної роботи системи. До переваг Redis при збереженні перцептивних хешів належить компактність зберігання, оскільки 64-бітні хеші у hex-форматі займають 16 байт пам'яті. При типовому обсязі датасету в 10000 кадрів загальний обсяг даних становить ~160 КБ для frame_hashes. Клас RedisHashStorageManager (src/storage/redis_manager.py) інкапсулює всю логіку взаємодії з Redis. Методи get_*_hashes() перетворюють hex-представлення хешів назад в об'єкти ImageHash для обчислення відстані Хеммінга за допомогою бібліотеки imagehash.

4.4.3 Апаратне забезпечення та програмне середовище інтелектуальної системи

Розроблена інтелектуальна система функціонує на апаратній платформі одноплатного мікрокомп’ютера Raspberry Pi 5 (8 ГБ оперативної пам'яті), який виконує роль локального обчислювального вузла. Програмне забезпечення вузла

виконує зчитування кадрів з веб-камери, початкову обробку кадрів, перевірку їх на унікальність, збереження унікальних кадрів та їх хешів у датасет. Вибір Raspberry Pi 5 обумовлено комбінацією ряду факторів: його енергоефективністю, низькою вартістю, компактністю та можливістю запускати повноцінне Linux-середовище з підтримкою Docker. На практиці Raspberry Pi використовувався як самостійний вузол, розташований безпосередньо біля відеокамери. Камера під'єднувалася до Raspberry Pi по шині USB і налаштовувалася на зчитування кадрів певної роздільної здатності (наприклад, 640 × 480 пікселів).

Модель веб-камери обрано з урахуванням стабільності потокового захоплення, фіксації фокусу та можливості відключити автоматичні режими (автоекспозиція, автобаланс білого). Отримання відеопотоку без використання автоматичних режимів відеокамери значно зменшує флуктуації яскравості та кольору кадрів, які негативно впливають на точність селекції з використанням перцептивних хешів. Захоплення відеопотоку здійснювалося за допомогою бюджетної веб камери споживчого класу виробника Canyon.

Мікрокомп'ютер Raspberry Pi 5 забезпечує значну обчислювальну потужність, достатню для роботи з моделями YOLO. Застосування мікрокомп'ютера Rapsberry Pi5 також є вигідним економічно з урахуванням його низької вартості та енергоспоживання. В якості носія даних використовувалася стандартна SD-карта, швидкості якої виявилося достатньо для буферизації та збереження відібраних кадрів.

Програмне середовище розгорнуто на базі операційної системи Debian GNU/Linux 12 (bookworm). Для виконання коду використано інтерпретатор Python версії 3.13.7, що забезпечило підтримку найновіших можливостей асинхронного виконання та оптимізації пам'яті.

4.5 Веб-інтерфейс для моніторингу та керування програмою

Для забезпечення зручного контролю над програмою «YOLOAutoDatasetCreatio» розроблено веб-орієнтований інтерфейс на основі архітектури "одно-сторінкового застосунку" (англ. Single Page Application, SPA). Веб-застосунок реалізовано як розподілену систему з трьома основними компонентами:

1. Фронтенд-шар (англ. Presentation Layer)

Реалізований як SPA на базі екосистеми React 18.3 з використанням TypeScript для статичної типізації та Vite як збирача модулів (англ. bundler). Вибір технологічного стеку обумовлений наступними факторами:

- React: Декларативна парадигма опису інтерфейсу з віртуальним DOM дозволяє ефективно оновлювати лише змінені компоненти UI при надходженні нових даних від backend. Механізм hooks (useEffect, useState) забезпечує зручне управління станом та побічними ефектами без використання класових компонентів.
- TypeScript: Статична типізація важлива для підтримки типобезпеки (typesafety) при роботі з API. Інтерфейси TypeScript (DatasetStatus, BoundingBox, Match тощо) гарантують відповідність структур даних між frontend та backend на етапі компіляції, що попереджує цілий клас runtime помилок.
- Tailwind CSS: Utility-first CSS фреймворк дозволяє швидко створювати responsive інтерфейси без написання власних CSS класів. Підхід atomic CSS забезпечує мінімальний розмір фінального bundle за рахунок автоматичного tree-shaking невикористаних стилів.
- Vite: Сучасний збирач модулів використовує нативні ES modules та esbuild для досягнення над швидкого холодного старту (англ. cold start) та гарячої заміни модулів (англ. Hot Module Replacement, HMR) під час розробки. У production режимі Vite генерує оптимізовані bundle з code-splitting та lazy loading.

2. Бекенд-шар (Application Layer)

Реалізований на базі асинхронного веб-фреймворку FastAPI, що базується на стандарті ASGI (англ. Asynchronous Server Gateway Interface). Архітектурні переваги FastAPI [177]:

- Асинхронність – нативна підтримка async/await дозволяє обробляти багато паралельних запитів на одному потоці без блокування. Це важливо для endpoints, що виконують тривалі операції (захоплення зображень з камери, YOLO inference).
- Автоматична валідація – інтеграція з Pydantic забезпечує автоматичну валідацію вхідних даних та серіалізацію відповідей згідно з оголошеними моделями (DatasetConfig, HungarianResult тощо). Будь-які відхилення від схеми автоматично генерують HTTP 422 Unprocessable Entity з детальним описом помилок.
- Автодокументація – FastAPI автоматично генерує OpenAPI (Swagger) специфікацію та інтерактивну документацію API за адресою /docs.
- Продуктивність – за benchmark результатами, FastAPI демонструє продуктивність на рівні Node.js та Go завдяки використанню Starlette та Pydantic, написаних на Cython.

3. Шар персистентності (Data Layer)

Представлений Redis in-memory базою даних для зберігання хешів та локальною файловою системою для зберігання зображень, crop-файлів (фрагментів зображень), YOLO-анотацій та метаданих. Компоненти програми взаємодіють за RESTful архітектурою через HTTP/JSON API (рис. 4.10).

Frontend виконує періодичні запити до backend кожні 2 секунди для отримання актуального статусу системи через endpoint GET /api/dataset/status. Такий підхід забезпечує квазі-реального часу оновлення UI без необхідності впровадження WebSocket з'єднань. Для забезпечення ізольованості компонентів програми, відтворюваності середовища та спрощення розгортання інтелектуальна система реалізована як набір Docker контейнерів [178, 179] (додаток Ж).

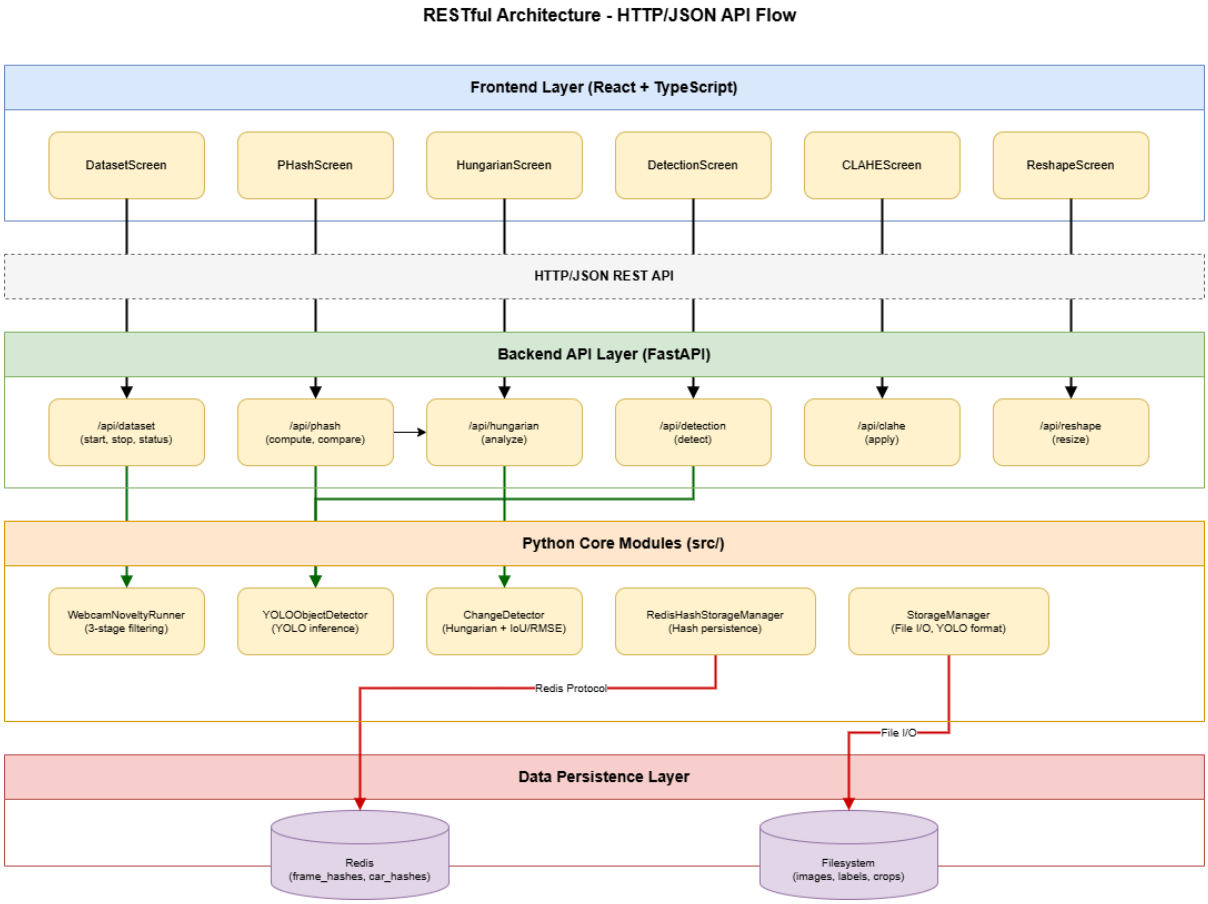


Рисунок 4.10 – RESTful архітектура програми

Веб-інтерфейс організовано як багатосторінковий застосунок з табовою (tabs) навігацією. Кожна вкладка реалізує окремий функціональний модуль системи (рис. 4.11). Компонент Navigation.tsx реалізує табову навігацію між 5 екранами (вікнами, вкладками):

1. Збір набору даних (англ. Dataset Screen).
2. CLAHE покращення (англ. CLAHE Screen) та зміна розміру зображення (англ. Reshape Screen).
3. Перцептивне хешування (англ. PHash Screen).
4. Угорський алгоритм (англ. Hungarian Screen).
5. Порівняння YOLO – детектування об’єктів нейронною мережею YOLO та оцінка результатів детектування (англ. Detection Screen).

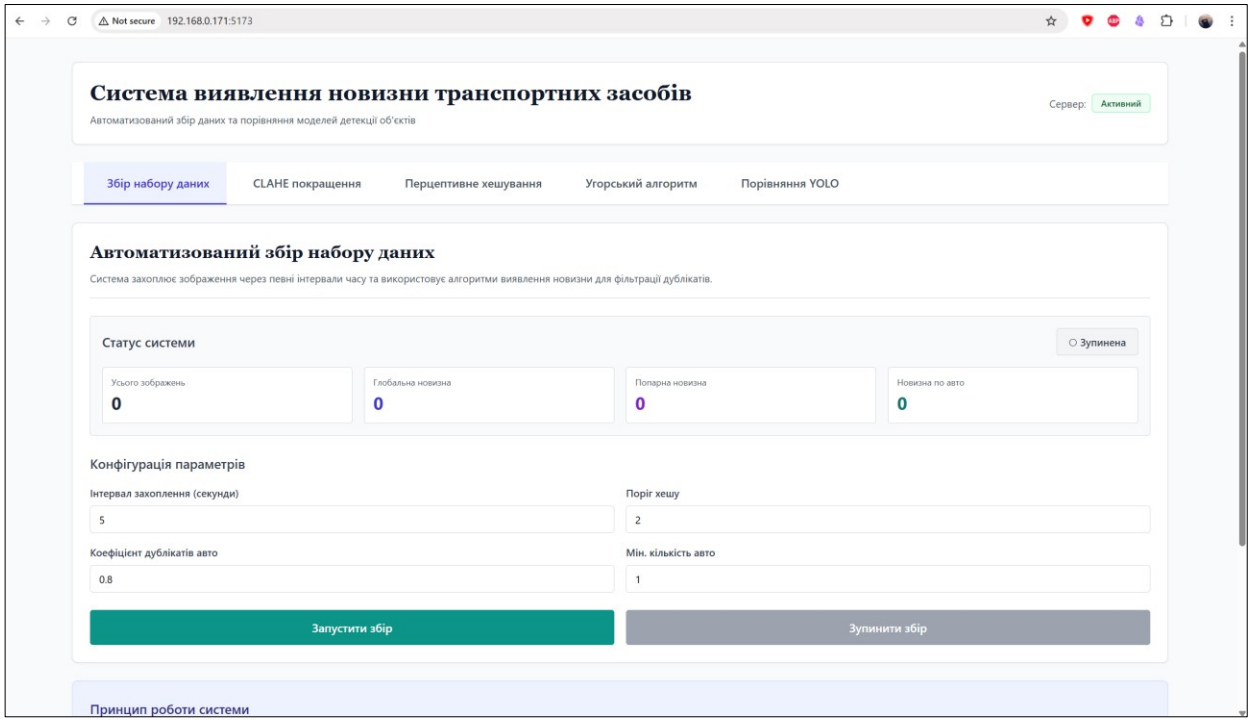


Рисунок 4.11 – Користувацький інтерфейс розробленої програми, екран «Автоматизований збір даних»

Кожний екран інтерфейсу реалізований як окремий React компонент з власним станом та логікою взаємодії з backend API. Розглянемо детальніше призначення екранів інтерфейсу.

4.5.1 Екран автоматизованого збору даних

Центральний екран автоматизованого збору набору даних «Dataset Screen» реалізує комплексний інтерфейс управління процесом автоматизованого створення датасету. Інтерфейс організовано з використанням чотирьох основних функціональних блоків, що забезпечують повний цикл контролю над системою:

1. Блок моніторингу статусу системи надає користувачу інформацію про поточний стан роботи у реальному часі. Головним елементом є індикатор стану, що відображає два можливі режими роботи системи: "Активна" або "Зупинена". Стан

візуалізується через кольорове кодування – зелений колір сигналізує про активну роботу системи, тоді як сірий вказує на призупинений стан. Під індикатором розташовано чотири числові лічильники, що відображають ключові метрики процесу селекції кадрів. Перший лічильник показує загальну кількість захоплених зображень з веб-камери. Другий відображає кількість кадрів, які успішно пройшли перевірку глобального перцептивного хешу. Третій лічильник демонструє кількість зображень, що пройшли етап попарного порівняння з використанням детектування зображень об’єктів та угорського алгоритму. Четвертий лічильник показує фінальну кількість кадрів, які пройшли локальну селекцію хешів автомобілів та були збережені у датасет. Усі метрики автоматично оновлюються кожні дві секунди завдяки використанню React hook useEffect, що створює setInterval з відповідним інтервалом та виконує асинхронний запит до API ендпоінту (додаток К) для отримання актуальних даних.

2. Блок конфігурації параметрів збору дозволяє налаштувати чотири важливі параметри роботи системи. Параметр інтервалу захоплення визначає часову затримку між послідовними кадрами у діапазоні від одної до шістдесяти секунд. Поріг хешу задає максимально допустиму відстань Хеммінга між перцептивними хешами для визнання зображень подібними (від нуля до десяти біт). Коефіцієнт дублікатів автомобілів встановлює частку детектованих транспортних засобів, які повинні бути розпізнані як дублікати для відхилення кадру, з діапазоном від 0.0 до 1.0. Мінімальна кількість автомобілів для розгляду визначає порогову кількість детектованих об'єктів, необхідну для застосування правила коефіцієнту дублікатів. Важливою особливістю інтерфейсу є автоматична деактивація полів вводу параметрів під час активної роботи системи – всі поля отримують атрибут disabled, що унеможливлює зміну конфігурації в процесі збору даних та гарантує консистентність експерименту.

3. Блок управління процесом збору представлений двома функціональними кнопками з взаємовиключною логікою активації. Кнопка "Запустити збір" ініціює асинхронний POST запит до API ендпоінту /api/dataset/start, передаючи поточну конфігурацію параметрів у JSON форматі. При успішному запуску на backend створюється екземпляр класу WebcamNoveltyRunner з переданими параметрами, який виконується у фоновій asyncio задачі для забезпечення неблокуючої роботи веб-сервера. Кнопка "Зупинити збір" виконує POST запит до /api/dataset/stop, що ініціює graceful shutdown процесу збору. Backend скасовує фонову задачу через виклик task.cancel(), чекає на коректне завершення через await задачі у блоці try-except для обробки CancelledError, та зберігає фінальний Redis RDB snapshot для персистентності накопичених хешів. Обидві кнопки динамічно змінюють свій стан доступності залежно від значення прапорця is_running у глобальному статусі системи – кнопка запуску активна лише коли система зупинена, а кнопка зупинки доступна виключно під час активної роботи.

4. Інформаційна панель розташована у нижній частині екрану та виконує освітню функцію, надаючи детальний опис принципу роботи трьох етапів селекції кадрів. Панель описує послідовність операцій, що виконуються над захопленим кадром, пояснюючи логіку прийняття рішень на кожному етапі та обґрунтовуючи необхідність багатоетапної обробки для досягнення оптимального балансу між різноманітністю датасету та ефективністю використання дискового простору.

4.5.2 Екрани попередньої оброки зображень

Результати попередньої обробки зображень відображаються на екранах «CLAHE Screen» та «Reshape Screen». Така обробка застосовується у процесі підготовки датасетів та для дослідження впливу трансформацій зображень на точність детектування.

На екрані «CLAHE Screen» виконується демонстрація впливу збільшення контрасту зображення на точність детектування. Реалізовано, зокрема, збільшення контрасту методом адаптивної еквалізації гістограми (методом CLAHE), робота якого регулюється параметрами clipLimit та tileGridSize. Параметр clipLimit означає максимальну відносну кількість пікселів у кожному інтервалі (біні) гістограми і застосовується для попередження надмірного посилення шуму в однорідних регіонах зображення. Параметр tile_grid_size визначає розмір прямокутних ділянок (tiles) зображення, в межах яких підвищується локальних контраст. Після завантаження зображення та налаштування параметрів виконується запит до /api/clahe/apply, який застосовує scikit-image реалізацію CLAHE алгоритму. Результати відображаються у side-by-side компонуванні з оригінальним зображенням ліворуч та обробленим праворуч, що дозволяє безпосередньо оцінити вплив обробки (рис. 4.12).

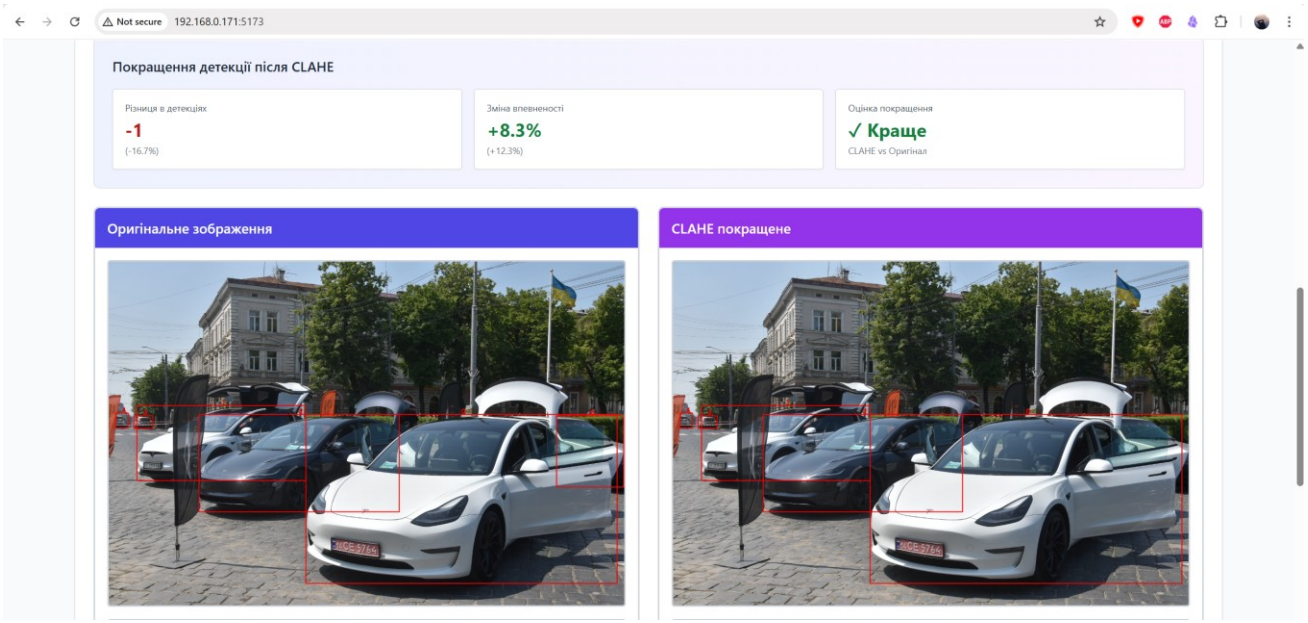


Рисунок 4.12 – Приклад екрану «CLAHE Screen» із результатами детектування автомобілів

Для кожного зображення також обчислюється його гістограма. У розглянутому прикладі на оригінальному зображення детектується зайвий об’єкт, а на зображенні після обробки виявляються тільки автомобілі, які при цьому розпізнаються з вищою впевненістю. В програмі також реалізується попередня обробка зображень методами глобального вирівнювання (еквалізації) гістограми та запропонованим методом вирівнювання та центрування гістограми.

Екран «Reshape Screen» реалізує функціональність пакетної (batch) обробки зображень для зміни їх розмірів з підтримкою двох режимів масштабування: 1) зі збереженням пропорцій (aspect ratio); 2) без збереження пропорцій (force resize). Backend обробляє кожне зображення паралельно через asyncio.gather для максимізації throughput, зберігає результати у тимчасовий директорій та повертає ZIP архів з усіма обробленими зображеннями через StreamingResponse з MIME типом application/zip для автоматичного завантаження у браузері користувача.

4.5.3 Екран перцептивного хешування

Призначення екрану перцептивного хешування «PHash Screen» полягає в інтерактивній демонстрації принципів роботи алгоритму pHash та візуалізації процесу обчислення хеш-дескрипторів зображень. Інтерфейс підтримує режими роботи №1-3, кожен з яких фокусується на різних аспектах перцептивного хешування.

У режимі №1 завантаження одиничного зображення користувач взаємодіє з drag-and-drop областю, реалізованою через події ondragover, ondrop та onclick для альтернативного вибору файлу через системний діалог. Після завантаження файлу виконується асинхронний POST запит до ендпоінту /api/phash/compute, який приймає зображення у форматі multipart/form-data, виконує конвертацію у RGB колірний простір через Pillow, обчислює 64-бітний перцептивний хеш засобами

бібліотеки imagehash, та повертає результат у трьох представленнях. Hex представлення відображає хеш як шістнадцяткове число, зручне для зберігання у Redis. Бінарне представлення візуалізує 64-бітну послідовність з автоматичним групуванням по вісім біт для покращення читабельності та демонстрації внутрішньої структури хешу. Зображення повертається у base64-encoded форматі для відображення безпосередньо у браузері без додаткових HTTP запитів.

Режим №2 порівняння двох зображень розширює функціональність, дозволяючи користувачу завантажити дві фотографії. Після завантаження обох файлів виконується запит до /api/phash/compare, який обчислює перцептивні хеші для обох зображень та розраховує відстань Хеммінга між ними. Інтерфейс відображає обчислені хеші для обох зображень, числове значення відстані Хеммінга, та бінарний індикатор схожості з фіксованим порогом $\tau = 5$ біт. Якщо відстань Хеммінга не перевищує п'яти біт, зображення класифікуються як "Similar" та підсвічуються зеленим кольором; у протилежному випадку відображається мітка "Different" з червоним кольоровим кодуванням.

Розглянемо приклад порівняння перцептивних хешів двох зображень (рис. 4.13). Незважаючи на те, що зображення схожі, є відмінності у кількості автомобілів, їх кольорах, тощо. Такі зміни також відображаються на зображеннях хешів. В програмі можна інтерактивно змінити поріг відстані Хеммінга; якщо відстань Хеммінга не перевищує поріг, то зображення вважаються подібними.

На рисунку 4.14 показано візуалізацію різниці двох хешів: хеш зображення № 1 позначено червоним кольором, а хеш зображення № 2 – синім кольором. У розглянутому випадку відстань Хемінга (4.2) перевищує поріг (4.3), тому зображення вважаються унікальними (кадри проходить даний етап селекції). Тобто, виконується класифікація даних хешів як унікальних або неунікальних, а відповідно і класифікація зображень. Режим №3 захоплення з камери демонструє можливість обчислення перцептивних хешів у реальному часі для відеопотоку.

Рисунок 4.13 – Приклад вікна програми для порівняння перцептивних хешів двох зображень

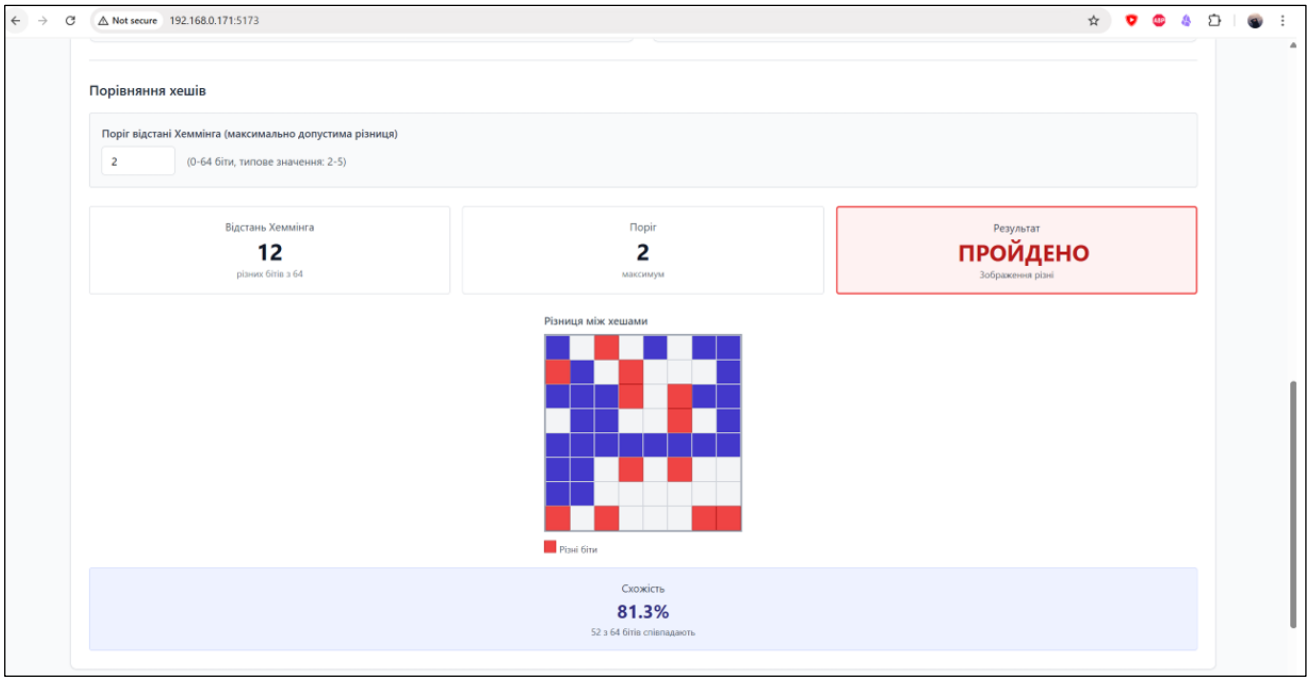


Рисунок 4.14 – Приклад вікна програми для різниці перцептивних хешів двох зображень

При натисканні кнопки "Capture and Hash" виконується запит до /api/phash/capture-and-hash, який на стороні backend ініціалізує cv2.VideoCapture з V4L2 backend, виконує пропускання 5 перших кадрів для стабілізації автоматичних налаштувань камери, захоплює кадр з роздільністю 640×480 пікселів, конвертує його у PIL Image через cv2.cvtColor з COLOR_BGR2RGB перетворенням, обчислює pHash та повертає результат разом з base64-encoded зображенням для візуалізації. Цей режим особливо корисний для розуміння того, як система обробляє реальні кадри з камери у процесі автоматизованого збору датасету.

4.5.4 Екран угорського алгоритму

На екрані угорського алгоритму «Hungarian Screen» виконується детальна візуалізація процесу оптимального зіставлення детектованих рамок автомобілів між двома послідовними кадрами, що є ключовим компонентом другого етапу селекції кадрів. Для порівняння зображень використовується три функціональні блоки.

Після вибору файлів відображаються прев’ю зображень з автоматичним масштабуванням для забезпечення однакових розмірів у інтерфейсі, що полегшує візуальне порівняння.

Команда аналізу кадрів викликає POST запит до /api/hungarian/analyze з обома зображеннями у multipart/form-data форматі. Backend послідовно виконує декілька операцій: спочатку конвертує обидва зображення у RGB формат через Pillow, потім виконує YOLO детектування для кожного зображення окремо з виділенням певних класів (car, truck та bus), будує матрицю вартості розміром n × m елементів, де кожен елемент визначається комбінацією метрик IoU та RMSE. Після цього застосовується Угорський алгоритм через scipy.optimize.linear_sum_assignment для знаходження оптимального зіставлення, фільтрує отримані результати з порогами IoU ≥ 0.5 та RMSE ≤ 30.0. Результати селекції відображаються у вікні.

4.5.5 Екран детектування об’єктів нейронною мережею YOLO

Екран детектування зображень об’єктів ЗНМ YOLO «Detection Screen» реалізує інтерактивну демонстрацію роботи моделі YOLOv8m з можливістю обробки як завантажених файлів, так і зображень з веб-камери у реальному часі. Інтерфейс організовано у вигляді двох основних блоків з чітким розділенням функціональності вводу та виводу.

Блок завантаження зображення надає користувачу два альтернативних методи отримання вхідних зображень. File upload інтерфейс реалізовано через стандартний HTML input елемент з типом file та accept атрибутом, обмеженим форматами image/png, image/jpeg та image/jpg для забезпечення коректної обробки. Drag-and-drop функціональність додає зручності при виборі файлу. Опція "Capture from Camera" виконує асинхронний запит до backend для захоплення поточного кадру з веб-камери через cv2.VideoCapture, що застосовується при детектуванні зображень у режимі реального часу.

Після отримання зображення виконується POST запит до /api/detection/detect, що передає файл у multipart/form-data форматі. Backend завантажує глобальний екземпляр YOLO моделі через lazy loading патерн – модель ініціалізується лише при першому запиті для мінімізації startup latency, після чого зберігається у глобальній змінній для повторного використання. Модель виконує виявлення об’єктів на вхідному зображенні, фільтрує об’єкти заданих класів (транспортних засобів), повертає координати рамок об’єктів у форматі (x1, y1, x2, y2) – як координати лівої верхньої та правої нижньої вершин прямокутника. Функціями бібліотеки OpenCV виконується візуалізація детектованих прямокутників об’єктів на зображеннях з вказанням класів об’єктів та впевненості їх розпізнавання. Зображення ділянок детектованих об'єктів (crop-зображення), масштабованих до фіксованої ширини, відображаються у вигляді галереї мініатюр. Crop-зображення кодуються у base64

формат для вбудовування безпосередньо у HTML через data URI scheme. При наведенні курсору на crop-зображення спрацьовує hover ефект зі збільшенням масштабу через CSS transform, що дозволяє детальніше роздивитись об'єкт.

4.6 Забезпечення кібербезпеки для розробленої системи

Системи автоматичного детектування транспортних засобів обробляють потенційно чутливі дані спостереження, які містять інформацію про місцезнаходження камер, часові патерни руху транспорту та візуальні характеристики автомобілів. В контексті наукового дослідження та можливого практичного застосування важливо забезпечити захист цих даних від несанкціонованого доступу, перехоплення та модифікації. Це обґрунтовує необхідність впровадження комплексної системи кібербезпеки, яка охоплює всі рівні передачі та зберігання даних.

Розроблена архітектура безпеки базується на принципі ешелонованого захисту (англ. defense-in-depth), який передбачає використання множинних незалежних рівнів захисту. Такий підхід гарантує, що навіть у разі компрометації одного рівня безпеки, інші рівні продовжують захищати систему від атак. В нашій реалізації використовуються три основні рівні захисту: шифрування транспортного рівня, аутентифікація користувачів та криптографічне маркування даних.

Перший рівень захисту реалізовано через протокол Transport Layer Security (TLS) версії 1.2, який забезпечує шифрування всіх даних, що передаються між веб-інтерфейсом системи та серверним застосунком [180]. TLS використовує гібридну криптографічну схему, що поєднує асиметричне шифрування для безпечного обміну ключами та симетричне шифрування для швидкої передачі даних. Для шифрування даних використовується алгоритм AES-256 в режимі GCM (англ. Galois/Counter Mode), який забезпечує як конфіденційність, так і автентичність повідомлень.

Другий рівень захисту забезпечує аутентифікацію користувачів та контроль доступу до функцій системи. Для цього використовується технологія JSON Web Token (JWT), яка дозволяє створювати токени доступу без необхідності зберігання стану сесій на сервері. Кожен JWT токен містить інформацію про користувача (ім'я, роль, доступні камери) та підписується за допомогою алгоритму HMAC-SHA256, що унеможливлює підробку токенів без знання секретного ключа [181].

Третій рівень захисту стосується безпеки самих даних детекції і реалізований через систему ролей та прав доступу. Система підтримує два основні типи користувачів: адміністратори з повним доступом до всіх камер та функцій управління, та дослідники з обмеженим доступом лише для перегляду даних з певних камер.

4.7 Автоматизоване формування набору зображень

З метою автоматизованого формування датасету отримано серію зображень за допомогою відеокамери, розташованої на висоті третього поверху будинку під кутом 45° до проїжджої частини. Відеокамери приєднувалася до пристрою збору та обробки даних (мікрокомп’ютер Raspberry Pi 5). Поле зору камери охоплювало складну урбаністичну сцену, яка включала:

1. Вулицю з трафіком середньої інтенсивності безпосередньо перед камерою.
2. Зону паркування (статичні об'єкти).
3. Пішохідну прохідну частину.
4. Фоновий огляд вулиці з високою інтенсивністю руху.

Типова тривалість експериментальної сесії перевищувала 2 години. Важливою особливістю протоколу був вибір часу зйомки: сеанси розпочиналися за годину до сутінок і тривали до настання повної темряви. Такий підхід дозволив протестувати систему в умовах динамічного світлового переходу. Це створило екстремальні умови

для алгоритмів комп'ютерного зору, оскільки камера фіксувала сцени при різних типах освітлення:

- Природне денне світло (хмарність, прямі сонячні промені, дощ).
- Змішане освітлення (сутінки).
- Штучне нічне освітлення (світло фар, вуличні ліхтарі).

У рамках експериментальної сесії зчитано 993 зображення з інтервалом у 5 секунд. Селекція кадрів виконувалася на всіх трьох етапах обробки [168] (табл. 4.1).

Таблиця 4.1 – Результати багатоетапної селекції кадрів

Етап перевірки	Кількість зображень після етапу	Частка від початкової вибірки	Опис впливу
Загальний обсяг даних (усі зчитані кадри)	993	100 %	Усі зображення, отримані камерою
1. Глобальна селекція з використанням pHash	150	15,1 %	Відкинуті майже ідентичні сцени на рівні зображення
2. Селекція кадрів з урахуванням IoU та RMSE	131	13,2 %	Вилучені кадри, де склад автомобілів не змінювався суттєво між сусідніми сценами
3. Локальна селекція кадрів	121	12,2 %	Відібрані лише ті кадри, де з’явилися нові унікальні транспортні засоби

Точність функціонування системи визначається її здатністю відкидати кадри-дублікати та зберігати лише ті кадри, які містять нові об’єкти або унікальні комбінації транспортних засобів. Саме здатність системи коректно фільтрувати кадри-дублікати в умовах, коли сенсор камери генерує значний шум через нестачу світла в сутінках, стала головним критерієм успішності експерименту.

Як видно з таблиці 4.1, найсуттєвіше скорочення відбулося вже на першому етапі глобальної селекції, на якому відсіяло понад 84 % кадрів як надмірно схожих. Подальші два етапи зменшили набір даних до 121 унікального кадру (~12 % від початкових даних) [30]. Таким чином, система ефективно зменшує надлишковість у даних, зберігаючи при цьому найбільш інформативні приклади для формування унікального датасету (рис. 4.15). Сформований набір кадрів характеризується підвищеною різноманітністю та придатністю для подальшого використання у задачах донавчання нейронних мереж.

Рисунок 4.15 – Приклад 10 кадрів після селекції

Отримані результати демонструють, що запропонована система є придатною для автоматизованого створення унікальних датасетів.

4.8 Донавчання моделі нейронної мережі YOLO

Спеціалізований датасет автомобілів, створений розробленої програмою в автоматизованому режимі з використанням моделі YOLOv8m середнього розміру, використано для донавчання моделі YOLOv8n нано-розміру, яка характеризується високою швидкодією та низькими витратами обчислювальних ресурсів. Такий підхід поєднує точність моделі YOLOv8m при формуванні датасету із швидкодією моделі YOLOv8n після донавчання, що особливо важливо для систем реального часу. У результаті автоматизованої обробки кадрів, зчитаних на протязі 8-ми сесій, відібрано 1063 унікальні зображення, з яких 744 використано для навчання нейромережі, 212 – для валідації та 107 – для тестування. Після донавчання модель YOLOv8n продемонструвала суттєве покращення якості детекції автомобілів порівняно з моделлю без донавчання (табл. 4.2).

Таблиця 4.2 – Результати детектування зображень автомобілів [176]

Показники точності детектування	Модель YOLOv8n без донавчання	Донавчена модель YOLOv8n
mAP50	0.6864	0.9292
mAP50-95	0.5290	0.8270
Precision	0.7406	0.8887
Recall	0.5869	0.8640

Зростання показника mAP50 до 0.93 після донавчання підтверджує високу точність локалізації автомобілів. Підвищення mAP50-95 (mean Average Precision для IoU від 0.5 до 0.95) з 0.53 до 0.83 свідчить про стабільність моделі при роботі з об’єктами різних розмірів та ракурсів. Зростання показника precision (точність) з 0.74 до 0.89 свідчить про значне зменшення кількості хибно позитивних

спрацьовувань. Підвищення значень recall (повнота) з 0.59 до 0.86 означає, що донавчена модель пропускає набагато менше об’єктів [176].

Таким чином, автоматизоване створення датасету з використанням попередньо навченої моделі YOLOv8m та подальше донавчання менших моделей (YOLOv8n) дозволяє підвищити точність детектування об’єктів. Такий підхід може використовуватися для створення ефективних систем моніторингу транспортних засобів або аналізу дорожньої ситуації.

4.9 Прикладне застосування нейронних мереж для детектування зображень автомобілів

Тестування донавченої моделі YOLOv8n проведено на зображеннях тестової вибірки (рис. 4.16) [176]. Експериментальні дослідження показали, що модель YOLOv8n забезпечує точне детектування автомобілів для різних умов освітлення та для різноманітної дорожньої обстановки.

Рисунок 4.16 – Приклади детектування зображень автомобілів за допомогою донавченої моделі YOLOv8n

Практична цінність отриманої моделі YOLOv8n полягає не тільки у високій точності детектування, але й в здатності працювати в режимі реального часу на периферійних пристроях (англ. Edge Computing), зокрема, на мікрокомп'ютерах Raspberry Pi. Такий підхід відкриває можливості для широкого впровадження розробленої інтелектуальної системи, зокрема, у наступних сферах:

- Автоматизований аналіз трафіку та стану дорожньої обстановки на певних ділянках вулиць.
- Моніторинг стану паркувальних майданчиків – виявлення зайнятих та вільних місць у реальному часі.
- Системи безпеки, зокрема, фіксація транспортних засобів на закритих територіях з високою точністю розпізнавання, адаптованою до конкретних умов спостереження.

У більшості випадків для отримання зображень об'єктів використовуються стаціонарні відеокамери, проте, застосовується також розміщення відеокамер на рухомих платформах, зокрема, на безпілотних літальних апаратах (БПЛА) [182, 183]. У випадку отримання відеопотоку з БПЛА розроблена інтелектуальна система детектування зображень об'єктів може застосовуватися для дистанційного зондування певних територій на предмет присутності транспортних засобів. Аналіз аерофотознімків, отриманих з великої висоти, дозволяє здійснювати моніторинг транспортної інфраструктури, забезпечуючи широкий огляд сцени.

Особливістю зображень автомобілів, отриманих з БПЛА, як правило, є малий розмір об'єктів, специфічний ракурс фотографій та значне затінення від автомобілів та інших об'єктів (за рахунок бічного освітлення). Параметри автомобілів на аерофотознімках значно відрізняються від параметрів автомобілів у більшості поширених датасетів (наприклад, COCO). Наприклад, для набору даних COCO є характерним розподіл розмірів автомобілів у широкому діапазоні від мінімального до максимального (рис. 2.15), а також характерні ракурси автомобілів (вигляд

спереду, збоку, ззаду), але малопоширеними є вигляди зверху. Відповідно базові моделі ЗНМ YOLO не орієнтовані на розпізнавання зображень автомобілів, отриманих з БПЛА.

Тому в даній роботі з використанням розробленої методики та програми виконано донавчання моделі YOLOv8n на наборі зображень, отриманих з БПЛА. Зображення зчитувалися як кадри відеопотоку з інтервалом 5 секунд. У результаті аналізу відео тривалістю 13 хвилин шляхом отримано 157 кадрів, а шляхом селекції відібрано 99 унікальних кадрів (74 кадрів навчальної вибірки, 14 кадрів контрольної, 11 кадрів тестової). На кадрах рамками локалізовано положення автомобілів. Після донавчання модель YOLOv8n забезпечує точне детектування автомобілів (рис. 4.17), яке є достатнім для практичного застосування. Таким чином, за рахунок донавчання з автоматизованим формуванням датасету розроблена інтелектуальна система може адаптуватися до детектування автомобілів на зображеннях у різноманітних умовах.

Рисунок 4.17 – Приклад детектування автомобілів на зображенні, отриманому з БПЛА [184], моделлю YOLOv8n після донавчання

У випадку дистанційного зондування територій важливою є просторова локалізація результатів детектування. Сучасні відеокамери (зокрема, на БПЛА), дозволяють автоматично зберігати GPS-координати точки зйомки безпосередньо у метаданих зображення (найчастіше у форматах JPEG та TIFF). У програмі на мові Python проведено програмну обробку GPS-координат камер (географічну широту, довготу та висоту) у форматі EXIF (англ. Exchangeable Image File Format). EXIF є частиною контейнера файлу і зберігається разом із зображенням. Для роботи з геоданими в EXIF використано розділ GPS IFD (англ. Image File Directory), де зберігаються, зокрема, такі поля: GPSLatitude (широта), GPSLongitude (довгота), GPSAltitude (висота над рівнем моря).

Передача GPS-координат разом із зображенням дозволяє застосовувати результати детектування об’єктів для дистанційного зондування, оскільки при цьому виконується просторова локалізація всіх об’єктів. Перевагою розробленої інтелектуальної системи є автоматичне виявлення транспортних засобів на зображеннях, що дозволяє у значній мірі автоматизувати завдання аналізу сцен дорожнього руху.

Висновки до розділу 4

У четвертому розділі виконано програмну реалізацію та експериментальне дослідження інтелектуальної системи детектування зображень автомобілів із автоматизованим формуванням датасетів і донавчанням ЗНМ. За результатами проведеної роботи отримано такі результати:

1. Розроблено методику автоматизованого формування набору зображень, яка дозволяє шляхом селекції видаляти надлишкові кадри. Використано три етапи відбору кадрів: 1) глобальна селекція з використанням перцептивного хешування; 2) детектування автомобілів мережею-вчителем YOLOv8m і селекція кадрів

шляхом порівняння детектованих рамок автомобілів з існуючими рамками датасету за параметрами їх подібності (IoU, RMSE) з використанням угорського алгоритму; 3) локальна селекція кадрів шляхом порівняння перцептивних хешів рамок автомобілів з існуючими. Відібрані таким чином кадри анотуються і додаються до датасету. Під час тестування із 993 захоплених кадрів відібрано 121 унікальне зображення (~12 % від початкового обсягу).

2. Розроблено методику автоматизованого донавчання згорткової нейронної мережі YOLO малого розміру (Учень) на основі спеціалізованого датасету зображень, створеного ЗНМ YOLO середнього або великого розміру (Вчитель). Донавчання моделі YOLOv8n на автоматизовано сформованому датасеті дозволило підвищити метрику mAP50 з 0.6864 до 0.9292. Таке зростання точності детектування (на ~24 %) при збереженні високої швидкості роботи підтверджує доцільність адаптації нейромереж до конкретних умов сцени.
3. Методики автоматизованого формування датасету та донавчання ЗНМ реалізовано на мові Python в програмі «YOLOAutoDatasetCreation», яка призначена для детектування зображень автомобілів. У системі використано контейнеризацію (Docker) та ряд програмних бібліотек (Ultralytics, OpenCV, Redis). Розроблена програма функціонує на базі мікрокомп'ютера Raspberry Pi5 і виконує захоплення відео, селекцію кадрів, формування спеціалізованого датасету зображень, донавчання ЗНМ та детектування зображень автомобілів. Веб-інтерфейс програми розроблено на базі екосистеми React 18.3.
4. Показано, що розроблена інтелектуальна система забезпечує з високою точністю детектування автомобілів на зображеннях для різних розмірів, ракурсів та умов освітлення об'єктів. Обґрунтовано доцільність практичного застосування розробки для автоматизованого аналізу трафіку та стану дорожньої обстановки, виявлення автомобілів на аерофотознімках, моніторингу стану паркувальних майданчиків та в системах безпеки дорожнього руху.

ВИСНОВКИ

У дисертаційній роботі поставлено і розв’язано актуальну науково-прикладну задачу підвищення точності та швидкодії детектування зображень автомобілів шляхом вибору архітектури та версії згорткових нейронних мереж, попередньої обробки зображень та донавчання нейронних мереж з архітектурою YOLO на основі автоматизовано створених наборів даних, що дає можливість застосовувати такі навчені нейронні мережі для детектування зображень автомобілів та інших учасників дорожнього руху в прикладних системах комп’ютерного зору.

Основні результати дисертаційної роботи:

1. Проведено порівняльний аналіз існуючих нейромережевих архітектур, методів та програмних засобів для детектування зображень об'єктів, зокрема, транспортних засобів; у результаті такого аналізу зроблено висновок про доцільність застосування для детектування зображень автомобілів ЗНМ із архітектурою YOLO версій 8 та 11, які володіють задовільними характеристиками швидкодії та точності детектування.

2. Розроблено методику та програмні засоби на мові Python для попередньої обробки зображень шляхом збільшення їх локального контрасту, що забезпечило підвищення точності детектування зображень автомобілів засобами згорткової нейронної мережі з архітектурою YOLO за метриками Precision, Recall, IoU, AP та F1.

3. Розроблено методику та програмні засоби на мові Python для навчання різних версій моделей YOLO та для аналізу результатів навчання. Анотування набору зображень автомобілів виконано в ручному режимі засобами платформи Roboflow. За рахунок донавчання підвищено точність детектування об’єктів. Розроблено рекомендації для вибору версії YOLO з урахуванням вимог до точності, швидкодії та обсягу використаних ресурсів при вирішенні конкретної задачі детектування зображень об’єктів. Показано, що при порівнянні моделей YOLOv8m з YOLOv11m

для першої моделі отримано вищу точність детектування, а для другої – вищу швидкодію.

4. Розроблено методику для автоматизованого формування датасету, яка дозволяє шляхом селекції видаляти надлишкові кадри. Використано три етапи відбору кадрів: 1) глобальна селекція з використанням перцептивного хешування; 2) селекція кадрів шляхом порівняння рамок автомобілів з існуючими рамками датасету; 3) локальна селекція ділянок автомобілів з врахуванням їх перцептивних хешів. Під час тестування із 993 захоплених кадрів відібрано 121 унікальне зображення (~12 % від початкового обсягу).

5. Вперше розроблено методику з архітектурою «Вчитель-Учень» для донавчання згорткової нейронної мережі YOLO малого розміру (Учень) на основі датасету, створеного в автоматизованому режимі мережею YOLO великого розміру (Вчитель), що забезпечило високу точність та швидкодію детектування зображень автомобілів засобами моделі нейромережі малого розміру. Донавчання моделі YOLOv8n дозволило підвищити метрику mAP50 з 0.6864 до 0.9292 (на ~24 %)

6. Методики автоматизованого формування набору даних та донавчання ЗНМ реалізовано на мові Python в програмі «YOLOAutoDatasetCreation», яка призначена для детектування зображень автомобілів. Розроблено веб-інтерфейс для моніторингу та керування програмою.

7. Експериментальні дослідження підтвердили високу точність та швидкодію розроблених програмних засобів на прикладі детектування об’єктів на тестових зображеннях, продемонстровано можливості застосування розробленої інтелектуальної системи для оцінки автомобільного трафіку, аналізу зайнятості автомобільних парковок, контролю безпеки дорожнього руху та аналізу дорожньої обстановки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hadi R. A., Sulong G., George L. E. Vehicle Detection and Tracking Techniques: A Concise Review. *Signal & Image Processing: An International Journal (SIPIJ)*. 2014. Vol. 5, No. 1. P. 1–12. doi: 10.5121/sipij.2014.5101.

2. Zhu F., Li Z., Chen S., Xiong G. Parallel Transportation Management and Control System and Its Applications in Building Smart Cities. *IEEE Transactions on Intelligent Transportation Systems*. 2016. Vol. 17, No. 6. P. 1576–1585. doi: 10.1109/TITS.2015.2506156.

3. Bommes M., Fazekas A., Volkenhoff T., Oeser M. Video Based Intelligent Transportation Systems – State of the Art and Future Development. *Transportation Research Procedia*. 2016. Vol. 14. P. 4495–4504. doi: 10.1016/j.trpro.2016.05.372.

4. Auer A., Feese S., Lockwood S. History of Intelligent Transportation Systems: Report No. FHWA-JPO-16-329. U.S. Department of Transportation. 2016. URL: https://www.its.dot.gov/index.htm (date of access: 03.05.2025).

5. Pavlova O., Bilinska A., Holovatiuk A., Binkovskyi Y., Melnychuk D. Automated system for determining speed of cars ahead. *Computer systems and information technologies*. 2023. No. 3. P. 32–39. doi: 10.31891/csit-2023-3-4.

6. Баловсяк С.В., Стець С.Ю. Аналіз сучасних методів розпізнавання зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XI Міжнародної науково-практичної конференції (ПІКТ – 2022), м. Чернівці, 10–13 листопада 2022 р. Чернівці: Черн. нац. ун-т, 2022. С. 77-80.

7. Gani M. H. H., Khalifa O. O., Gunawan T. S., Shamsan E. Traffic intensity monitoring using multiple object detection with traffic surveillance cameras. *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA). IEEE*, 2017. P. 1–5. doi: 10.1109/ICSIMA.2017.8311983.

8. Chen Y., Hu W. Robust Vehicle Detection and Counting Algorithm Adapted to Complex Traffic Environments with Sudden Illumination Changes and Shadows. *Sensors*. 2020. Vol. 20, No. 9. Art. 2686. doi: 10.3390/s20092686.

9. Stelmakh O. P., Stetsenko I. V., Velyhotskyi D. V. Information Technology of Video Data Processing for Traffic Intensity Monitoring. *Control Systems and Computers.* 2020. No. 3 (287). P. 50–59. doi: 10.15407/csc.2020.03.050.

10. Zaid Al Bayati M. A., Çakmak M. Real-Time Vehicle Detection for Surveillance of River Dredging Areas Using Convolutional Neural Networks. *International Journal of Image, Graphics and Signal Processing*. 2023. Vol. 15, No. 5. P. 17–28. doi: 10.5815/ijigsp.2023.05.02.

11. Recky P. J. Total Solution For Smart Traffic and Toll Roads Management in Indonesia. *Devotion : Journal of Research and Community Service*. 2021. Vol. 3, No. 2. P. 149–157. doi: 10.36418/dev.v3i2.119.

12. Levkivskyi V. L., Marchuk D. K., Lobanchykova N. M., Pilkevych I. A. Available parking places recognition system. *2022 CEUR Workshop Proceedings*. 2022. Vol. 3077. P. 123-134. https://ceur-ws.org/Vol-3077/paper07.pdf.

13. Ashraf M. H., Jabeen F., Alghamdi H., Zia M. S., Almutairi M. S. HVD-Net: A Hybrid Vehicle Detection Network for Vision-Based Vehicle Tracking and Speed Estimation. *Journal of King Saud University - Computer and Information Sciences*. 2023. Vol. 35, No. 8. Art. 101657. doi: 10.1016/j.jksuci.2023.101657.

14. Shakhovska N., Turyk A., Sieck J., Sachenko A. The Novel Neural Network Architecture to Detect Car Damage. *2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2023. P. 508–512. doi: 10.1109/IDAACS58523.2023.10348825.

15. Kvetny R. N., Masliy R. V., Kyrylenko O. M. Detection and classification of traffic objects using the environment digits. *Optoelectronic information-power technologies*. 2020. Vol. 39, No. 1. P. 14–20. doi: 10.31649/1681-7893-2020-39-1-14-20.

16. Trivedi J. D., Mandalapu S. D., Dave D. H. Vision-based real-time vehicle detection and vehicle speed measurement using morphology and binary logical operation. *Journal of Industrial Information Integration*. 2022. Vol. 27. Art. 100280. doi: 10.1016/j.jii.2021.100280.

17. Svatiuk D., Svatiuk O., Belei O. Application of the convolutional neural networks for the security of the object recognition in a video stream. *Cybersecurity: Education, Science, Technique*. 2020. Vol. 4, No. 8. P. 97–112. doi: 10.28925/2663-4023.2020.8.97112.

18. Iftikhar S., Asim M., Zhang Z., El-Latif A. A. A. Advance generalization technique through 3D CNN to overcome the false positives pedestrian in autonomous vehicles. *Telecommunication Systems*. 2022. Vol. 80, No. 4. P. 545–557. doi: 10.1007/s11235-022-00930-1.

19. Ibrahim A. W. Vehicle Detection. Kaggle. URL: https://www.kaggle.com/code/abdallahwagih/vehicle-detection-cnn-acc-99-3 (date of access: 13.05.2025).

20. Singh A. Vehicles identification. Kaggle. URL: https://www.kaggle.com/code/abhijitsingh001/vehicles-identification-val-acc-99-62/notebook (date of access: 23.04.2025).

21. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. P. 779–788. doi: 10.1109/CVPR.2016.91.

22. Liu W., Anguelov D., Erhan D. Szegedy Ch., Reed S., Fu Ch.-Y., Berg A. SSD: Single Shot MultiBox Detector. 2015. P. 1-17. doi: 10.48550/ARXIV.1512.02325.

23. Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv. 2015. doi: 10.48550/ARXIV.1506.01497.

24. Wang R., Zhao H, Xu Z, Ding Y, Li G, Zhang Y and Li H. Real-time vehicle target detection in inclement weather conditions based on YOLOv4. *Frontiers in Neurorobotics*. 2023. Vol. 17. Art. 1058723. doi: 10.3389/fnbot.2023.1058723.

25. Tomas J. P. Q., Aquino A., David G. E. M., Del Ayre R. D., Alminiana J. K. L. YOLOv8-Based Vehicle Type Detection: An In-Depth Exploration of Deep Learning Techniques for Robust Analysis of Dashcam Footage. *Proceedings of the 2024 9th International Conference on Mathematics and Artificial Intelligence (Beijing, 2024)*. ACM, 2024. P. 91–96. doi: 10.1145/3670085.3670098.

26. Горєлов С., Бармак О., Манзюк Е. Метод виявлення багатороторних БПЛА засобами штучного інтелекту. *Herald of Khmelnytskyi National University. Technical sciences*. 2023. Вип. 329, № 6. С. 84–91. doi: 10.31891/2307-5732-2023-329-6-84-91.

27. Arushi0302. Car-Detection-with-YOLO. 2020. URL: https://github.com/Arushi0302/Car-Detection-with-YOLO (date of access: 02.05.2025).

28. Kelly S. Vehicle Detection Using YOLO. Medium. URL: https://medium.com/@spencer_kelly/vehicle-detection-using-yolo-612a3b9b54a9 (date of access: 28.04.2025).

29. Tsang S.-H. Review – YOLOv12: Attention-Centric Real-Time Object Detectors. Medium. URL: https://sh-tsang.medium.com/review-yolov12-attention-centric-real-time-object-detectors-2d601b1d94ad (date of access: 17.04.2025).

30. Vartika Agarwal V., Sharma S. EMVD: Efficient Multitype Vehicle Detection Algorithm Using Deep Learning Approach in Vehicular Communication Network for Radio Resource Management. *International Journal of Image, Graphics and Signal Processing*. 2022. Vol. 14, No. 2. P. 25–37. doi: 10.5815/ijigsp.2022.02.03.

31. Wanchaitanawong N., Tanaka M., Shibata T., Okutomi M. Multi-modal pedestrian detection with misalignment based on modal-wise regression and multi-modal IoU. *Journal of Electronic Imaging*. 2023. Vol. 32, No. 01. doi: 10.1117/1.JEI.32.1.013025.

32. Thakur N., Nagrath P., Jain R., Saini D., Sharma N., Hemanth D. J. Autonomous Pedestrian Detection model for Crowd Surveillance using Deep Learning Framework. 2022. In Review. doi: 10.21203/rs.3.rs-1194737/v1.

33. Kargin K. Computer Vision Fundamentals and OpenCV Overview. Medium. URL: https://medium.com/mlearning-ai/computer-vision-fundamentals-and-opencv-overview-9a30fe94f0ce (date of access: 19.05.2025).

34. Leal-Taixé L., Osep A. Computer Vision III: Detection, Segmentation and Tracking (CV3DST) (IN2375). Technical University of Munich. URL: https://dvl.in.tum.de/teaching/cv3dst-ss20/ (date of access: 06.05.2025).

35. Majumder A., Gopi M. Introduction to Visual Computing: Core Concepts in Computer Vision, Graphics, and Image Processing. 1st ed. CRC Press, 2018. 392 p. doi: 10.1201/9781315372846.

36. Білінська А., Біньковський Я., Головатюк А., Мельничук Д., Говорущенко Т. Автоматичне виявлення автомобільних порушників за допомогою комп'ютерного зору в рамках кіберфізичної системи запобігання аварійним ситуаціям. *Measuring and computing devices in technological processes*. 2024. Вип. 1. С. 176–185. doi: 10.31891/2219-9365-2024-77-22.

37. Кутковецький В. Я. Розпізнавання образів: навч. посібник. Миколаїв: ЧНУ ім. Петра Могили, 2017. 420 с.

38. He K., Zhang X., Ren S., Sun J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. arXiv:1406.4729. 2014. doi: 10.48550/ARXIV.1406.4729.

39. Zhu L., Spachos P. Towards Image Classification with Machine Learning Methodologies for Smartphones. *Machine Learning and Knowledge Extraction*. 2019. Vol. 1, No. 4. P. 1039–1057. doi: 10.3390/make1040059.

40. Hu Zh., Uhryn D., Ushenko Yu., Korolenko V., Lytvyn V., Vysotska V. System programming of a disease identification model based on medical images. *Proc. SPIE*:

Sixteenth International Conference on Correlation Optics. 2024. Vol. 12938. P. 129380F-1–129380F-4. https://doi.org/10.1117/12.3009245.

41. Uhryn D. I., Ushenko Yu. O., Dvorzhak V. V., Terletskyi T. V., Kaidyk O. L. Architecture of the intelligent system for risk management and recognition of mushroom species. *Optoelectronic Information-Power Technologies*. 2024. Vol. 48, No. 2. P. 114–127. doi: 10.31649/1681-7893-2024-48-2-114-127.

42. Abedi K. SSD: Single Shot Detection. Kaggle. URL: https://www.kaggle.com/code/abedi756/ssd-single-shot-detection (date of access: 23.05.2025).

43. Kolluri J., Das R. Multimodal Image Analysis Based Pedestrian Detection Using Optimization with Classification by Hybrid Machine Learning Model. *International Journal of Image, Graphics and Signal Processing.* 2025. Vol. 17, No. 1. P. 31–44. doi: 10.5815/ijigsp.2025.01.03.

44. Lee K. Mask R-CNN for Object Detection and Segmentation. GitHub. URL: https://github.com/leekunhee/Mask_RCNN (date of access: 23.05.2025).

45. Altunay H. C., Albayrak Z. A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks. *Engineering Science and Technology, an International Journal*. 2023. Vol. 38. Art. 101322. doi: 10.1016/j.jestch.2022.101322.

46. Multajam R., Mohamad Ayob A. F., Sanjaya W. S. M., Sambas A., Rusyn V., Samila A. Real-time detection and classification of fish in underwater environment using YOLOv5: a comparative study of deep learning architectures. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska.* 2024. Vol. 14, No. 3. P. 91–95. doi: 10.35784/iapgos.6022.

47. Zou Z., Chen K., Shi Z., Guo Y., Ye J. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE*. 2023. Vol. 111, No. 3. P. 257–276. doi: 10.1109/JPROC.2023.3238524.

48. Liu L., Ouyang, W., Wang, X., Fieguth P., Chen J., Liu X., Pietikainen M. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*. 2020. Vol. 128, No. 2. P. 261–318. doi: 10.1007/s11263-019-01247-4.

49. Diwakar, Raj D. Recent Object Detection Techniques: A Survey. *International Journal of Image, Graphics and Signal Processing*. 2022. Vol. 14, No. 2. P. 47–60. doi: 10.5815/ijigsp.2022.02.05.

50. Du K., Bobkov A. An Overview of Object Detection and Tracking Algorithms. *INTELS'22. MDPI*, 2023. P. 22. doi: 10.3390/engproc2023033022.

51. Xu L., Jia J., Matsushita Y. Motion Detail Preserving Optical Flow Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2012. Vol. 34, No. 9. P. 1744–1757. doi: 10.1109/TPAMI.2011.236.

52. Badura S., Lieskovsky A., Mokrys M. With shadow elimination towards effective foreground extraction. *2011 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT).* 2011. P. 404–408. doi: 10.1109/ISSPIT.2011.6151596.

53. Pagire V., Chavali M., Kale A. A comprehensive review of object detection with traditional and deep learning methods. *Signal Processing*. 2025. Vol. 237. Art. 110075. doi: 10.1016/j.sigpro.2025.110075.

54. Gonzalez R., Woods R. Digital image processing. 4th edition. Pearson/ Prentice Hall, NY, 2018. 1022 p.

55. Kim D., Hwang D. Intelligent Imaging and Analysis. Switzerland, Basel: MDPI, 2020. 492 p.

56. Balovsyak S., Derevyanchuk O., Kovalchuk V., Kravchenko H., Kozhokar M. Face Mask Recognition by the Viola-Jones Method Using Fuzzy Logic. *International Journal of Image, Graphics and Signal Processing*. 2024. Vol. 16, No. 3. P. 39–51. doi: 10.5815/ijigsp.2024.03.04.

57. Malagoli E., Di Persio L. 2D Object Detection: A Survey. *Mathematics*. 2025. Vol. 13, No. 6. Art. 893. doi: 10.3390/math13060893.

58. Dolhopolov S., Kruk P., Zhuk R., Honcharenko T. Methods of image pre-processing for automated object recognition systems. *Management of Development of Complex Systems*. 2025. Vol. 62. P. 155–163. doi: 10.32347/2412-9933.2025.62.155-163.

59. O'Mahony N., Campbell S., Carvalho A., Harapanahalli S., Hernandez G., Krpalkova L., Riordan D., Walsh J. Deep Learning vs. Traditional Computer Vision. *Advances in Intelligent Systems and Computing*. Vol. 943: Advances in Computer Vision. Cham: Springer, 2020. P. 128–144. doi: 10.1007/978-3-030-17795-9_10.

60. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. 2017. Vol. 60, No. 6. P. 84–90. doi: 10.1145/3065386.

61. Савченко А. С., Синельніков О. О. Методи та системи штучного інтелекту. Київ: НАУ, 2017. 132 с.

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

63. Субботін С. О. Нейронні мережі: теорія та практика: навч. посібник. Житомир: Вид. О.О. Євенок, 2020. 184 с. URL: http://eir.zntu.edu.ua/bitstream/123456789/6800/1/Subbotin_Neural.pdf.

64. Russell S. J., Norvig P. Artificial intelligence: a modern approach. *Fourth edition.* Harlow: Pearson, 2022. 1166 p.

65. Orlovskiy O. V., Sohrab K., Ostapov S. E., Hazdyuk K. P., Shumylyak L. M. Multilingual text classifier using pre-trained universal sentence encoder model. *Radio Electronics, Computer Science, Control*. 2022. No. 3. P. 102. doi: 10.15588/1607-3274-2022-3-10.

66. Balovsyak S., Odaiska Kh., Yakovenko O., Iakovlieva I. Adjusting the Brightness and Contrast parameters of digital video cameras using artificial neural networks. *Proc. SPIE*: Sixteenth International Conference on Correlation Optics. 2024. Vol. 12938, P. 129380I-1 – 129380I-4. https://doi.org/10.1117/12.3009429.

67. Minaee S., Boykov Y. Y., Porikli F., Plaza A. J., Kehtarnavaz N., Terzopoulos D. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2021. P. 1–1. doi: 10.1109/TPAMI.2021.3059968.

68. Li Z., Liu F., Yang W., Peng S., Zhou J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*. 2022. Vol. 33, No. 12. P. 6999–7019. doi: 10.1109/TNNLS.2021.3084827.

69. Tomka Yu. Ya., Talakh M. V., Dvorzhak V. V., Ushenko O. G. Practical Aspects of Forming Training/Test Samples for Convolutional Neural Networks. *Optoelectronic Information-Power Technologies*. 2022. Vol. 43, No. 1. P. 24–35. doi: 10.31649/1681-7893-2022-43-1-24-35.

70. Soni A., Rai A. An Efficient CNN Model for Automatic Diagnosis of Cardiomegaly from Chest Radiographic *International Journal of Image, Graphics and Signal Processing*. 2023. Vol. 15, No. 3. P. 81–96. doi: 10.5815/ijigsp.2023.03.07.

71. Mohan Bhuma C., Kongara R. A Novel Technique for Image Retrieval based on Concatenated features extracted from big dataset pre-trained CNNs. *International Journal of Image, Graphics and Signal Processing*. 2023. Vol. 15, No. 2. P. 1–12. doi: 10.5815/ijigsp.2023.02.01.

72. Tomka Yu. Ya., Talakh M. V., Dvorzhak V. V., Ushenko O. G. Implementation of a convolutional neural network using Tensorflow machine learning platform. *Optoelectronic Information-Power Technologies*. 2023. Vol. 44, No. 2. P. 55–65. doi: 10.31649/1681-7893-2022-44-2-55-65.

73. Lee B. The New Best Python Package for Visualising Network Graphs. Towards Data Science. URL: https://towardsdatascience.com/the-new-best-python-package-for-visualising-network-graphs-e220d59e054e (date of access: 13.05.2025).

74. Aamir M., Rahman Z., Ahmed Abro W., Tahir M., Ahmed S. An Optimized Architecture of Image Classification Using Convolutional Neural Network. *International*

Journal of Image, Graphics and Signal Processing. 2019. Vol. 11, No. 10. P. 30–39. doi: 10.5815/ijigsp.2019.10.05.

75. Izonin I., Tkachenko R., Krak I., Berezsky O., Shevchuk I., Shandilya S. K. A cascade ensemble-learning model for the deployment at the edge: case on missing IoT data recovery in environmental monitoring systems. *Frontiers in Environmental Science*. 2023. Vol. 11. Art. 1295526. doi: 10.3389/fenvs.2023.1295526.

76. Gu J., Wang Zh., Kuen J., Ma L., Shahroudy A., Shuai B., Liu T., Wang X., Wang L., Wang G., Cai J., Chen T. Recent Advances in Convolutional Neural Networks. *arXiv*: 1512.07108. 2017. doi: 10.48550/arXiv.1512.07108.

77. Berezsky O., Liashchynskyi P., Pitsun O., Liashchynskyi P., Berezkyy M. Comparison of Deep Neural Network Learning Algorithms for Biomedical Image Processing. *CEUR Workshop Proceedings*. 2022. Vol. 3302. P. 135-145. https://ceur-ws.org/Vol-3302/paper7.pdf.

78. Farid A., Hussain F., Khan K., Shahzad M., Khan U., Mahmood Z. A Fast and Accurate Real-Time Vehicle Detection Method Using Deep Learning for Unconstrained Environments. *Applied Sciences*. 2023. Vol. 13, No. 5. Art. 3059. doi: 10.3390/app13053059.

79. Radiuk P., Barmak O., Manziuk E., Krak I. Explainable Deep Learning: A Visual Analytics Approach with Transition Matrices. *Mathematics*. 2024. Vol. 12, No. 7. Art. 1024. doi: 10.3390/math12071024.

80. Khan A., Sohail A., Zahoora U., Qureshi A. S. A Survey of the Recent Architectures of Deep Convolutional Neural Networks. arXiv. 2019. doi: 10.48550/ARXIV.1901.06032.

81. Khan A., Sohail A., Zahoora U., Qureshi A. S. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*. 2020. Vol. 53, No. 8. P. 5455–5516. doi: 10.1007/s10462-020-09825-6.

82. Alto V. Understanding the Inception Module in Googlenet. Medium. URL: https://valentinaalto.medium.com/understanding-the-inception-module-in-googlenet-2e1b7c406106 (date of access: 03.04.2025).

83. Bala S. A., Kant S. Dense Dilated Inception Network for Medical Image Segmentation. *International Journal of Advanced Computer Science and Applications*. 2020. Vol. 11, No. 11. doi: 10.14569/IJACSA.2020.0111195.

84. Стець С.Ю. Підвищення точності нейромережного розпізнавання зображень за допомогою модуля Inception. *Проблеми інформатики та комп'ютерної техніки*: праці XII Міжнародної науково-практичної конференції (ПІКТ – 2023), м. Чернівці, 10-12 листопада 2023 р. Чернівці: Черн. нац. ун-т, 2023. С. 61-63.

85. Van Linh N. Deeplearning.ai: CNN Week 2 — Convolutional Neural Network Architecture. Medium. URL: https://medium.com/datatype/deeplearning-ai-cnn-week-2-97e075f8c801 (date of access: 05.05.2025).

86. Баловсяк С.В., Стець С.Ю. Використання модуля Inception для підвищення точності розпізнавання зображень у згорткових нейронних мережах. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*: матеріали міжнародної наукової інтернет-конференції, 8-9 червня 2023 р. Вип. 78. Тернопіль, 2023. С. 30-32.

87. Buhl N. YOLO Object Detection Explained: Evolution, Algorithm, and Applications. Encord. URL: https://encord.com/blog/yolo-object-detection-guide/ (date of access: 19.05.2025).

88. Mahmood Z., Khan K., Khan U., Adil S. H., Ali S. S. A., Shahzad M. Towards Automatic License Plate Detection. *Sensors*. 2022. Vol. 22, No. 3. Art. 1245. doi: 10.3390/s22031245.

89. Vu T., Kang H., Yoo C. D. SCNet: Training Inference Sample Consistency for Instance Segmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021. Vol. 35, No. 3. P. 2701–2709. doi: 10.1609/aaai.v35i3.16374.

90. Qiao S., Chen L.-C., Yuille A. DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution. arXiv. 2020. doi: 10.48550/ARXIV.2006.02334.

91. Tan M., Le Q. V. EfficientNetV2: Smaller Models and Faster Training. arXiv. 2021. doi: 10.48550/ARXIV.2104.00298.

92. Carion N., Massa F., Synnaeve G., Usunier N., Kirillov A., Zagoruyko S. End-to-End Object Detection with Transformers. arXiv. 2020. doi: 10.48550/ARXIV.2005.12872.

93. Liu Z., Lin Y., Cao Y., Hu H., Wei Y., Zhang Zh., Lin S., Guo B. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. arXiv. 2021. doi: 10.48550/ARXIV.2103.14030.

94. Erden K. Introduction to Object Detection: Vehicle Detection with OpenCV and Cascade Classifiers. Medium. URL: https://medium.com/@kaanerdenn/introduction-to-object-detection-vehicle-detection-with-opencv-and-cascade-classifiers-8c6834191a0b (date of access: 19.05.2025).

95. Yadav U., Banerjee S. Advancing Vehicle Image Recognition via Cloud Computing: A Review. *International Journal for Research in Applied Science and Engineering Technology*. 2024. Vol. 12, No. 6. P. 2114–2120. doi: 10.22214/ijraset.2024.63452.

96. Spichkova M., Severin A., Amornpatchara C., Le F., Tran T. H., Padmaprasad P. Analysis of AWS Rekognition and Azure Custom Vision Performance in Parking Sign Recognition. *Sensors*. 2025. Vol. 25, No. 19. Art. 5983. doi: 10.3390/s25195983.

97. AWS. Detecting objects and concepts. Amazon Rekognition Developer Guide. URL: https://docs.aws.amazon.com/rekognition/latest/dg/labels.html (date of access: 13.04.2025).

98. Comma.ai. Openpilot 0.10.3. Comma.ai Blog. URL: https://blog.comma.ai/0103release/ (date of access: 24.04.2025).

99. Baresi L., Tamburri D. A. Architecting Artificial Intelligence for Autonomous Cars: The OpenPilot Framework. *Software Architecture*. Vol. 14212: Lecture Notes in Computer Science. Cham: Springer, 2023. P. 189–204. doi: 10.1007/978-3-031-42592-9_13.

100. Toschi A., Sanic M., Leng J., Chen Q., Wang C., Guo M. Characterizing Perception Module Performance and Robustness in Production-Scale Autonomous Driving System. *Network and Parallel Computing*. Vol. 11783: Lecture Notes in Computer Science. Cham: Springer, 2019. P. 235–247. doi: 10.1007/978-3-030-30709-7_19.

101. Baidu Apollo. Apollo Navigation Pilot. Apollo Auto. URL: https://www.apollo.auto/en/anp (date of access: 24.04.2025).

102. Palani S. Principles of Digital Signal Processing. Springer, Cham. 2022. 689 p.

103. Мельник Р. А. Алгоритми та методи опрацювання зображень. Львів: Вид-во Львівської політехніки, 2017. 160 с.

104. Tuba E., Bacanin N., Strumberger I., Tuba M. Convolutional Neural Networks Hyperparameters Tuning. Studies in Computational Intelligence. Vol. 973: *Artificial Intelligence: Theory and Applications.* Cham: Springer, 2021. P. 65–84. doi: 10.1007/978-3-030-72711-6_4.

105. Alzubaidi L., Zhang J., Humaidi A.J., Al-Dujaili A., Duan Y., Al-Shamma O., Santamaría J., Fadhel M.A., Al-Amidie M., Farhan L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*. 2021. Vol. 8, No. 1. Art. 53. doi: 10.1186/s40537-021-00444-8.

106. Jocher G., Chaurasia A., Qiu J. YOLOv8 by Ultralytics. GitHub. 2023. URL: https://github.com/ultralytics/ultralytics (date of access: 23.05.2025).

107. Deepak N. R. YOLOv8 vs YOLOv11: A Comparison. Python in Plain English. URL: https://python.plainenglish.io/yolov8-vs-yolov11-a-comparison-94426b382367 (date of access: 23.05.2025).

108. Germanov A. How to Detect Objects in Images Using the YOLOv8 Neural Network. freeCodeCamp. URL: https://www.freecodecamp.org/news/how-to-detect-objects-in-images-using-yolov8/ (date of access: 23.05.2026).

109. Tsang S.-H. Review: YOLOv8 (Object Detection). Medium. URL: https://sh-tsang.medium.com/review-yolov8-object-detection-5214fa105731 (date of access: 23.05.2025).

110. Tsang S.-H. Brief Review: YOLOv5 for Object Detection. Medium. URL: https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a (date of access: 23.05.2025).

111. Uhryn D., Shved D. Modelling the Identification and Classification of Military Air Objects Based on Machine Learning. *Security of Infocommunication Systems and Internet of Things*. 2024. Vol. 2, No. 1. Art. 01001. doi: 10.31861/sisiot2024.1.01001.

112. Pethiyagoda N. Vehicle Dataset for YOLO. Kaggle. URL: https://www.kaggle.com/datasets/nadinpethiyagoda/vehicle-dataset-for-yolo (date of access: 23.05.2025).

113. Xu Y., Du W., Deng L., Zhang Y., Wen W. Ship target detection in SAR images based on SimAM attention YOLOv8. *IET Communications*. 2024. Vol. 18, No. 19. P. 1428–1436. doi: 10.1049/cmu2.12837.

114. Nepal U., Eslamiat H. Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *Sensors*. 2022. Vol. 22, No. 2. Art. 464. doi: 10.3390/s22020464.

115. Wang C.-Y., Bochkovskiy A., Liao H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv. 2022. doi: 10.48550/ARXIV.2207.02696.

116. Баловсяк С.В., Стець С.Ю. Детектування зображень пішохідних переходів за допомогою нейронних мереж YOLO. *Проблеми інформатики та комп'ютерної*

техніки: праці XIII Міжнародної науково-практичної конференції (ПІКТ – 2024), м. Чернівці, 1–3 листопада 2024 р. Чернівці: Черн. нац. ун-т, 2024. С. 49-51.

117. Yahuza K., Umar A. M., Baha'uddeen S. D., Atalabi E. T., Mukhtar G. L., Abdulkadir B. Recent Advancements in Detection and Quantification of Malaria Using Artificial Intelligence. *UMYU Journal of Microbiology Research (UJMR)*. 2024. Vol. 9, No. 2. P. 1–21. doi: 10.47430/ujmr.2492.001.

118. Wang C.-Y., Liao H.-Y. M., Yeh I.-H., Wu Y.-H., Chen P.-Y., Hsieh J.-W. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. arXiv. 2019. doi: 10.48550/ARXIV.1911.11929.

119. Liu S., Qi L., Qin H., Shi J., Jia J. Path Aggregation Network for Instance Segmentation. arXiv. 2018. doi: 10.48550/arXiv.1803.01534.

120. Hui J. mAP (mean Average Precision) for Object Detection. Towards Data Science. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (date of access: 27.05.2025).

121. Amrutkar P. The Complete Guide to Object Detection Evaluation Metrics: From IoU to mAP and More. Medium. URL: https://medium.com/@prathameshamrutkar3/the-complete-guide-to-object-detection-evaluation-metrics-from-iou-to-map-and-more-1a23c0ea3c9d (date of access: 27.05.2025).

122. Марчук Д. К., Граф М. С. Методи оцінки ефективності моделей виявлення об'єктів у комп'ютерному зорі. *Вісник Херсонського національного технічного університету*. 2023. Вип. 2 (85). С. 181–186. doi: 10.35546/kntu2078-4481.2023.2.25.

123. Padilla R., Netto S. L., Da Silva E. A. B. A Survey on Performance Metrics for Object-Detection Algorithms. *International Conference on Systems, Signals and Image Processing (IWSSIP)*. July 2020. IEEE, 2020. P. 237–242. doi: 10.1109/IWSSIP48289.2020.9145130.

124. Lin T.-Y., Maire M., Belongie S., Bourdev L., Girshick R., Hays J., Perona P., Ramanan D., Zitnick C.L., Dollar P. Microsoft COCO: Common Objects in Context. arXiv. 2014. doi: 10.48550/ARXIV.1405.0312.

125. Balovsyak S., Stets S. Preprocessing of object images before their detection using YOLO neural network. *Security of Infocommunication Systems and Internet of Things*. 2025. Vol. 3, No. 2, P. 1-5.

126. Omar H. Kh., Tawfiq N. E. Face Recognition Based on Histogram Equalization and LBP Algorithm. *Academic Journal of Nawroz University*. 2019. Vol. 8, No. 3. P. 33. doi: 10.25007/ajnu.v8n3a394.

127. Bhandari A. K. A logarithmic law based histogram modification scheme for naturalness image contrast enhancement. *Journal of Ambient Intelligence and Humanized Computing*. 2020. Vol. 11, No. 4. P. 1605–1627. doi: 10.1007/s12652-019-01258-6.

128. Tao P., Pei Y., Celenk M., Fu Q., Wu A. Adaptive image enhancement method using contrast limitation based on multiple layers BOHE. *Journal of Ambient Intelligence and Humanized Computing*. 2020. Vol. 11, No. 11. P. 5031–5043. doi: 10.1007/s12652-020-01810-9.

129. Thakur N., Uddin Khan N., Sharma S. D., Bashar A. Pelican Optimization based Histogram Equalization for Contrast Enhancement and Brightness Preservation. *International Journal of Image, Graphics and Signal Processing*. 2025. Vol. 17, No. 3. P. 12–33. doi: 10.5815/ijigsp.2025.03.02.

130. Bai L., Zhang W., Pan X., Zhao C. Underwater Image Enhancement Based on Global and Local Equalization of Histogram and Dual-Image Multi-Scale Fusion. *IEEE Access*. 2020. Vol. 8. P. 128973–128990. doi: 10.1109/ACCESS.2020.3009161.

131. Bhandari A. K., Kandhway P., Maurya S. Salp Swarm Algorithm-Based Optimally Weighted Histogram Framework for Image Enhancement. *IEEE Transactions on Instrumentation and Measurement*. 2020. Vol. 69, No. 9. P. 6807–6815. doi: 10.1109/TIM.2020.2976279.

132. Kuran U., Kuran E. C. Parameter selection for CLAHE using multi-objective cuckoo search algorithm for image contrast enhancement. *Intelligent Systems with Applications*. 2021. Vol. 12. Art. 200051. doi: 10.1016/j.iswa.2021.200051.

133. Fawzi A., Achuthan A., Belaton B. Adaptive Clip Limit Tile Size Histogram Equalization for Non-Homogenized Intensity Images. *IEEE Access*. 2021. Vol. 9. P. 164466–164492. doi: 10.1109/ACCESS.2021.3134170.

134. Saifullah S., Pranolo A., Drezewski R. Comparative analysis of image enhancement techniques for braintumor segmentation: contrast, histogram, and hybrid approaches. *E3S Web of Conferences.* 2024. Vol. 501. Art. 01020. doi: 10.1051/e3sconf/202450101020.

135. Yoshimi Y., Mine Y., Ito S., Takeda S., Okazaki S., Nakamoto T., Nagasaki T., Kakimoto N., Murayama T., Tanimoto K. Image preprocessing with contrast-limited adaptive histogram equalization improves the segmentation performance of deep learning for the articular disk of the temporomandibular joint on magnetic resonance images. *Oral Surgery, Oral Medicine, Oral Pathology and Oral Radiology*. 2024. Vol. 138, No. 1. P. 128–141. doi: 10.1016/j.oooo.2023.01.016.

136. Saifullah S., Drezewski R. Modified Histogram Equalization for Improved CNN Medical Image Segmentation. *Procedia Computer Science*. 2023. Vol. 225. P. 3021–3030. doi: 10.1016/j.procs.2023.10.295.

137. Omarova G., Aitkozha, Z., Sadirmekova, Z., Zhidekulova, G., Kazimova, D., Muratkhan, R., Takuadina, A., & Abdykeshova, D. Devising a methodology for X-ray image contrast enhancement by combining CLAHE and gamma correction. *Eastern-European Journal of Enterprise Technologies*. 2022. Vol. 3, No. 2 (117). P. 18–29. doi: 10.15587/1729-4061.2022.258092.

138. Khan A. H., Ahmed S., Bera S. K., Mirjalili S., Oliva D., Sarkar R. Enhancing the contrast of the grey-scale image based on meta-heuristic optimization algorithm. *Soft Computing*. 2022. Vol. 26, No. 13. P. 6293–6315. doi: 10.1007/s00500-022-07033-8.

139. Balovsyak S., Odaiska Kh. Automatic Determination of the Gaussian Noise Level on Digital Images by High-Pass Filtering for Regions of Interest. *Cybernetics and Systems Analysis*. 2018. Vol. 54, No. 4. P. 662–670. doi: 10.1007/s10559-018-0067-3.

140. Баловсяк С.В., Стець С.Ю. Комп'ютерна програма «Цифрова обробка зображень об'єктів перед їх детектуванням нейронною мережею YOLO» («YOLOContrast25»). Український національний офіс інтелектуальної власності та інновацій (УКРНОІВІ). Свідоцтво про реєстрацію авторського права на твір, №141329, 2.12.2025. Ідентифікатор CR0380021225. https://sis.nipo.gov.ua/uk/services/original-document.

141. Adhikari B., Li J., Michel E.S., Dykes J., Tseng T.-M.P., Tagert M.L., Chen D. A Comprehensive Evaluation of YOLO-based Deer Detection Performance on Edge Devices. arXiv. 2025. doi: 10.48550/arXiv.2509.20318.

142. Баловсяк С.В., Стець С.Ю. Попередня обробка зображень об'єктів перед їх детектуванням засобами нейронної мережі YOLO. *Фізико-технологічні проблеми передачі, обробки та зберігання інформації в інфокомунікаційних системах*: матеріали X Міжнародної науково-практичної конференції (ПРЕДТ-2025). 15-17 травня 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 181-182

143. Штовба С.Д., Мазуренко В.В. Інтелектуальні технології ідентифікації залежностей. Лабораторний практикум: електронний навчальний посібник. Вінниця: ВНТУ, 2014. 113 с.

144. Сидор П., Виклюк Я. Ансамблеві моделі прогнозування повеней у Великій Британії на основі сонячної активності. *Herald of Khmelnytskyi National University. Technical sciences*. 2024. Вип. 333, № 2. С. 218–231. doi: 10.31891/2307-5732-2024-333-2-35.

145. Невінський Д.В., Мартьянов Д.І., Господарський О.А., Виклюк Я.І., Сем’янів І.О. Визначення детермінантів туберкульозу: аналіз методів машинного навчання та нейронних мереж. *Зв’язок*. 2025. Т. 3. С. 73-86. https://doi.org/10.31673/2412-9070.2025.021793.

146. Господарчук Д., Невінський Д., Мартьянов Д., Виклюк Я., Сем'янів І. Оптимізація моделей машинного навчання для оцінки ризику поширення туберкульозу. *Управління розвитком складних систем*. 2025. Т. 61. С. 160–169. doi: 10.32347/2412-9933.2025.61.160-169.

147. Uhryn D. I., Ushenko Yu. O., Yatsko O. M., Dovhun A. Ya., Dobrovolsky Yu. G. Intelligent systems for forecasting demographic changes and their impact on marketing strategies in the IT industry. *Optoelectronic Information-Power Technologies*. 2024. Vol. 48, No. 2. P. 13–23. doi: 10.31649/1681-7893-2024-48-2-13-23.

148. Гнєденко Б.В. Курс теорії ймовірностей. Київ: ВПЦ Київський університет, 2010. 464 с. https://probability.knu.ua/userfiles/yamnenko/Gnedenko.pdf.

149. Карташов М. В. Імовірність, процеси, статистика. Київ : ВПЦ Київський університет, 2007. 504 с. https://probability.knu.ua/userfiles/kmv/VPS_Pv.pdf.

150. Carranza-Garcia M., Torres-Mateo J., Lara-Benitez P., Garcia-Gutierrez J. On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. *Remote Sensing*. 2020. Vol. 13, No. 1. Art. 89. doi: 10.3390/rs13010089.

151. Hassaballah M., Kenk M. A., Muhammad K., Minaee S. Vehicle Detection and Tracking in Adverse Weather Using a Deep Learning Framework. *IEEE Transactions on Intelligent Transportation Systems.* 2021. Vol. 22, No. 7. P. 4230–4242. doi: 10.1109/TITS.2020.3014013.

152. Valverde M., Moutinho A., Zacchi J.-V. A Survey of Deep Learning-Based 3D Object Detection Methods for Autonomous Driving Across Different Sensor Modalities. *Sensors*. 2025. Vol. 25, No. 17. Art. 5264. doi: 10.3390/s25175264.

153. Durusoy O. Open-Source Datasets for Image Processing and Artificial Intelligence Research: A Comparison of ImageNet and MS COCO Datasets. *International Journal of Sciences and Innovation Engineering*. 2025. Vol. 2, No. 5. P. 639–653. doi: 10.70849/IJSCI0205202575.

154. Kiran B. R., Sobh I., Talpaert V., Mannion P., Al Sallab S., Yogamani S., Perez P. Deep Reinforcement Learning for Autonomous Driving: A Survey. *arXiv*. 2020. doi: 10.48550/ARXIV.2002.00444.

155. Стець С. Ю. Аналіз точності та швидкодії детекції автомобілів за допомогою нейронних мереж YOLOv8 та YOLOv11. *Herald of Khmelnytskyi National University. Technical sciences*. 2025. Т. 357, № 5.2. С. 123–130. doi: 10.31891/2307-5732-2025-357-74.

156. Tribuana D., Hazriani, Arda A. L. Image Preprocessing Approaches Toward Better Learning Performance with CNN. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*. 2024. Vol. 8, No. 1. P. 1–9. doi: 10.29207/resti.v8i1.5417.

157. Yousif M. J. Enhancing The Accuracy of Image Classification Using Deep Learning and Preprocessing Methods. *Artificial Intelligence & Robotics Development Journal*. 2024. Vol. 3, No. 4, P. 269-281. doi: 10.52098/airdj.2023348.

158. Balovsyak S., Kroitor O., Odaiska Kh., Salem A.B.M., Stets S. Car image recognition using convolutional neural network with efficient met architecture. *CEUR Workshop Proceeding*. 2024. Vol. 3675: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, IntelITSIS 2024. P. 182-

159. Selvan A. A Comprehensive Study on Image Preprocessing Techniques for Enhanced Image Analysis. 2025. doi: 10.13140/RG.2.2.33569.67683.

160. Wang Z., Zhang K., Wu F., Lv H. YOLO-PEL: The Efficient and Lightweight Vehicle Detection Method Based on YOLO Algorithm. *Sensors*. 2025. Vol. 25, No. 7. Art. 1959. doi: 10.3390/s25071959.

161. Tong K., Wu Y. Rethinking PASCAL-VOC and MS-COCO dataset for small object detection. *Journal of Visual Communication and Image Representation*. 2023. Vol. 93. Art. 103830. doi: 10.1016/j.jvcir.2023.103830.

162. Murray D. G., Simsa J., Klimovic A., Indyk I. tf.data: a machine learning data processing framework. *Proceedings of the VLDB Endowment*. 2021. Vol. 14, No. 12. P. 2945–2958. doi: 10.14778/3476311.3476374.

163. Soni N. Forensic Value of Exif Data: An Analytical Evaluation of Metadata Integrity across Image Transfer Methods. *Perspectives in Legal and Forensic Sciences*. 2025. Vol. 2, No. 2. Art. 10006. doi: 10.70322/plfs.2025.10006.

164. Edwards K., Scalisi C., DeMars-Smith J., Lee K. Google Colab for Teaching CS and ML. *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, 2024). ACM, 2024. P. 1925. doi: 10.1145/3626253.3635432.

165. Pimentel J. F., Murta L., Braganholo V., Freire J. Understanding and improving the quality and reproducibility of Jupyter notebooks. *Empirical Software Engineering*. 2021. Vol. 26, No. 4. Art. 65. doi: 10.1007/s10664-021-09961-9.

166. Fang H., Chockchowwat S., Sundaram H., Park Y. Enhancing Computational Notebooks with Code+Data Space Versioning. *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems* (Yokohama, 2025). ACM, 2025. P. 1–17. doi: 10.1145/3706598.3714141.

167. Majek K. 4K Road traffic video for object detection and tracking. 2018. URL: https://youtu.be/MNn9qKG2UFI (date of access: 30.05.2025).

168. Баловсяк С., Стець С. Автоматизоване створення спеціалізованого датасету для зображень автомобілів. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. Т. 359 (6.2). С. 278-285. https://doi.org/10.31891/2307-5732-2025-359-111.

169. Biswas R., Blanco-Medina P. State of the Art: Image Hashing. arXiv. 2021. doi: 10.48550/ARXIV.2108.11794.

170. Hassan A. Perceptual image hashing by exploiting a visual attention model along with discrete wavelet transform and discrete cosine transform. *Journal of Electronic Imaging*. 2024. Vol. 33, No. 06. doi: 10.1117/1.JEI.33.6.063057.

171. Weng X., Wang J., Held D., Kitani K. 3D Multi-Object Tracking: A Baseline and New Evaluation Metrics. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Las Vegas, 2020). IEEE, 2020. P. 10359–10366. doi: 10.1109/IROS45743.2020.9341164.

172. Kuhn H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955. Vol. 2, No. 1–2. P. 83–97. doi: 10.1002/nav.3800020109.

173. Aironi C., Cornell S., Squartini S. A Graph-Based Neural Approach to Linear Sum Assignment Problems. *International Journal of Neural Systems*. 2024. Vol. 34, No. 03. Art. 2450011. doi: 10.1142/S0129065724500114.

174. Wang G., Song M., Hwang J.-N. Recent Advances in Embedding Methods for Multi-Object Tracking: A Survey. arXiv. 2024. doi: 10.48550/arXiv.2205.10766.

175. Ye Y., Ke X., Yu Z. A Cost Matrix Optimization Method Based on Spatial Constraints under Hungarian Algorithm. *2020 6th International Conference on Robotics and Artificial Intelligence* (Singapore, 2020). ACM, 2020. P. 134–139. doi: 10.1145/3449301.3449324.

176. Стець С.Ю. Донавчання нейронної мережі YOLO за допомогою автоматизовано створеного датасету зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XIV Міжнародної науково-практичної конференції (ПІКТ – 2025), м. Чернівці, 13-15 листопада 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 64-66.

177. Howard J., Gugger S. Fastai: A Layered API for Deep Learning. *Information*. 2020. Vol. 11, No. 2. Art. 108. doi: 10.3390/info11020108.

178. Gupta R., Nahrstedt K. Performance Characterization of Containers in Edge Computing. arXiv. 2025. doi: 10.48550/arXiv.2505.02082.

179. Al-Rakhami M. et al. A lightweight and cost effective edge intelligence architecture based on containerization technology. *World Wide Web*. 2020. Vol. 23, No. 2. P. 1341–1360. doi: 10.1007/s11280-019-00692-y.

180. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. 2018. URL: https://doi.org/10.17487/RFC8446.

181. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. 2015. URL: https://doi.org/10.17487/RFC7519.

182. Hospodariuk S., Samila A. Review of Nuclear Quadrupole Resonance Signal Processing and Analysis Methods. *Security of Infocommunication Systems and Internet of Things*. 2025. Vol. 3, No. 2, Paper 02004, P. 1-7. doi: 10.31861/sisiot2025.2.02004.

183. Roboflow. Cars From Drone Computer Vision Model. VilniusTech University Dataset. 2024. URL: https://universe.roboflow.com/vilniustech-university/cars-from-drone-f5kny (date of access: 30.05.2025)

184. Qpu523. HDI-Dataset. Drone Data Analytics for Measuring Traffic Metrics at Intersections in High-Density Areas. URL: https://github.com/Qpu523/HDI-Dataset. (date of access: 26.05.2025)

ДОДАТКИ

Додаток А

Список публікацій здобувача за темою дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації:

Наукові праці у фахових виданнях України:

1. **Стець С.** Аналіз точності та швидкодії детекції автомобілів за допомогою нейронних мереж YOLOV8 та YOLOV11. *Herald of Khmelnytskyi National University*. Technical Sciences. 2025. Т. 357 (5.2). С. 123-130. https://doi.org/10.31891/2307-5732-2025-357-74.
2. Balovsyak S., **Stets S**. Preprocessing of object images before their detection using YOLO neural network. *Security of Infocommunication Systems and Internet of Things*. 2025. Vol. 3, No. 2, Paper 02002. P. 1-5. ISSN 2786-8443. https://doi.org/10.31861/sisiot2025.2.02002. (Баловсяк С. – наукове керівництво і редагування; **Стець С.** – розробка методики та програмного забезпечення для попередньої обробки зображень перед їх детектуванням).
3. Баловсяк С., **Стець С.** Автоматизоване створення спеціалізованого датасету для зображень автомобілів. *Herald of Khmelnytskyi National University. Technical Sciences*. 2025. Т. 359 (6.2). С. 278-285. https://doi.org/10.31891/2307-5732-2025-359-111. (Баловсяк С. – наукове керівництво і редагування; **Стець С.** – розробка методики та програмного забезпечення для автоматичного створення набору даних та донавчання згорткової нейронної мережі).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

4. Баловсяк С.В., **Стець С.Ю.** Аналіз сучасних методів розпізнавання зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XI Міжнародної науково-практичної конференції (ПІКТ – 2022), м. Чернівці, 10–13 листопада 2022 р. Чернівці: Черн. нац. ун-т, 2022. С. 77-80. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – аналіз сучасних методів розпізнавання зображень автомобілів).
5. Баловсяк С.В., **Стець С.Ю.** Використання модуля Inception для підвищення точності розпізнавання зображень у згорткових нейронних мережах. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення*: матеріали міжнародної наукової інтернет-конференції, 8-9 червня 2023 р. Вип. 78. Тернопіль, 2023. С. 30-32. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмного забезпечення з використанням модуля Inception).
6. **Стець С.Ю.** Підвищення точності нейромережного розпізнавання зображень за допомогою модуля Inception. *Проблеми інформатики та комп'ютерної техніки*: праці XII Міжнародної науково-практичної конференції (ПІКТ – 2023), м. Чернівці, 10-12 листопада 2023 р. Чернівці: Черн. нац. ун-т, 2023. С. 61-63.
7. Баловсяк С.В., **Стець С.Ю.** Детектування зображень пішохідних переходів за допомогою нейронних мереж YOLO. *Проблеми інформатики та комп'ютерної техніки*: праці XIII Міжнародної науково-практичної конференції (ПІКТ – 2024), м. Чернівці, 1–3 листопада 2024 р. Чернівці: Черн. нац. ун-т, 2024. С. 49-51. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмного забезпечення для детектування зображень пішохідних переходів).
8. Баловсяк С.В., **Стець С.Ю.** Попередня обробка зображень об'єктів перед їх детектуванням засобами нейронної мережі YOLO. *Фізико-технологічні проблеми передачі, обробки та зберігання інформації в інфокомунікаційних системах*:

матеріали X Міжнародної науково-практичної конференції (ПРЕДТ-2025). 15-17 травня 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 181-182. (Баловсяк С.В. – наукове керівництво і редагування; **Стець С.Ю.** – розробка програмних засобів для попередньої обробки зображень).

9. **Стець С.Ю.** Донавчання нейронної мережі YOLO за допомогою автоматизовано створеного датасету зображень автомобілів. *Проблеми інформатики та комп'ютерної техніки*: праці XIV Міжнародної науково-практичної конференції (ПІКТ – 2025), м. Чернівці, 13-15 листопада 2025 р. Чернівці: Черн. нац. ун-т, 2025. С. 64-66.

Публікації, які додатково відображають наукові результати дисертації:

10. Balovsyak S., Kroitor O., Odaiska Kh., Salem A.B.M., **Stets S.** Car image recognition using convolutional neural network with efficient met architecture. *CEUR Workshop Proceeding*. 2024. Vol. 3675: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, IntelITSIS 2024. P. 182-195 (Scopus). (Balovsyak S. – наукове керівництво, Kroitor O. – аналіз наборів даних, Odaiska Kh. – редагування, Salem A.B.M. – аналіз архітектур згорткових нейронних мереж, **Stets S.** – розробка моделі згорткової нейронної мережі з архітектурою EfficientNet, призначеної для розпізнавання зображень автомобілів).

Додаток Б

Відомості про апробацію результатів дисертації

1. XI Міжнародна науково-практична конференція «Проблеми інформатики та комп'ютерної техніки» (ПІКТ – 2022), Чернівці, 10–13 листопада 2022 р.; форма участі – очна.
2. Міжнародна наукова інтернет-конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення». Вип. 78, Тернопіль, 8-9 червня 2023 р.; форма участі – заочна.
3. XII Міжнародна науково-практична конференція «Проблеми інформатики та комп'ютерної техніки» (ПІКТ – 2023), Чернівці, 10–12 листопада 2023 р.; форма участі – очна.
4. XIII Міжнародна науково-практична конференція «Проблеми інформатики та комп'ютерної техніки» (ПІКТ – 2024), Чернівці, 1–3 листопада 2024 р.; форма участі – очна.
5. X Міжнародна науково-практична конференція «Фізико-технологічні проблеми передачі, обробки та зберігання інформації в інфокомунікаційних системах» (ПРЕДТ-2025), Чернівці, 15-17 травня 2025 р.; форма участі – очна.
6. XIV Міжнародна науково-практична конференція «Проблеми інформатики та комп'ютерної техніки» (ПІКТ – 2025), Чернівці, 13–15 листопада 2025 р.; форма участі – очна.

Додаток В

Акти впровадження результатів дисертаційної роботи

Україна, ТОВ «ДВА ВІДРА» 58029,
м. Чернівці, просп. Незалежності, 111
Тел./факс: (050) 43-43-111
E-mail: voda@mc.cv.ua
https://www.dvavidra.ua/

«ЗАТВЕРДЖУЮ»
Директор
ТОВ «ДВА ВІДРА»
_____ Б. З. Мариній
«__» _____ 2025 р.

АКТ

впровадження результатів дисертаційної роботи Стеця Сергія Юрійовича «Підвищення точності та швидкодії детектування зображень автомобілів засобами згорткової нейронної мережі YOLO», що представлена до захисту на здобуття наукового ступеня доктора філософії за спеціальністю «Інженерія програмного забезпечення», у виробничому процесі ТОВ «ДВА ВІДРА»

Комісія у складі Маринія Богдана Зіновійовича та Чибуряк Людмили Петрівни, склала акт про те, що результати дисертаційної роботи аспіранта Чернівецького національного університету імені Юрія Федьковича Стеця С. Ю. впроваджені для автоматичної детекції автомобілів компанії та паркомісць в системах відеоспостереження, які використовуються в ТОВ «ДВА ВІДРА».

Розроблені Стецем С. Ю. програмні засоби дозволили автоматично визначати стан паркомісць та положення автомобілів компанії на автомобільній стоянці за їх зображеннями. Це дало змогу фіксувати точний час прибуття та відбуття автомобілів компанії, а також присутність сторонніх автомобілів у зоні відеоспостереження.

Даний акт не є підставою для отримання будь-якої винагороди й складений для представлення в разову спеціалізовану вчену раду в зв'язку із захистом дисертації Стеця С. Ю.

Члени комісії:

Директор ТОВ «ДВА ВІДРА» _____ Мариній Б. З.

Заступник директора ТОВ «ДВА ВІДРА» _____ Чибуряк Л.П.

Україна, ТОВ "ТРК А.С.С."

58029, м. Чернівці, просп. Незалежності, 111

Тел./факс: (0372) 515-445

E-mail: news@fm.cv.ua

https://acc.cv.ua/

«ЗАТВЕРДЖУЮ»

Директор

ТОВ "ТРК А.С.С."

«__» _____ 2025 р.

АКТ

впровадження результатів дисертаційної роботи Стеця Сергія Юрійовича «Підвищення точності та швидкодії детектування зображень автомобілів засобами згорткової нейронної мережі YOLO», що представлена до захисту на здобуття наукового ступеня доктора філософії за спеціальністю «Інженерія програмного забезпечення», у виробничому процесі ТОВ «ТРК А.С.С.»

Комісія у складі : Данилюк О.В, Скупова С.О., склала акт про те, що результати дисертаційної роботи аспіранта Чернівецького національного університету імені Юрія Федьковича Стеця С. Ю. впроваджені для автоматичної детекції автомобілів компанії та паркомісць в системах відеоспостереження, які використовуються в ТОВ "ТРК А.С.С.".

Розроблена Стецем С. Ю. програма дозволила замінити ручний облік автомобілів на автоматичний. Система самостійно фіксує кількість машин, що в'їжджають та виїжджають, що дозволило отримати точну статистику відвідуваності та пікових годин навантаження без залучення персоналу. Завдяки вбудованим алгоритмам покращення зображення програма стабільно розпізнає автомобілі навіть у складних умовах, що забезпечило цілодобовий безперебійний контроль території. Даний акт не є підставою для отримання будь-якої винагороди й складений для представлення в спеціалізовану вчену раду в зв'язку із захистом дисертації Стеця С. Ю.

Директор ТОВ "ТРК А.С.С." Данилюк

Головний бухгалтер ТОВ "ТРК А.С.С." Скупова С.О.

"ЗАТВЕРДЖУЮ"
Директор Навчально-наукового
інституту фізико-технічних та
комп'ютерних наук
Чернівецького національного
університету
імені Юрія Федьковича,
кандидат технічних наук, доцент
_____ Петро ШПАТАР
"23" 12 2025 р.

АКТ

про впровадження у навчальний процес результатів дисертаційної роботи Стеця Сергія Юрійовича

«Підвищення точності та швидкодії детектування зображень автомобілів засобами згорткової нейронної мережі YOLO», що представлена на здобуття наукового ступеня доктора філософії за спеціальністю 121 «Інженерія програмного забезпечення»

Комісія Чернівецького національного університету імені Юрія Федьковича в складі: заступника директора Навчально-наукового інституту фізико-технічних та комп'ютерних наук з наукової роботи, доктора фізико-математичних наук, професора Дуболазова О.В., завідувача кафедри комп'ютерних систем та мереж, кандидата фізико-математичних наук, доцента Воробця Г. І. склали цей акт в тому, що результати дисертаційної роботи здобувача наукового ступеня доктора філософії Стеця С.Ю. впроваджені у навчальний процес на кафедрі комп'ютерних систем та мереж, зокрема, в дисциплінах «Методи цифрової обробки зображень», «Комп'ютерні системи штучного інтелекту» та «Системи комп'ютерного зору».

Під час викладання дисципліни «Методи цифрової обробки зображень» використано такі результати досліджень автора: методика та програмні засоби для попередньої обробки зображень, які подаються на входи згорткової нейронної мережі з архітектурою YOLO (описані у розділі

«2 Згорткова нейронна мережа для детектування зображень автомобілів з модулем попередньої обробки зображень») використовуються при вивченні способів підвищення візуальної якості зображень шляхом еквалізації їх гістограм та підвищення локального контрасту.

Під час викладання дисципліни «Комп'ютерні системи штучного інтелекту» використано такі результати досліджень автора: програмні засоби для донавчання нейронних мереж YOLO та для детектування з їх допомогою зображень автомобілів та інших учасників дорожнього руху (описані у розділі «2 Згорткова нейронна мережа для детектування зображень автомобілів з модулем попередньої обробки зображень») застосовуються при вивчені засобів детектування зображень об'єктів.

При викладанні дисципліни «Системи комп'ютерного зору» використано такі результати досліджень автора: методика та програмні засоби для донавчання різних версій згорткових нейронних мереж YOLO з використанням наборів даних (датасетів), створених у ручному режимі, та аналізу результатів навчання (описані у розділі «3 Навчання згорткових нейронних мереж YOLO для детектування зображень автомобілів із ручним формуванням датасету») використовуються при вивчені методів та засобів виявлення (детектування) зображень об'єктів.

Використання зазначених результатів дозволило підвищити якість навчального процесу із згаданих дисциплін.

Акт складений для представлення в спеціалізовану вчену раду в зв'язку із захистом дисертації Стеця С.Ю.

Заступник директора ННІФТКН з
наукової роботи, д.ф.-м.н., професор Олександр ДУБОЛАЗОВ

Завідувач кафедри КСМ,
к.ф.-м.н., доцент Георгій ВОРОБЕЦЬ

Додаток Г

Лістинг коду комп’ютерної програми "YOLOContrast25" (фрагмент)

```
# Програма (Python) призначена для детектування об’єктів на зображеннях згортковою
# нейронною мережею із використанням попередньої обробки зображень
import numpy as np   # імпорт бібліотеки для математичних обчислень
import matplotlib.pyplot as plt  # імпорт бібліотеки для побудови графіків
import cv2 # імпорт бібліотеки opencv
from ultralytics import YOLO # імпорт моделей нейронної мережі YOLO

def compute_iou(box1, box2):
    #  Обчислення IoU між двома рамками у форматі (x1, y1, x2, y2)
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])
    inter_area = max(0, x2 - x1) * max(0, y2 - y1)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
    iou = inter_area / (box1_area + box2_area - inter_area)
    return iou

def calc_mTrue_Pred_IOU(mIOU,mIOU2,mTrue_Pred):
  # Обчислення mTrue_Pred - Номеру прогнозованого вікна /рамки/ Predicted,
  # який відповідає вікну True
  mTrue_Pred=mTrue_Pred-1 # -1 - для рамки True не існує рамки Predicted
  mIOU2=mIOU2+mIOU
  for ntrue in range(0,QTr_Box):
    max_index = np.unravel_index(np.argmax(mIOU2), mIOU2.shape)
    nt_max=max_index[0]  # номер максимуму для номеру рамки True
    np_max=max_index[1]  # номер максимуму для номеру рамки Predicted
    if mIOU2[nt_max,np_max]>0:
      mTrue_Pred[nt_max]=np_max
      # номеру рамки nt_max /True/  відповідає np_max /Predicted/
    mIOU2[nt_max,0:QPr_Box]=-1
    mIOU2[0:QTr_Box,np_max]=-1
  return mTrue_Pred

def calc_par_detect(QPr_Box1,ni,niA,mIOU1,mTrue_Pred1,conf_A1,numE):
  # обчислення параметрів детектування зображення
  # numE=0 - origin image (початкове зображення)
  # numE=1 - enchance image (змінене зображення)
```

```
# обчислення середніх значень iou: average_iou, average_conf
average_iou=0
average_conf=0
TP=0
mIOU_True_Pred=np.zeros(shape=(QTr_Box),dtype=np.float64)
mIOU_True_Pred=mIOU_True_Pred-1
mconf_A=np.zeros(shape=(QTr_Box),dtype=np.float64)
mconf_A=mconf_A-1
mTP_Pred=np.zeros(shape=(QPr_Box1),dtype=np.int64)
mFP_Pred=np.zeros(shape=(QPr_Box1),dtype=np.int64)

for ntrue in range(0,QTr_Box): # QTr_Box - Quantity True Box
  npred=mTrue_Pred1[ntrue] # Номер вікна Pred, який відповідає вікну True
  if npred>=0: # якщо існує таке вікно
    if mIOU1[ntrue,npred]>0.5:
      average_iou=average_iou+mIOU1[ntrue,npred]
      TP=TP+1 #         TP (True Positives) – правильно знайдені об'єкти
      mTP_Pred[npred]=1  # 0 - FP, FalsePositive; 1 - TP, TruePositive
      mIOU_True_Pred[ntrue]=mIOU1[ntrue,npred]
      mconf_A[ntrue]=conf_A1[npred]
      average_conf=average_conf+conf_A1[npred]
if TP>0:
  average_iou=average_iou/TP
  average_conf=average_conf/TP  #  Confidence /Probability/ of class

mFP_Pred=1-mTP_Pred # FP, FalsePositive

# Precision= TP/(TP+FP)
FP=QPr_Box1-TP # FP (False Positives) –неправильно знайдені об'єкти (хибні)
Precision=0
if (TP+FP)>0:
  Precision= TP/(TP+FP)

# Recall = TP/(TP+FN)
FN=QTr_Box-TP # FN (False Negatives) – пропущені об'єкти
Recall = 0
if (TP+FN)>0:
  Recall = TP/(TP+FN)
mTP[ni,numE]=TP; mFP[ni,numE]=FP; mFN[ni,numE]=FN

# F1-score – баланс між Precision і Recall:
F1=0
if (Precision+Recall)>0:
  F1=2*(Precision*Recall)/(Precision+Recall)
```

```
  mPrecision[ni,numE]=Precision
  mRecall[ni,numE]=Recall
  mF1[ni,numE]=F1
  return mIOUAi, mconf_Ai, mPrecision, mRecall, mF1, mAP

def img_hist_C(Y,hfn_y,hZ_y,Qh,YhC):
  # Histogram Center,
  # вирівнювання та центрування гістограми для каналу яскравості YhC
  zC1=0
  sum_hfn=np.sum(hfn_y)
  for b in range(0,Qh):
    zC1=zC1+hfn_y[b]*hZ_y[b]
  if sum_hfn>0:
    zC1=zC1/sum_hfn
  print('Центр ваги гістограми zC1= ',zC1)

  zC2=0
  sum1=0
  for b in range(0,Qh):
    sum1=sum1+hfn_y[b]
    if sum1>0.5:
      b1=b
      if b1<0:
        b1=0
      zC2=hZ_y[b1]
      break
  print('Точка рівної площі (лівої і правої частини гістограми) zC2= ',zC2)
  zC=(zC1+zC2)/2  #   zC  - Центр гістограми

  mask = Y <= zC
  YhC[mask] = Y[mask] * (127.5 / zC) # для Y <= zC
  YhC[~mask] = 127.5+127.5*(Y[~mask] -zC) / (255.0 - zC) # для Y>zC
  return img_hCy

def image_enchance(image_BGR,mode_enchance):
  # Покращення зображення перед детектуванням у режимах (mode_enchance):
  if mode_enchance==1: # Еквалізація гістограми зображення
    im_equ1[:,:,0] = cv2.equalizeHist(image_BGR[:,:,0]) # змінене зображення
    im_equ1[:,:,1] = cv2.equalizeHist(image_BGR[:,:,1])
    im_equ1[:,:,2] = cv2.equalizeHist(image_BGR[:,:,2])

  # 2. Підвищення локального контрасту
  # Використання Adaptive Histogram Equalization (CLAHE)
  if mode_enchance==2:
```

```
    clahe = cv2.createCLAHE(clipLimit=0.1, tileGridSize=(8,8))
    im_equ1[:,:,0] = clahe.apply(image_BGR[:,:,0])
    im_equ1[:,:,1] = clahe.apply(image_BGR[:,:,1])
    im_equ1[:,:,2] = clahe.apply(image_BGR[:,:,2])

  # 3- Histogram Center
  if mode_enchance==3: # Histogram Center
    # вирівнювання та центрування гістограми
    img_255=cv2.cvtColor(image_BGR, cv2.COLOR_BGR2RGB)
    img_u8 = img_255.astype('uint8')
    ycrcb = cv2.cvtColor(img_255, cv2.COLOR_RGB2YCrCb)
    Y=ycrcb[:,:,0]
    hfn_y,hZ_y=fn_histogram(ycrcb[:,:,0],Qh)
    hfn=hfn_y
    hZ=hZ_y
    YhC=np.zeros(shape=(M,N),dtype=np.uint8)
    YhC=img_hist_C(Y,hfn_y,hZ_y,Qh, YhC) # YhC - змінений канал яскравості
    ycrcb_2=ycrcb.copy()
    ycrcb_2[:,:,0]=0
    ycrcb_2[:,:,0]=ycrcb_2[:,:,0]+ YhC
    img_E = cv2.cvtColor(ycrcb_2, cv2.COLOR_YCrCb2RGB)
    img_E= np.clip(img_E,0,255)
    img_E=img_E.astype('uint8')
    im_equ1 = cv2.cvtColor(img_E, cv2.COLOR_RGB2BGR)
  return im_equ1

# головна програма ------------------------------------

# Завантаження моделі YOLOv8
model = YOLO("yolov8m.pt")
# Завантаження валідаційного набору COCO
coco_path = "instances_val2017.json" # Шлях до COCO валідаційного набору

# Завантаження COCO анотації
coco = COCO(coco_path)
# Отримання ID для категорії «Автомобіль» (car)
category_id = 3 # 3 - car
image_ids = coco.getImgIds(catIds=[category_id])
id_s=category_id-1

# Отримання інформації про зображення
images = coco.loadImgs(image_ids)
QIm=len(images)  # кількість зображень категорії «Автомобіль» (car)
class_name=result.names[id_s]
```

```
# Детектування в YOLO початкового зображення та зображення після
# покращення (еквалізація гістограми та ін.), порівняння результатів

QImR=100  # QImR - кількість тестових зображень
id_s=2 # 2- car, детектування зображень автомобілів
ni=-1

for img_info in images[0:QImR]:
  ni=ni+1 # номер зображення (відносний, починаючи з 0)
  file_name=img_info["file_name"]
  url=img_info["coco_url"]
  resp = urllib.request.urlopen(url)
  image_array = np.asarray(bytearray(resp.read()), dtype=np.uint8)
  image_BGR = cv2.imdecode(image_array, cv2.IMREAD_COLOR)  # зображення
  im_rgb = cv2.cvtColor(image_BGR, cv2.COLOR_BGR2RGB)

  # Детекція зображення image_BGR -------------------------
  results = model.predict(image_BGR)
  result = results[0] # result for image #0
  Q_obj=len(result.boxes) # кількість об’єктів

  # Отримання прогнозованих рамок об’єктів з YOLO (початкових зображень)
  pred_boxes = result.boxes.xyxy.cpu().numpy()  # координати (x1, y1, x2, y2)
  pred_boxes_A=[] # координати (x1, y1, x2, y2) автомобілів
  conf_A=[]  # Confidence /Probability/ of class

  for nb in range(0,Q_obj):  # nb – номер об’єкта
    box = result.boxes[nb]
    class_id = int(box.cls[0].item())
    if class_id==id_s:
      class_name=result.names[class_id]
      cords0 = box.xyxy[0].tolist()
      cords = [round(x,2) for x in cords0]
      pred_boxes_A.append(cords) # координати (x1, y1, x2, y2) Автомобілів

  # Зміна /покращення/ зображення image_BGR перед детектуванням
  mode_enchance=2 # 1 - equalizeHis, 2- CLAHE; 3- Histogram Center
  # CLAHE - Adaptive Histogram Equalization
  im_equ = image_enchance(image_BGR,mode_enchance)
  im_rgbE = cv2.cvtColor(im_equ, cv2.COLOR_BGR2RGB)

  # Детектування зміненого зображення im_equ ------------------------------
  resultsE = model.predict(im_equ)
```

```
resultE = resultsE[0] # result fot image #0
Q_objE=len(resultE.boxes)

# Отримання прогнозованих рамок об’єктів з YOLO (змінених зображень)
pred_boxesE = resultE.boxes.xyxy.cpu().numpy()  # (x1, y1, x2, y2)
pred_boxes_AE=[] # координати (x1, y1, x2, y2) Автомобілів   conf_AE=[]
for nb in range(0,Q_objE):  # nb - номер об’єкта
  box = resultE.boxes[nb]
  class_id = int(box.cls[0].item())
  if class_id==id_s:
    class_name=result.names[class_id]
    cords0 = box.xyxy[0].tolist()
    cords = [round(x,2) for x in cords0]
    pred_boxes_AE.append(cords)  # координати (x1, y1, x2, y2) Автомобілів

# Отримання реальних /true, правильних/ рамок з COCO
ann_ids = coco.getAnnIds(imgIds=[images[niA]["id"]], catIds=[id_s+1])
annotations = coco.loadAnns(ann_ids)
true_boxes = [ann["bbox"] for ann in annotations]  # [x, y, width, height]

# Конвертація COCO формату в (x1, y1, x2, y2)
true_boxes = [[x, y, x + w, y + h] for x, y, w, h in true_boxes]
QTr_Box=len(true_boxes)

# Обчислення IoU (Intersection over Union)
mIOU=np.zeros(shape=(QTr_Box,QPr_Box),dtype=np.float64)
for ntrue in range(0,QTr_Box):  # номер реальної рамки
  for npred in range(0,QPr_Box): # номер детектованої рамки
    btrue=true_boxes[ntrue]  # координати реальної рамки
    bpred=pred_boxes_A[npred] # координати детектованої рамки
    mIOU[ntrue,npred] = compute_iou(bpred, btrue)

mTrue_Pred = calc_mTrue_Pred_IOU(mIOU,mIOU2,mTrue_Pred)

# обчислення параметрів детектування зображення:
numE=0 # 0 - origin image (початкове зображення)
mIOUAi,mconf_Ai,mPrecision,mRecall,mF1,mAP=
calc_par_detect(QPr_Box,ni,niA,mIOU,mTrue_Pred,conf_A,numE)

numE=1 # 1 - змінене зображення
mIOUAi,mconf_Ai,mPrecision,mRecall,mF1,mAP=
calc_par_detect(QPr_BoxE,ni,niA,mIOUE,mTrue_PredE,conf_AE,numE)
```

Додаток Д

Лістинг коду комп’ютерної програми "FineTunedYOLOCarDetection"

(фрагмент)

```
# Програма (Python) призначена для порівняльного аналізу ЗНМ
# з архітектурами YOLOv8 та YOLOv11 до та після
# донавчання на спеціалізованому датасеті зображень автомобілів
# із автоматичним моніторингом часових витрат
# та метрик точності детектування.

pip install ultralytics

"""Installing necessary libraries and mounting google drive to upload and download data such as model results, plots, etc"""

from ultralytics import YOLO
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
import yaml
from google.colab import files

# Mount Google Drive for saving results
from google.colab import drive
drive.mount('/content/drive')

"""## Описуємо шляхи збереження моделі, графіків та результатів"""

base_path = '/content/drive/MyDrive/yolo8_vs_yolo11_new'
dataset_path = os.path.join(base_path, 'dataset')  # Roboflow exported dataset
yolo8_model_path = os.path.join(base_path, 'yolov8m.pt')
yolo11_model_path = os.path.join(base_path, 'yolov11m.pt')
results_path = os.path.join(base_path, 'results')
plot_output_path = os.path.join(base_path, 'plots')
models_save_path = os.path.join(base_path, 'models')

# Ensure output directories exist
os.makedirs(plot_output_path, exist_ok=True)
os.makedirs(models_save_path, exist_ok=True)

"""## Описуємо скільки зображень і класів знаходяться в датасеті"""

dataset_yaml = f"{dataset_path}/data.yaml"
```

```
with open(dataset_yaml, 'r') as file:
    dataset_info = yaml.safe_load(file)

def count_images(path):
    try:
        full_path = os.path.join(os.path.dirname(dataset_yaml), path)
        return len(os.listdir(full_path)) if os.path.exists(full_path) else 0
    except Exception as e:
        print(f"Error counting images in {path}: {e}")
        return 0

train_images_count = count_images('train/images')
valid_images_count = count_images('valid/images')
test_images_count = count_images('test/images')

dataset_count = train_images_count + valid_images_count + test_images_count

print("Dataset Summary:")
print(f"Train images: {train_images_count}")
print(f"Validation images: {valid_images_count}")
print(f"Test images: {test_images_count}")

# Additional metrics
categories = dataset_info.get('names', [])
print(f"Number of categories: {len(categories)}")
print("Categories:")
for idx, category in enumerate(categories):
    print(f"  {idx}: {category}")

"""## Тренуємо yolo8 та yolo11"""

# Load YOLO models
yolo8 = YOLO(yolo8_model_path)

yolo11 = YOLO(yolo11_model_path)
hyperparams = {
        'amp': True,  # Use mixed precision for faster training
        'patience': 20,  # Early stopping patience
        'lr0': 0.01,  # Initial learning rate
        'lrf': 0.01,  # Final learning rate as a fraction of initial
        'momentum': 0.937,
        'weight_decay': 0.0005,
        'warmup_epochs': 5,  # Extended warmup for stable training
        'warmup_momentum': 0.8,
        'warmup_bias_lr': 0.1,
        'box': 7.5,  # Box loss gain - increased for better localization
        'cls': 0.5,  # cls loss gain - reduced since we focus on fewer classes
```

```
        'dfl': 1.5,  # Distribution focal loss gain
        'hsv_h': 0.015,  # HSV augmentation parameters
        'hsv_s': 0.2,  # Increased saturation variety
        'hsv_v': 0.1,
        'translate': 0.1,
        'scale': 0.3,  # Increased scale for better size variation
        'shear': 0.1,
        'perspective': 0.0001,  # Minimal perspective change
        'flipud': 0.0,  # No vertical flips for cars
        'mosaic': 0.5,  # Moderate mosaic augmentation
        'copy_paste': 0.0  # No copy-paste for cars

}
# Fine-tune YOLO8m
print("Fine-tuning YOLO8m model...")
start_time = time.time()
results_yolo8m = yolo8m.train(
    data=f'{dataset_path}/data.yaml',
    epochs=75,
    imgsz=640,
    batch=16,
    name='yolo8m_finetuned',
    project=results_path,
    **hyperparams
)
end_time = time.time()
training_time_yolo8m = end_time - start_time
print(training_time_yolo8m)
# Save trained models
yolo8m.save(os.path.join(models_save_path, "yolo8m_trained.pt"))
# Fine-tune YOLO11m
print("Fine-tuning YOLO11m model...")
start_time = time.time()
results_yolo11m = yolo11m.train(
    data=f'{dataset_path}/data.yaml',
    epochs=75,
    imgsz=640,
    batch=16,
    name='yolo11m_finetuned',
    project=results_path,
    **hyperparams
)
end_time = time.time()
training_time_yolo11m = end_time - start_time
print(training_time_yolo11m)
# Save trained models
yolo11m.save(os.path.join(models_save_path, "yolo11m_trained.pt"))
```

Додаток Е

Лістинг коду комп’ютерної програми "YOLOAutoDatasetCreation" (фрагмент)

```
# Програму розміщено: https://github.com/SerhiiStets/PhD
# Програма (Python) призначена для автоматизованого формування
# датасету зображень ЗНМ YOLO середнього розміру (модель-вчитель YOLOv8m),
# донавчання ЗНМ YOLO нано-розміру (модель-учень YOLOv8n),
# попередньої обробки зображень, детектування зображень транспортних засобів
# навченою мережею YOLO нано-розміру

# Модуль #1 програми "YOLOAutoDatasetCreation" призначений для попередньої обробки
# та детектування зображень ЗНМ YOLO, в якому попередня обробка зображень виконується
# шляхом підвищення їх контрасту; з дослідницькою метою виконується візуалізація гістограм
# розподілу яскравості зображень; виконується статистична оцінка точності детектування
# транспортних засобів; виявлення об’єктів можливе як на окремих зображеннях, так й на відео
# у режимі реального часу.
"""
API routes for CLAHE (Contrast Limited Adaptive Histogram Equalization) comparison
"""

from fastapi import APIRouter, File, UploadFile, HTTPException
from pydantic import BaseModel
from typing import List
import io
import base64
import logging
from PIL import Image
import cv2
import numpy as np
import matplotlib

logger = logging.getLogger(__name__)
router = APIRouter(prefix="/api/clahe", tags=["clahe"])

class BoundingBox(BaseModel):
    x1: float
    y1: float
    x2: float
    y2: float
    confidence: float
    class_name: str

class CLAHEDetectionResult(BaseModel):
    image_base64: str  # Image with detection boxes drawn
    raw_image_base64: str  # Raw image without boxes (to show CLAHE effect)
```

```
    histogram_base64: str
    detections: List[BoundingBox]
    detection_count: int
    avg_confidence: float

class CLAHEComparisonResult(BaseModel):
    original: CLAHEDetectionResult
    clahe_enhanced: CLAHEDetectionResult
    improvement_metrics: dict

# Global YOLO model (lazy loaded)
yolo_model = None

def get_yolo_model():
    """Lazy load YOLO model"""
    global yolo_model
    if yolo_model is None:
        from ultralytics import YOLO

        yolo_model = YOLO("yolov8m.pt")
        logger.info("Loaded YOLOv8m model for CLAHE comparison")
    return yolo_model

def apply_clahe(
    image_bgr: np.ndarray, clip_limit: float = 0.1, tile_grid_size: tuple = (8, 8)
) -> np.ndarray:
    """
    Застосування алгоритму CLAHE для покращення локального контрасту зображення.
    Метод адаптивної гістограми дозволяє вирівняти освітленість, зберігаючи деталі в тінях.
    """
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_grid_size)
    enhanced = np.zeros_like(image_bgr)
    # Apply CLAHE to each channel
    enhanced[:, :, 0] = clahe.apply(image_bgr[:, :, 0])
    enhanced[:, :, 1] = clahe.apply(image_bgr[:, :, 1])
    enhanced[:, :, 2] = clahe.apply(image_bgr[:, :, 2])
    return enhanced

def compute_histogram(image_bgr: np.ndarray, num_bins: int = 32) -> tuple:
    """
    Основний конвеєр обробки: детекція YOLO, генерація гістограми та анотування кадру.
    """
    gray = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2GRAY)
    hist, bin_edges = np.histogram(gray, bins=num_bins)

    # Calculate bin centers
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
```

```
    # Normalize histogram
    hist = hist / np.sum(hist)
    return hist.tolist(), bin_centers.tolist()

def create_histogram_image(hist: list, bin_centers: list, title: str) -> str:
    """
    Create histogram visualization as base64 image
    """
    matplotlib.use("Agg")
    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(figsize=(4, 3))

    # Calculate bar width
    if len(bin_centers) > 1:
        width = (bin_centers[-1] - bin_centers[0]) / (len(bin_centers) + 1)
    else:
        width = 1

    ax.bar(bin_centers, hist, width=width, color="#9999ff", edgecolor="blue")
    ax.set_title(f"Histogram: {title}", fontsize=10)
    ax.set_xlabel("Intensity")
    ax.set_ylabel("Frequency")

    # Save to buffer
    buf = io.BytesIO()
    plt.tight_layout()
    plt.savefig(buf, format="png", dpi=100)
    plt.close(fig)
    buf.seek(0)
    img_base64 = base64.b64encode(buf.read()).decode()
    return img_base64

def draw_detections(image_bgr: np.ndarray, detections: List[BoundingBox]) -> np.ndarray:
    """Draw bounding boxes on image"""
    img = image_bgr.copy()

    for i, det in enumerate(detections):
        # Draw rectangle in red
        cv2.rectangle(
            img,
            (int(det.x1), int(det.y1)),
            (int(det.x2), int(det.y2)),
            (0, 0, 255),  # Red in BGR
            2,
        )
        # Draw detection number
        text = str(i)
```

```
        cv2.putText(
            img,
            text,
            (int((det.x1 + det.x2) / 2), int(det.y1) - 4),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.6,
            (0, 0, 255),  # Red
            2,
        )
    return img

async def process_image(
    image_bgr: np.ndarray, is_enhanced: bool = False
) -> CLAHEDetectionResult:
    """
    Process image: run YOLO detection, create histogram, draw annotations
    """
    model = get_yolo_model()
    # Convert BGR to RGB for YOLO
    image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    # Run YOLO detection
    results = model(image_rgb)
    # Parse detections (filter for cars - class_id=2)
    detections = []
    for result in results:
        for box in result.boxes:
            class_id = int(box.cls[0].cpu().numpy())

            # Filter for cars only (class_id=2 in COCO)
            if class_id == 2:
                x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                confidence = float(box.conf[0].cpu().numpy())
                class_name = model.names[class_id]

                detections.append(
                    BoundingBox(
                        x1=float(x1),
                        y1=float(y1),
                        x2=float(x2),
                        y2=float(y2),
                        confidence=confidence,
                        class_name=class_name,
                    )
                )

    # Convert raw image to base64 (without boxes - to show CLAHE effect)
    raw_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    raw_pil = Image.fromarray(raw_rgb)
```

```
    raw_buffered = io.BytesIO()
    raw_pil.save(raw_buffered, format="JPEG", quality=95)
    raw_img_base64 = base64.b64encode(raw_buffered.getvalue()).decode()

    # Draw detections on image
    annotated_bgr = draw_detections(image_bgr, detections)

    # Convert annotated image to base64
    annotated_rgb = cv2.cvtColor(annotated_bgr, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(annotated_rgb)
    buffered = io.BytesIO()
    pil_img.save(buffered, format="JPEG", quality=95)
    img_base64 = base64.b64encode(buffered.getvalue()).decode()

    # Compute histogram
    hist, bin_centers = compute_histogram(image_bgr)
    title = "Enhanced" if is_enhanced else "Original"
    histogram_base64 = create_histogram_image(hist, bin_centers, title)

    # Calculate average confidence
    avg_confidence = (
        sum(d.confidence for d in detections) / len(detections) if detections else 0.0
    )
    return CLAHEDetectionResult(
        image_base64=img_base64,
        raw_image_base64=raw_img_base64,
        histogram_base64=histogram_base64,
        detections=detections,
        detection_count=len(detections),
        avg_confidence=round(avg_confidence, 3),
    )

@router.post("/analyze", response_model=CLAHEComparisonResult)
async def analyze_clahe(file: UploadFile = File(...)):
    """
    Analyze image with and without CLAHE enhancement
    """
    try:
        # Read image
        contents = await file.read()
        image = Image.open(io.BytesIO(contents))

        # Convert to BGR for OpenCV
        if image.mode != "RGB":
            image = image.convert("RGB")
        image_bgr = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

        # Process original image
```

```
        original_result = await process_image(image_bgr, is_enhanced=False)

        # Apply CLAHE enhancement
        enhanced_bgr = apply_clahe(image_bgr, clip_limit=0.1, tile_grid_size=(8, 8))
        # Process enhanced image
        enhanced_result = await process_image(enhanced_bgr, is_enhanced=True)
        # Calculate improvement metrics
        detection_diff = (
            enhanced_result.detection_count - original_result.detection_count
        )
        confidence_diff = (
            enhanced_result.avg_confidence - original_result.avg_confidence
        )
        improvement_metrics = {
            "detection_difference": detection_diff,
            "confidence_difference": round(confidence_diff, 3),
            "detection_improvement_pct": round(
                (detection_diff / max(original_result.detection_count, 1)) * 100, 1
            )
            if original_result.detection_count > 0
            else 0,
            "confidence_improvement_pct": round(
                (confidence_diff / max(original_result.avg_confidence, 0.001)) * 100, 1
            )
            if original_result.avg_confidence > 0
            else 0,
        }
        return CLAHEComparisonResult(
            original=original_result,
            clahe_enhanced=enhanced_result,
            improvement_metrics=improvement_metrics,
        )
    except Exception as e:
        logger.error(f"Error in CLAHE analysis: {e}")
        raise HTTPException(status_code=500, detail=f"CLAHE analysis failed: {str(e)}")

@router.post("/analyze-from-camera", response_model=CLAHEComparisonResult)
async def analyze_from_camera():
    """
    Capture image from webcam and analyze with CLAHE
    """
    try:
        # Capture from webcam
        cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        # Warm-up
        for _ in range(5):
```

```
            cap.read()
        # Capture frame
        ret, frame = cap.read()
        cap.release()

        if not ret:
            raise HTTPException(status_code=500, detail="Failed to capture image")

        # Process original image
        original_result = await process_image(frame, is_enhanced=False)

        # Apply CLAHE enhancement
        enhanced_bgr = apply_clahe(frame, clip_limit=0.1, tile_grid_size=(8, 8))

        # Process enhanced image
        enhanced_result = await process_image(enhanced_bgr, is_enhanced=True)

        # Calculate improvement metrics
        detection_diff = (
            enhanced_result.detection_count - original_result.detection_count
        )
        confidence_diff = (
            enhanced_result.avg_confidence - original_result.avg_confidence
        )
        improvement_metrics = {
            "detection_difference": detection_diff,
            "confidence_difference": round(confidence_diff, 3),
            "detection_improvement_pct": round(
                (detection_diff / max(original_result.detection_count, 1)) * 100, 1
            )
            if original_result.detection_count > 0
            else 0,
            "confidence_improvement_pct": round(
                (confidence_diff / max(original_result.avg_confidence, 0.001)) * 100, 1
            )
            if original_result.avg_confidence > 0
            else 0,
        }
        return CLAHEComparisonResult(
            original=original_result,
            clahe_enhanced=enhanced_result,
            improvement_metrics=improvement_metrics,
        )
    except Exception as e:
        logger.error(f"Error in camera CLAHE analysis: {e}")
        raise HTTPException(
            status_code=500, detail=f"Camera CLAHE analysis failed: {str(e)}"
        )
```

 # Модуль #2 програми «YOLOAutoDatasetCreation» призначений для автоматизованого
формування датасету та донавчання моделі YOLOv8n нано-розміру; формування датасету
виконується шляхом 3 етапів селекції кадрів відеопотоку.

```
"""
API routes for automated dataset collection (Screen 1)
"""
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
import asyncio
import logging
from typing import Optional
from datetime import datetime

logger = logging.getLogger(__name__)
router = APIRouter(prefix="/api/dataset", tags=["dataset"])

class DatasetStatus(BaseModel):
    is_running: bool
    total_images: int
    global_novel: int
    pairwise_novel: int
    per_car_novel: int
    last_capture: Optional[str] = None

class DatasetConfig(BaseModel):
    interval_sec: int = 5
    camera_index: int = 0
    hash_threshold: int = 2
    per_car_duplicate_ratio: float = 0.8
    min_cars_to_consider: int = 1

# Global state for the dataset collection process
dataset_runner_task: Optional[asyncio.Task] = None
dataset_runner_instance = None
dataset_status = DatasetStatus(
    is_running=False, total_images=0, global_novel=0, pairwise_novel=0, per_car_novel=0
)

@router.post("/start")
async def start_dataset_collection(config: DatasetConfig):
    """Start the automated dataset collection process"""
    global dataset_runner_task, dataset_runner_instance, dataset_status

    if dataset_status.is_running:
        raise HTTPException(
            status_code=400, detail="Dataset collection already running"
        )
```

```
    try:
        # Import here to avoid circular dependencies
        from src.webcam.runner import WebcamNoveltyRunner

        # Create runner instance
        dataset_runner_instance = WebcamNoveltyRunner(
            interval_sec=config.interval_sec,
            camera_index=config.camera_index,
            hash_threshold=config.hash_threshold,
            per_car_duplicate_ratio=config.per_car_duplicate_ratio,
            min_cars_to_consider=config.min_cars_to_consider,
        )

        # Start in background task
        async def run_dataset_collection():
            try:
                dataset_status.is_running = True
                # Run the sync method in executor
                loop = asyncio.get_event_loop()
                await loop.run_in_executor(None, dataset_runner_instance.run)
            except Exception as e:
                logger.error(f"Error in dataset collection: {e}")
                dataset_status.is_running = False
            finally:
                dataset_status.is_running = False

        dataset_runner_task = asyncio.create_task(run_dataset_collection())

        return {"message": "Dataset collection started", "status": dataset_status}

    except Exception as e:
        logger.error(f"Error starting dataset collection: {e}")
        raise HTTPException(status_code=500, detail=f"Failed to start: {str(e)}")

import time
import cv2
from pathlib import Path
import imagehash
from PIL import Image
import logging

from object_detectors.yolo_object_detector import YOLOObjectDetector
from change_detectors.change_detector import ChangeDetector
from storage.storage_manager import StorageManager
from storage.redis_manager import RedisHashStorageManager
from core.utils import crop_and_phash

logger = logging.getLogger(__name__)
```

```
class WebcamNoveltyRunner:
    def __init__(
        self,
        interval_sec=5,
        camera_index=0,
        storage_dir="output",
        hash_threshold=2,
        per_car_duplicate_ratio: float = 0.8,  # fraction required to treat frame as duplicate
        min_cars_to_consider: int = 1,  # optional minimum number of cars to consider ratio (keeps logic
simple)
    ):
        self.interval_sec = interval_sec
        self.camera_index = camera_index
        self.hash_threshold = hash_threshold

        # per-car duplicate policy
        self.per_car_duplicate_ratio = per_car_duplicate_ratio
        self.min_cars_to_consider = min_cars_to_consider

        self.storage_manager = StorageManager(storage_dir)
        self.detector = YOLOObjectDetector("yolov8m.pt", conf=0.25)
        self.change_detector = ChangeDetector()
        self.redis_store = RedisHashStorageManager()

        self.prev_image = None
        self.prev_boxes = None

        self.global_image_counter = 0
        self.global_novel_image_counter = 0
        self.per_car_novel_image_counter = 0
        self.pairwise_novel_image_counter = 0

    def run(self):
        cap = cv2.VideoCapture(self.camera_index, cv2.CAP_V4L2)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        # Warm-up
        for _ in range(5):
            cap.read()

        try:
            while True:
                time.sleep(self.interval_sec)
                ret, frame = cap.read()
                if not ret:
                    logger.warning("Failed to grab frame")
                    continue
```

```
            self.global_image_counter += 1
            self._process_frame(frame)

        finally:
            cap.release()
            cv2.destroyAllWindows()
            logger.info(f"Total images captured: {self.global_image_counter}")
            logger.info(
                f"Total novel images captured after global hash check: {self.global_novel_image_counter}"
            )
            logger.info(
                f"Total pariwise images captured after Hungarian IoU + RMSE check:
{self.pairwise_novel_image_counter}"
            )
            logger.info(
                f"Total per car hashes images captured after per car hash check:
{self.per_car_novel_image_counter}"
            )
            logger.info("Saving latest hashes to Redis RDB...")
            self.redis_store.r.save()
            logger.info("Shutdown complete.")

    def _process_frame(self, frame):
        image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        frame_name = time.strftime("%Y%m%d_%H%M%S")

        # If first image, save it and it's hashes with car bounding boxes
        if self.prev_image is None:
            boxes = self.detector.detect(image)
            frame_hash = imagehash.phash(image)
            car_hashes = [crop_and_phash(image, b) for b in boxes]
            self._save(image, boxes, frame_name, frame_hash, car_hashes, first=True)
            self.prev_image, self.prev_boxes = image, boxes
            return

        global_novel_save_path = self.storage_manager.out_dir / "all_images"
        Path.mkdir(global_novel_save_path, exist_ok=True)
        self.storage_manager.save_image(image, frame_name, global_novel_save_path)
        logger.info(f"Checking images: {frame_name}")
        # Global hash novelty checks
        frame_hash = imagehash.phash(image)
        is_global_novel = self._check_global(frame_hash)
        if not is_global_novel:
            logger.info("Global hash: Frame too similar, not saved.")
            return
        self.global_novel_image_counter += 1
        global_novel_save_path = self.storage_manager.out_dir / "global_novel_images"
        Path.mkdir(global_novel_save_path, exist_ok=True)
```

```
        self.storage_manager.save_image(image, frame_name, global_novel_save_path)

        # Detect cars via YOLO only after global hash check
        boxes = self.detector.detect(image)

        # Paiwise Hungarian + IoU + RMSE novetly check
        is_pairwise_novel = self._check_pairwise(image, boxes)
        if not is_pairwise_novel:
            self.prev_image, self.prev_boxes = image, boxes
            logger.info("Pairwise: Frame too similar, not saved.")
            return
        self.pairwise_novel_image_counter += 1
        pairwise_novel_save_path = self.storage_manager.out_dir / "paiwise_novel_images"
        Path.mkdir(pairwise_novel_save_path, exist_ok=True)
        self.storage_manager.save_image(image, frame_name, pairwise_novel_save_path)

        # Per car hash novetly check
        car_hashes = [crop_and_phash(image, b) for b in boxes]
        is_per_car_novel = self._check_per_car(car_hashes)
        if not is_per_car_novel:
            self.prev_image, self.prev_boxes = image, boxes
            logger.info("Per car hash: Frame too similar, not saved.")
            return
        self.per_car_novel_image_counter += 1

        self.prev_image, self.prev_boxes = image, boxes

        # save logic
        self._save(image, boxes, frame_name, frame_hash, car_hashes)

    def _check_global(self, frame_hash):
        for h in self.redis_store.get_frame_hashes():
            if abs(frame_hash - h) <= self.hash_threshold:
                return False
        return True

    def _check_per_car(self, car_hashes):
        """
        Return True if frame is considered novel based on per-car hashes.
        We mark frame as NOT novel (False) only if >= per_car_duplicate_ratio
        of detected car hashes are duplicates according to hash_threshold.
        """
        # If no cars detected -> consider novel (so frame can still be saved by other checks)
        if not car_hashes:
            logger.debug("No car hashes to check -> treating as novel.")
            return True

        stored_hashes = list(self.redis_store.get_car_hashes())
```

```
    # If DB empty -> everything is novel
    if not stored_hashes:
        logger.debug("No stored car hashes -> treating as novel.")
        return True
    num_cars = len(car_hashes)
    num_duplicates = 0

    # For each detected car hash, check if it matches ANY stored hash within threshold
    for ph in car_hashes:
        is_dup = False
        for stored_ph in stored_hashes:
            if abs(ph - stored_ph) <= self.hash_threshold:
                is_dup = True
                break
        if is_dup:
            num_duplicates += 1

    dup_ratio = num_duplicates / num_cars
    logger.info(
        f"Per-car duplicate check: {num_duplicates}/{num_cars} duplicates "
        f"(ratio={dup_ratio:.2f}, threshold={self.per_car_duplicate_ratio})"
    )

    # If we have very few cars: still apply the ratio rule (you can change this policy if you want)
    # Discard (not novel) only if duplicate ratio >= per_car_duplicate_ratio
    if dup_ratio >= self.per_car_duplicate_ratio:
        return False
    return True

def _check_pairwise(self, image, boxes):
    if self.prev_image is None or self.prev_boxes is None:
        return True
    return self.change_detector.is_novel(
        self.prev_image,
        self.prev_boxes,
        image,
        boxes,
        Path(f"output/matching_vis/{time.strftime('%Y%m%d_%H%M%S')}.png"),
    )

def _save(self, image, boxes, frame_name, frame_hash, car_hashes, first=False):
    self.storage_manager.save_frame(image, boxes, frame_name)
    self.redis_store.add_frame_hash(str(frame_hash))
    for ph in car_hashes:
        self.redis_store.add_car_hash(str(ph))
    msg = "Saved first frame" if first else "Saved novel frame"
    logger.info(f"{msg}: {frame_name}")
```

Додаток Ж

Контейнеризація та оркестрація засобами Docker Compose в програмі "YOLOAutoDatasetCreation"

З метою ізольованості компонентів, відтворюваності середовища та спрощення розгортання комп'ютерна програма "YOLOAutoDatasetCreation" реалізована як набір Docker контейнерів, оркестрованих за допомогою Docker Compose. Переваги застосування Docker Compose такі:

1. Ізоляція залежностей – кожен компонент (frontend, backend, Redis) виконується у власному контейнері з ізольованим файловим простором та мережевим стеком. Це усуває конфлікти версій бібліотек та забезпечує детерміновану поведінку системи.

2. Портативність – Docker образи інкапсулюють всі runtime залежності (Python interpreter, Node.js, системні бібліотеки OpenCV). Система може бути розгорнута на будь-якій платформі з підтримкою Docker без модифікації.

3. Відтворюваність – Dockerfile описує процес побудови образу декларативно. Це гарантує, що production середовище ідентичне development середовищу.

4. Ефективність на ARM64 – Docker нативно підтримує ARM64 архітектуру Raspberry Pi 5. Використання multi-stage builds дозволяє мінімізувати розмір фінальних образів.

Файл docker-compose.yml описує три сервіси з їх конфігурацією:

```
services:
  redis:
    image: redis: latest
    volumes: ./redis_data:/data
    command: redis-server --save 60 1 --loglevel warning
  backend:
    build: ./backend
    volumes:
      - .:/app
      - /app/.venv
    environment:
      - REDIS_HOST=redis
    depends_on: [redis]
  frontend:
```

```
build: ./frontend
volumes:
  - ./frontend:/app
  - /app/node_modules
depends_on: [backend]
```

Ключові аспекти конфігурації:

1. Мережева ізоляція: Всі сервіси підключені до bridge мережі phd-network. Сервіси можуть звертатись один до одного за DNS іменами (наприклад, backend звертається до Redis як redis:6379).
2. Volume монтування:

а) Development режим: Код монтується як volume для підтримки hot-reload. Зміни у вихідних файлах негайно відображаються без перебудови контейнерів;

б) Production режим: Код копіюється в образ під час build. Контейнери є self-contained та не залежать від host файлової системи.

3. Залежності сервісів: Директиви depends_on визначають порядок запуску контейнерів. Docker Compose гарантує, що Redis запуститься перед backend, а backend – перед frontend.
4. Персистентність даних: Redis том ./redis_data:/data забезпечує збереження RDB файлів на host системі. Дані зберігаються навіть при видаленні контейнера.

Розроблена програма для детектування зображень автомобілів виконується на Raspberry Pi 5, в якому застосовано ARM64 архітектуру з обмеженими обчислювальними ресурсами порівняно з x86_64 серверами. Це накладає специфічні вимоги на оптимізацію параметрів системи:

1. Dockerfile використовують multi-architecture образи, що підтримують ARM64:

- python:3.13-slim для backend.
- node:20-alpine для frontend.
- redis: latest (офіційний образ підтримує ARM64).

Alpine Linux обрано для frontend через мінімальний розмір (5 МБ base image проти 120 МБ для Ubuntu).

2. Оптимізація OpenCV:

OpenCV з PyPI включає прекомпільовані бінарні wheels для ARM64. Використання `opencv-python` замість компіляції вихідного коду зменшує час побудови образу з 45 хвилин до 5 хвилин.

3. Апаратне прискорення відео захоплення:

Використання V4L2 backend (cv2.CAP_V4L2) забезпечує пряме звернення до апаратного відео декодера Broadcom VideoCore VII, що зменшує навантаження на CPU.

4. Управління пам'яттю:

Raspberry Pi 5 містить 8 ГБ RAM, що достатньо для паралельної роботи всіх контейнерів:

- Redis: ~100 МБ (для 100K хешів).
- Backend: ~500 МБ (Python + YOLO model).
- Frontend: ~150 МБ (Node.js development server).

У production режимі frontend віддає статичні файли через nginx, зменшуючи споживання пам'яті до ~20 МБ. Візуалізацію контейнеризації програми за допомогою Docker показано на рисунку Ж.1.

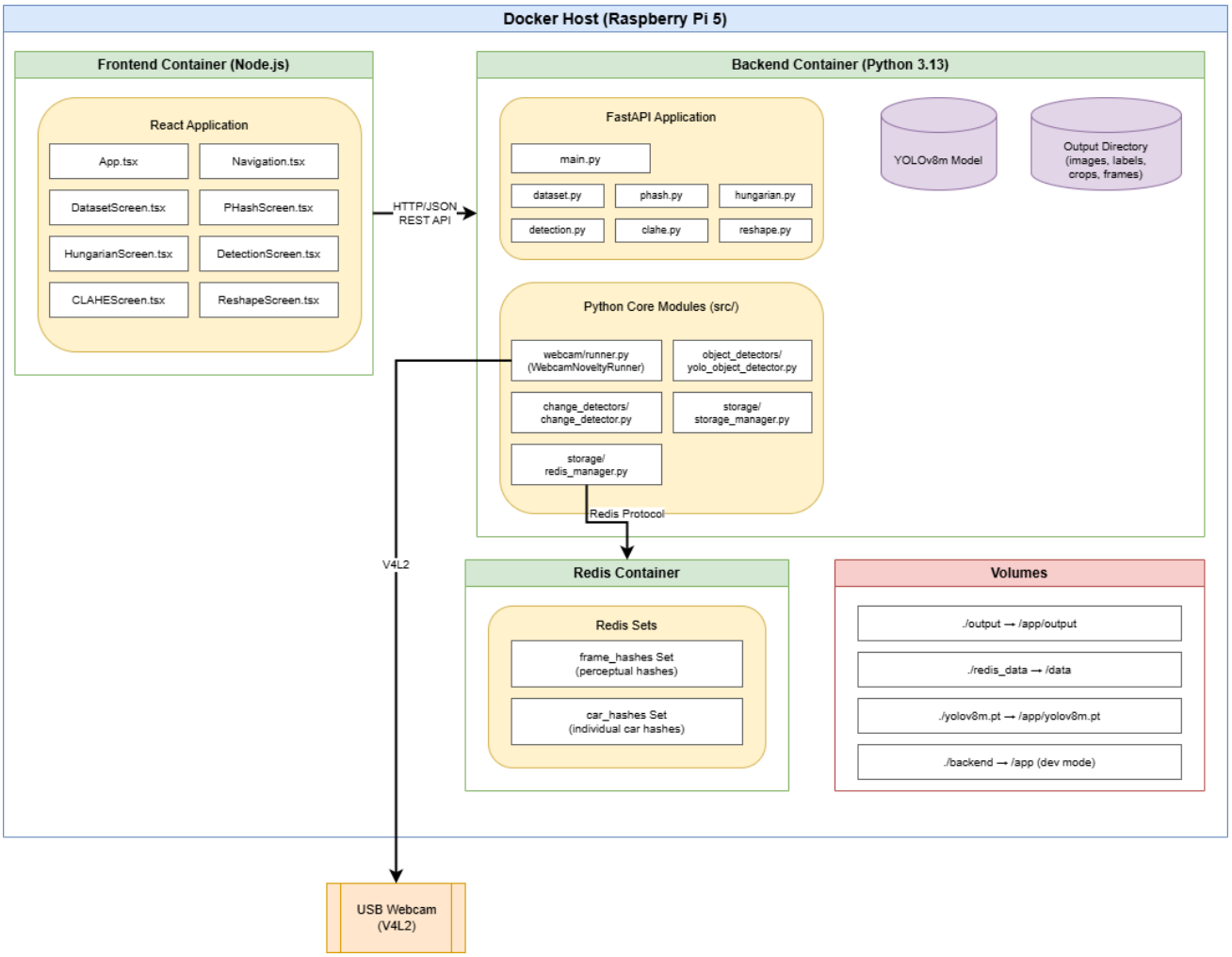


Рисунок Ж.1 – Діаграма компонентів контейнеризації Docker

Додаток К

API ендпоінти та їх функціональність в програмі "YOLOAutoDatasetCreation"

Backend надає RESTful API для управління системою збору даних та виконання аналітичних операцій. API організовано у шість функціональних модулів (routers):

1. Dataset Collection Router (/api/dataset)

   Керування процесом автоматизованого збору датасету:

- POST /api/dataset/start – запуск процесу збору з конфігураційними параметрами (DatasetConfig). Ендпоінт створює екземпляр WebcamNoveltyRunner та запускає його у фоновій asyncio задачі. Це дозволяє синхронному коду Python виконуватись у асинхронному контексті FastAPI.
- POST /api/dataset/stop – зупинка процесу збору. Викликає «task.cancel()» для graceful shutdown та зберігає фінальний RDB snapshot.

2. Perceptual Hash Router (/api/phash)

   Інтерактивна демонстрація роботи перцептивного хешування:

- POST /api/phash/compute – обчислення pHash для завантаженого зображення. Повертає hex-представлення хешу, бінарну форму та base64-encoded зображення для відображення у UI.
- POST /api/phash/compare – порівняння двох зображень через відстань Геммінга їх pHash. Візуалізує схожість через числовий дескриптор відстані.
- POST /api/phash/capture-and-hash – з ахоплення кадру з камери та обчислення його pHash у реальному часі.

3. Hungarian Algorithm Router (/api/hungarian)

   Візуалізація роботи Угорського алгоритму з IoU та RMSE:

- POST /api/hungarian/analyze – приймає два зображення, виконує YOLO детекцію, застосовує Угорський алгоритм. Анотовані зображення містять bounding boxes з кольоровим кодуванням: matching pairs мають однаковий колір, unmatched boxes — сірий.

4. CLAHE Router (/api/clahe)

   Демонстрація адаптивної еквалізації гістограми (англ. Contrast Limited Adaptive Histogram Equalization):

- POST /api/clahe/apply – застосування CLAHE до завантаженого зображення з конфігурованим параметрами clip_limit та tile_grid_size. Повертає порівняльну візуалізацію оригінального та обробленого зображення.

6. Detection Router (/api/detection)

   Демонстрація роботи YOLO детектора:

- POST /api/detection/detect – виконує детекцію на завантаженому зображенні та повертає анотоване зображення з bounding boxes та confidence scores.

7. Reshape Router (/api/reshape)

   Утилітарні операції зміни розміру зображень для підготовки датасетів.