МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ЗАСТОСУВАННЯ АЛГОРИТМІВ ВИЯВЛЕННЯ ОБ’ЄКТІВ У ЗАДАЧАХ АВТОПІЛОТУВАННЯ АВТОМОБІЛІВ

Виконав: студент групи 2П-21
Спеціальності
121 Інженерія програмного забезпечення
Дмитро НЕЧКО
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2025

Завідувачу кафедри
комп’ютерної інженерії та
інформаційних технологій
ХОТУНОВУ Владиславу
студента групи 2П-21
НЕЧКО Дмитро

ЗАЯВА

Прошу затвердити тему кваліфікаційної роботи по кафедрі комп’ютерної інженерії та інформаційних технологій
«Застосування алгоритмів виявлення об’єктів у задачах автопілотування автомобілів» та призначити керівником кваліфікаційної роботи Марченко Станіслав Віталійович, спеціаліста І категорії.

Кваліфікаційну роботу буду виконувати у період з «16» вересня 2024р. по «30» травня 2025 р. із захистом у червні 2025 р.

«_____» _____ 2024 р. Підпис студента _____

УЗГОДЖЕНО
Завідувач кафедри
комп’ютерної інженерії та
інформаційних технологій

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.
Керівник
кваліфікаційної роботи
_____ Марченко Станіслав
(підпис)
«_____»_____2024 р.

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Нечку Дмитру Васильовичу

1. Тема кваліфікаційної роботи Застосування алгоритмів виявлення об'єктів у задачах автопілотування автомобілів
Керівник роботи Марченко Станіслав Віталійович, спеціаліста І категорії

затверджені наказом закладу вищої освіти від «13» жовтня 2024 року № 68у.
2. Строк подання студентом кваліфікаційної роботи 03.06.2025
3. Вихідні дані до кваліфікаційної роботи мова програмування Python, спеціалізовані бібліотеки для обробки зображень і глибокого навчання, зокрема, OpenCV, PyTorch, TensorFlow та Ultralytics, набори даних KITTI.
4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити) огляд сучасних алгоритмів виявлення об'єктів (основні поняття та задачі комп'ютерного зору, огляд методів виявлення об'єктів, популярні підходи до реалізації та огляд платформ з готовими рішеннями для реалізації, постановка завдання на розробку), вибір та оптимізація алгоритмів для виявлення об'єктів (формування вимог та процедури підготовки вхідних даних, алгоритми виявлення об'єктів на основі традиційних методів, алгоритми виявлення об'єктів на основі згорткових нейронних мереж), експериментальні дослідження та оцінка результатів (метрики та процедури оцінки якості алгоритмів виявлення об'єктів, тестування алгоритмів, аналіз результатів експериментів).
5. Дата видачі завдання 15.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1 Огляд сучасних алгоритмів виявлення об’єктів	9.12.2024	
3	Розділ 2 Вибір та оптимізація алгоритмів для виявлення об’єктів	10.03.2025	
4	Розділ 3 Експериментальні дослідження та оцінка результатів	28.04.2025	
5	Висновки	12.05.2025	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2025	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2025	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2025	

Студент _____ Дмитро НЕЧКО
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена дослідженню сучасних технологій виявлення об’єктів у комп’ютерному зорі та їх практичного застосування в системах автономного транспорту. У межах дослідження проведено комплексний аналіз алгоритмів детекції об’єктів, включаючи як традиційні методи (HOG, SIFT), так і сучасні архітектури глибокого навчання (YOLO, Faster R-CNN, SSD, RetinaNet).

У роботі розглянуто еволюцію технологій комп’ютерного зору, особливості реалізації алгоритмів у популярних фреймворках (TensorFlow, PyTorch, Hugging Face), а також проведено порівняльний аналіз їх продуктивності за ключовими метриками (mAP, FPS). Особлива увага приділена дослідженню можливостей оптимізації роботи алгоритмів для систем реального часу.

Визначено критерії оцінки ефективності методів детекції об’єктів, розроблено методику порівняльного тестування на стандартних наборах даних. Експериментальна частина включає аналіз точності та швидкодії різних архітектур у типових дорожніх сценаріях.

Результати дослідження можуть бути корисними для розробників систем комп’ютерного зору, інженерів у галузі автономного транспорту та дослідників у сфері штучного інтелекту. Отримані висновки дозволяють визначити оптимальні підходи для реалізації систем детекції об’єктів у залежності від конкретних вимог до точності та продуктивності.

Ключові слова: КОМП’ЮТЕРНИЙ ЗІР, ВИЯВЛЕННЯ ОБ’ЄКТІВ, ГЛИБОКЕ НАВЧАННЯ, АВТОНОМНИЙ ТРАНСПОРТ, НЕЙРОННІ МЕРЕЖІ, АЛГОРИТМИ ОБРОБКИ ЗОБРАЖЕНЬ.

ABSTRACT

The qualification work is devoted to studying modern technologies for detecting objects in computer vision and their practical application in autonomous transportation systems. The study comprehensively analyzed object detection algorithms, including traditional methods (HOG, SIFT) and modern deep learning architectures (YOLO, Faster R-CNN, SSD, RetinaNet).

The paper considers the evolution of computer vision technologies, the features of algorithm implementation in popular frameworks (TensorFlow, PyTorch, Hugging Face), and a comparative analysis of their performance by key metrics (mAP, FPS). Particular attention is paid to the study of the possibilities of optimizing the work of algorithms for real-time systems.

The criteria for evaluating the effectiveness of object detection methods are defined, and a methodology for comparative testing on standard data sets is developed. The experimental part includes an analysis of the accuracy and performance of different architectures in typical traffic scenarios.

The results of the work can be useful for developers of computer vision systems, engineers in the field of autonomous vehicles, and researchers in the field of artificial intelligence. The findings allow us to determine the optimal approaches for implementing object detection systems depending on specific accuracy and performance requirements.

Keywords: COMPUTER VISION, OBJECT DETECTION, DEEP LEARNING, AUTONOMOUS VEHICLES, NEURAL NETWORKS, IMAGE PROCESSING ALGORITHMS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

mAP	mean Average Precision
CNN	Convolution Neural Network
DBT	Detection Based Tracking
GPU	Graphics processing unit
HOG	Histogram of Oriented Gradients
SURF	Speeded Up Robust Features
SIFT	Scale-Invariant Feature Transform
MAP	maximal a posteriori
MOT	Multi-Object Tracking
SSS	Small Sample Size
Faster R-CNN	Faster Regions with Convolutional Neural Networks
SSD	Single Shot MultiBox Detector
YOLO	You Only Look Once
YOLOv11	You Only Look Once version 11
FPS	Frames per second

ВСТУП

На сьогоднішній день технології комп’ютерного зору знаходять застосування у різноманітних галузях – від автономного транспорту до медичної діагностики. Проте саме у сфері автономного водіння вони набувають особливого значення, оскільки від точності виявлення об’єктів безпосередньо залежить безпека руху. За даними 2024 року, 78% сучасних систем автопілотування використовують алгоритми на основі глибокого навчання, тоді як 22% все ще покладаються на традиційні методи обробки зображень [1]. У різних регіонах ці пропорції відрізняються: наприклад, у Європі перевага надається гібридним системам, що поєднують нейромережі з класичними алгоритмами, тоді як у Північній Америці домінують чисті нейромережеві рішення [1].

Ринок технологій детекції об’єктів постійно еволюціонує під впливом нових архітектур глибокого навчання, зростаючої потужності обчислювальних систем та зміни вимог до точності та швидкодії. Популярність таких алгоритмів як YOLO, Faster R-CNN та їхніх модифікацій, поширення трансформерних архітектур у комп’ютерному зорі, а також інтеграція традиційних методів з сучасними нейромережами змушують переосмислювати підходи до розроблення систем комп’ютерного зору. У цьому контексті особливої актуальності набуває пошук оптимальних технічних рішень, що дозволяють досягати високої точності детекції при збереженні можливості роботи у реальному часі.

Об’єктом дослідження є архітектури та алгоритми виявлення об’єктів, зокрема сучасні нейромережеві моделі та традиційні методи, а також їх реалізації у популярних фреймворках машинного навчання.

Предметом дослідження виступають технологічні підходи до реалізації систем детекції об’єктів, зокрема методи оцінки їх ефективності, оптимізації продуктивності та адаптації до різних умов роботи.

Мета дослідження полягає в проведенні комплексного порівняльного аналізу сучасних методів виявлення об’єктів для визначення оптимальних рішень для різних прикладних задач.

У межах кваліфікаційної роботи обрано до розгляду та реалізації такі завдання:

1) дослідити сучасні методи виявлення об’єктів на основі глибокого навчання та традиційних алгоритмів комп’ютерного зору;

2) проаналізувати популярні архітектури нейромереж (YOLO, Faster R-CNN) та їх реалізації;

3) розробити систему критеріїв для оцінки ефективності алгоритмів детекції об’єктів;

4) провести експериментальне порівняння обраних методів на стандартних наборах даних.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ АЛГОРИТМІВ ВИЯВЛЕННЯ ОБ'ЄКТІВ

1.1 Основні поняття та задачі комп'ютерного зору

Комп'ютерний зір як складова штучного інтелекту займається розробкою методів автоматизованого аналізу й інтерпретації візуальної інформації та охоплює комплекс підходів до отримання, обробки, аналізу та семантичного розуміння зображень і відеопотоків, що знаходять застосування у різноманітних галузях сучасної науки і техніки. Центральне місце серед задач комп'ютерного зору займає проблема виявлення об'єктів, яка передбачає їхню ідентифікацію, аналіз та подальшу взаємодію у просторово-часовому континуумі візуальних даних [1].

Фундаментальною складовою процесу виявлення об'єктів є їх локалізація (рис. 1.1, 1.2), яка реалізується через визначення точних координат за допомогою обмежувальних рамок [1]. Ця методика є особливо важливою для автономних систем, зокрема безпілотних транспортних засобів, де необхідно забезпечити високоточне розпізнавання елементів дорожньої інфраструктури та інших учасників руху. Локалізація об'єктів виступає ключовим етапом для подальшого аналізу, класифікації та організації взаємодії з виявленими об'єктами в таких важливих сферах як автономні транспортні системи, сучасна медична діагностика та робототехнічні комплекси [1].

Значний науковий інтерес представляють практичні застосування методів локалізації об'єктів у різних галузях. У медичній діагностиці ці технології дозволяють точно визначати зони патологічних змін на рентгенографічних знімках та результатах магнітно-резонансної томографії (рис. 1.1) [2]. У сфері відеоспостереження методи автоматичного виявлення об'єктів застосовуються для ідентифікації облич та потенційно небезпечних предметів у громадських місцях [2]. Промислові підприємства активно використовують алгоритми локалізації для контролю якості продукції та оптимізації виробничих процесів,

що підтверджується результатами останніх досліджень у галузі комп'ютерного зору [2].

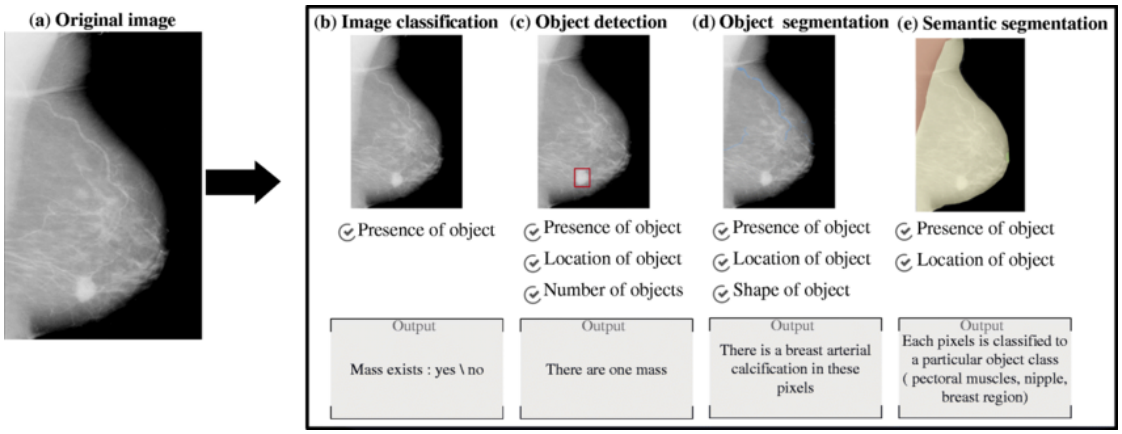

Рисунок 1.1 – a) Вхідне зображення; b) Класифікація зображень; c) Виявлення об'єктів; d) Сегментація артеріальної кальцифікації молочної залози на зображеннях; e) Сегмент різних ділянок молочної залози [3]

Сучасні дослідження в галузі комп'ютерного зору свідчать про значний потенціал удосконалення методів локалізації об'єктів шляхом інтеграції класичних підходів обробки зображень із передовими архітектурами глибокого навчання [1, 2]. Особливий науковий інтерес викликають гібридні методи, які комбінують переваги різних підхідів, забезпечуючи при цьому високий рівень точності розпізнавання при збереженні оптимальної обчислювальної складності [1, 2]. Важливим напрямком досліджень є класифікація об'єктів, де комп'ютерні системи аналізують характерні ознаки та визначають їх приналежність до певних категорій [4]. У сфері біомедичної діагностики цей підхід дозволяє ідентифікувати патологічні зміни клітинних структур на медичних зображеннях, що є критично важливим для ранньої діагностики захворювань [3].

Системи комп'ютерного зору ефективно вирішують задачу одночасного виявлення множинних об'єктів, що передбачає їх паралельну локалізацію та класифікацію [5]. Такий підхід знайшов широке застосування у системах відеоспостереження та аналізу складних сцен у реальному часі, де необхідно

одночасно ідентифікувати численні об’єкти різних типів. Методи точної сегментації, що дозволяють детально визначати межі об’єктів, відіграють ключову роль у подальшому аналізі їх форми та взаємодії з навколишнім середовищем [6]. Ці технології знайшли активне застосування у робототехнічних системах та медичних дослідженнях, де точність виділення об’єктів є вирішальним фактором для подальшого аналізу.

Важливим напрямком досліджень у галузі комп’ютерного зору є аналіз контексту сцени, який дозволяє не лише ідентифікувати окремі об’єкти, а й визначати їх семантичні взаємозв’язки та функціональну роль у межах аналізованого простору [7]. Особливе значення цей підхід набуває в системах розширеної реальності, де точна інтерпретація взаємодії об’єктів є критичною для створення інтерактивних середовищ та імерсивних додатків.

Сукупність задач комп’ютерного зору включає також методи розпізнавання руху та динамічних змін у сцені. Ці технології забезпечують адаптацію систем до змінних умов навколишнього середовища, дозволяючи не тільки фіксувати поточний стан, але й формувати прогностичні моделі подальшого розвитку подій [2]. Такі можливості мають принципове значення для безпілотних транспортних систем та інтелектуальних систем безпеки, де реакція на зміни має відбуватися в режимі реального часу.

Теоретичний фундамент комп’ютерного зору як наукової дисципліни ґрунтується на системі ключових понять, серед яких центральне місце займає концепція обмежувального прямокутника (bounding box). Ця математична модель, що визначається чотирма параметрами – координатами центру (x, y), шириною та висотою – забезпечує точне просторове позиціонування об’єкта на зображенні (рис. 1.2) [12]. Сучасні алгоритми розширюють цю концепцію за рахунок використання орієнтованих прямокутників, що дозволяє враховувати просторову орієнтацію об’єктів відносно осей зображення.

Критично важливим параметром у процесі виявлення об’єктів є показник достовірності (confidence score), який кількісно оцінює рівень довіри системи до коректності виявлення. Ця метрика, що приймає значення в інтервалі [0, 1],

інтегрує оцінки як точності локалізації, так і правильності класифікації, виконуючи ключову роль у процедурах фільтрації та верифікації результатів [6]. Оптимізація цього показника є предметом інтенсивних досліджень, оскільки безпосередньо впливає на якість роботи систем комп'ютерного зору в реальних умовах.

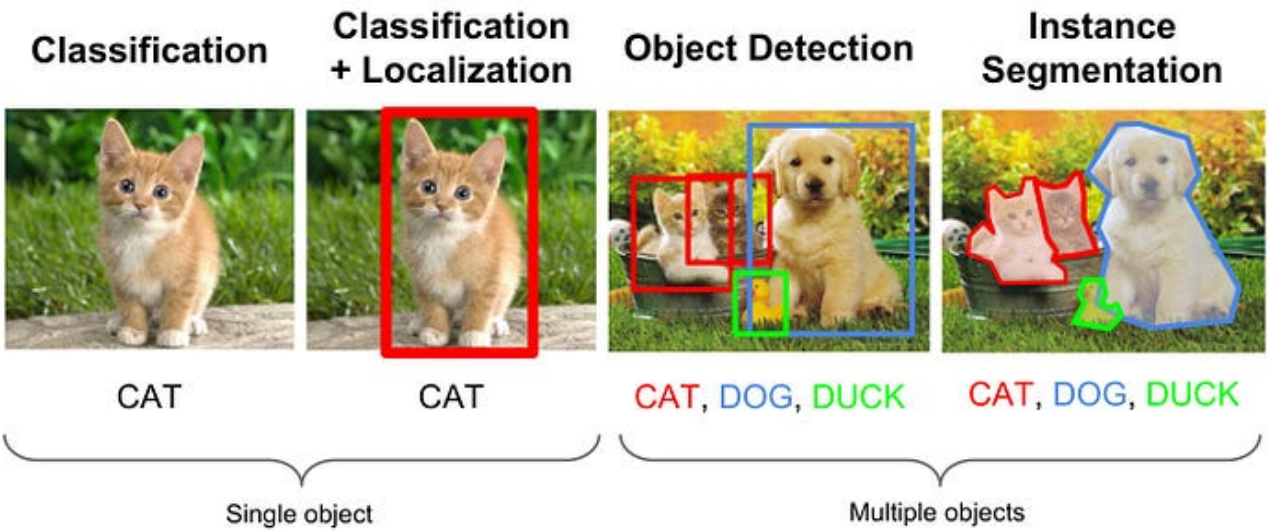

Рисунок 1.2 – Задачі комп'ютерного зору в контексті виявлення об'єктів [8]

Концепція сегментації зображень (рис. 1.2) охоплює два принципово різних підходи до обробки візуальних даних. Сегментація екземплярів передбачає точне піксельне виділення окремих об'єктів, тоді як семантична сегментація фокусується на класифікації пікселів без розрізнення конкретних екземплярів [1]. Ці методи знайшли широке застосування в задачах, що вимагають високої точності визначення меж об'єктів, таких як медична діагностика або автоматизований контроль якості. Аналіз характерних точок (keypoints) представляє собою важливий інструмент дослідження просторової структури об'єктів. У системах розпізнавання жестів і аналізу поз людини такі точки відповідають анатомічним орієнтирам (наприклад, суглобам), що дозволяє точно відстежувати динаміку змін положення об'єкта [4]. Цей підхід особливо ефективний у задачах, що вимагають аналізу просторової орієнтації та рухів.

Класифікація об’єктів (рис. 1.2) формує фундамент для багатьох практичних застосувань комп’ютерного зору. Сучасні системи аналізують комплекс візуальних ознак, включаючи текстуру, форму, геометричні параметри та колірні характеристики, що забезпечує високу точність ідентифікації [1]. Важливість цієї задачі підтверджується її застосуванням у різних галузях: від систем безпеки з розпізнаванням осіб до медичної діагностики та промислового контролю якості [2].

Історичний розвиток методів класифікації демонструє еволюцію від традиційних підходів, таких як гістограми направлених градієнтів (HOG) [9], до сучасних архітектур глибокого навчання. Ранні алгоритми (SIFT, SURF) забезпечували стійкість до масштабних перетворень і обертань за рахунок аналізу ключових точок [9, 10], тоді як сучасні нейромережні моделі дозволяють виявляти складні залежності у візуальних даних [4].

Пороговий аналіз залишається базовим методом для відокремлення об’єктів від фону в умовах високого контрасту [13], тоді як методи на основі градієнтів (оператори Собеля, Кенні) ефективно виявляють контури та межі складних структур [1]. Ці підходи становлять теоретичну основу для більш складних алгоритмів обробки зображень.

Регіональні методи обробки зображень базуються на аналізі зв’язності пікселів та їх групуванні за критеріями подібності візуальних ознак. Такі підходи демонструють особливу ефективність при виявленні складних об’єктів із неоднорідною структурою [14]. Паралельно розвиваються статистичні методи, які використовують аналіз розподілу таких характеристик як колір, текстура та інтенсивність. Зокрема, моделі гаусової суміші (Gaussian Mixture Model) дозволяють не лише виділяти об’єкти, а й прогнозувати їх просторове положення у складних сценах [4].

Сучасні системи комп’ютерного зору все частіше використовують комбінований підхід, що поєднує локалізацію та класифікацію об’єктів. Цей двоетапний процес передбачає спочатку точне визначення положення об’єкта у просторі сцени, а потім детальний аналіз його характеристик для точної

ідентифікації. Особливу важливість такий підхід набуває в автономних транспортних системах, де необхідно не лише виявити учасників руху, а й аналізувати їх поведінку для прогнозування небезпечних ситуацій [7].

Локалізація об’єктів як фундаментальний компонент комп’ютерного зору забезпечує можливість ефективної взаємодії з об’єктами у складних динамічних сценах. Сучасні дослідження демонструють тенденцію до інтеграції процесів локалізації та класифікації в єдиний конвеєр обробки даних. У таких системах етап локалізації формує регіони інтересу, які потім підлягають детальному аналізу класифікаційними алгоритмами. Подібні архітектури знайшли широке застосування у таких критично важливих галузях як автономний транспорт, промислова автоматизація та обробка супутникових знімків [7].

1.2 Огляд методів виявлення об’єктів

Центральним питанням дослідження було обрано задачу виявлення об’єктів. Сучасні методи можна класифікувати за різними критеріями, проте основним є поділ на традиційні підходи та методи на основі глибокого навчання.

Традиційні методи детекції об’єктів, які активно використовувалися до появи глибокого навчання, базуються на класичних алгоритмах обробки зображень і машинного навчання. Ці підходи характеризуються ручним виділенням ознак і застосуванням стандартних класифікаторів. До основних традиційних методів виявлення відносяться HOG, SIFT та SURF.

Метод HOG (Histograms of Oriented Gradients), розроблений Dalal та Triggs у 2005 році, становить фундаментальний підхід у задачі виявлення об’єктів, заснований на аналізі їх контурних характеристик та просторової структури [9]. Цей алгоритм знайшов широке застосування у системах детекції пішоходів, демонструючи високу ефективність у стандартизованих умовах.

Метод ґрунтується на аналізі локальних градієнтів яскравості зображення. Процес обробки починається з нормалізації гамма-корекції та кольорового простору, після чого відбувається обчислення градієнтів у кожній точці

зображення. Наступним етапом є квантування напрямків градієнтів у локальних областях та формування гістограм орієнтованих градієнтів. Завершальна стадія передбачає нормалізацію гістограм у блоках для підвищення інваріантності (рис. 1.3) [9].

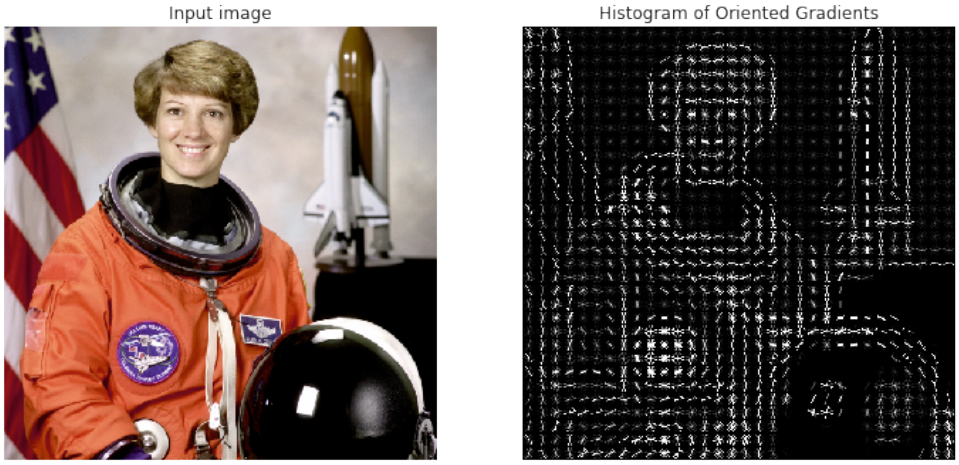

Рисунок 1.3 – Візуалізація для гістограми направлених градієнтів із фото Айлін Коллінз [14]

Аналіз продуктивності методу виявляє певні обмеження. HOG демонструє чутливість до значних змін перспективи, зниження ефективності при нелінійних деформаціях об’єктів та складності у роботі з неоднорідним фоном [9]. Застосування HOG у практичних системах охоплює широкий спектр галузей, включаючи системи відеоспостереження та безпеки, автономні транспортні системи, медичну діагностику за зображеннями та біометричну ідентифікацію [2].

Метод SIFT (Scale-Invariant Feature Transform), розроблений Девідом Лоу у 1999 році та вдосконалений у 2004 році, став фундаментальним інструментом у комп’ютерному зорі для виявлення та опису локальних особливостей зображення [10]. Цей алгоритм здійснює аналіз зображень з метою виявлення точок (рис. 1.4), що володіють унікальними візуальними характеристиками, такими як кутові структури чи області з високим контрастом. Для кожної виявленої точки SIFT генерує дескриптор, що містить інформацію про локальний

градієнтний розподіл, що забезпечує інваріантність до масштабу, обертання та часткової оклюзії [10].

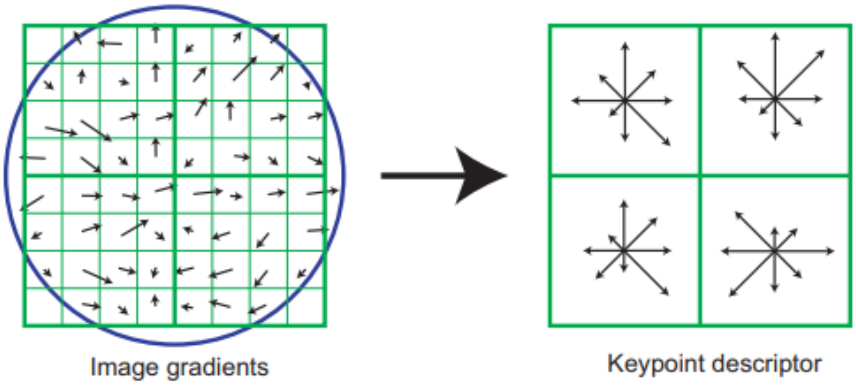

Рисунок 1.4 – Побудова характерних точок [11]

Особливістю методу є його здатність до стабільного функціонування в умовах змін перспективи та масштабу. Ця властивість робить SIFT особливо корисним у застосуваннях, де потрібна надійна робота при варіаціях умов зйомки, таких як біометрична ідентифікація, аналіз супутникових знімків чи системи доповненої реальності [2].

Технічна реалізація SIFT включає чотири основні етапи: побудову піраміди масштабів, виявлення ключових точок, орієнтацію дескрипторів та формування векторів ознак. Кожен з цих етапів сприяє загальній інваріантності методу до афінних перетворень та змін освітленості. Експериментальні дослідження демонструють, що SIFT забезпечує високу повторюваність дескрипторів (близько 80%) у широкому діапазоні умов зйомки, що перевершує показники багатьох альтернативних методів [10].

Незважаючи на численні переваги, метод SIFT має суттєві обмеження, пов’язані з високою обчислювальною складністю, що становить приблизно O(n log n) для зображення розміром n пікселів. Це обмеження робить його менш придатним для систем реального часу, особливо на пристроях з обмеженими обчислювальними ресурсами [12]. Крім того, ефективність методу

знижується при роботі з неструктурованими текстурами або низькоконтрастними зображеннями.

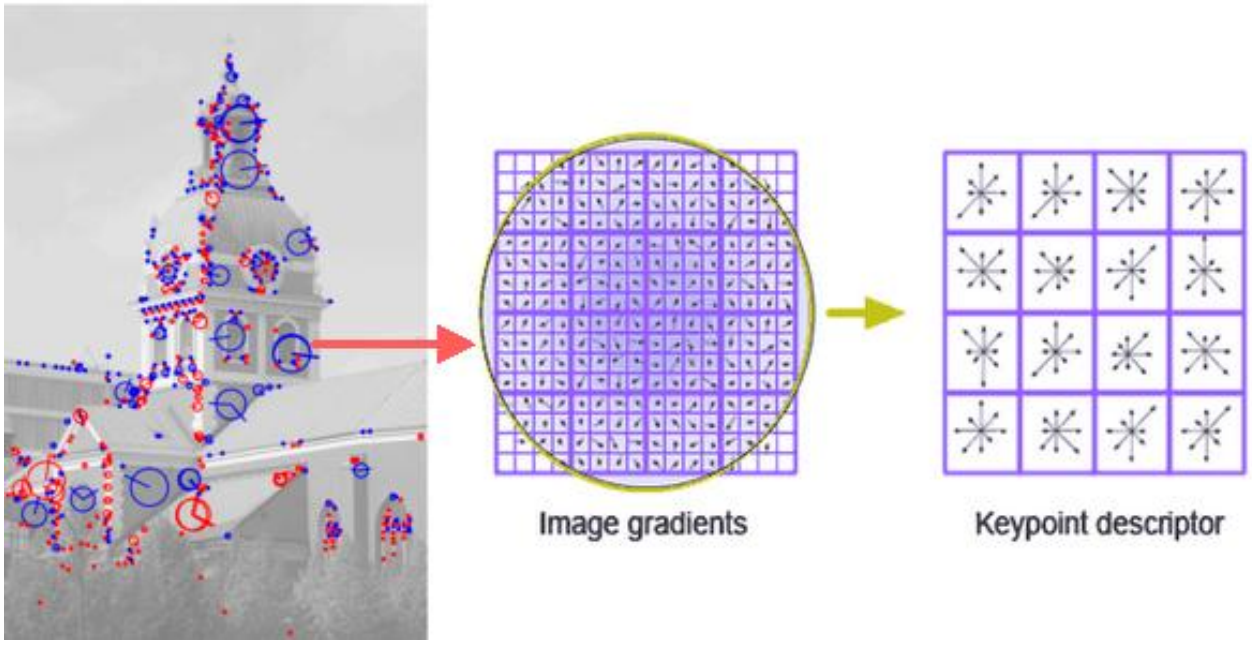

Рисунок 1.5 – Демонстрація роботи SIFT [11]

Метод SURF (Speeded Up Robust Features), розроблений групою дослідників під керівництвом Герберта Бая у 2006 році, представляє собою оптимізований алгоритм виявлення та опису локальних особливостей зображення [12]. Цей метод був створений як ефективна альтернатива SIFT, зберігаючи його основні переваги - інваріантність до масштабних перетворень, обертання та змін перспективи, але при цьому демонструючи значно вищу обчислювальну ефективність. Алгоритм застосовує інтегральні зображення для швидкого обчислення апроксимацій других похідних, що є ключовим фактором підвищення продуктивності. Для опису виявлених особливостей SURF використовує спрощені дескриптори на основі градієнтів Хаара, орієнтованих відносно головного напрямку кожної точки [12]. Приклад знайдених методом SURF характерних точок, накладених на зображення, продемонстровано на рис. 1.6.

Порівняльні дослідження ефективності SURF демонструють його значні переваги у швидкості обробки - алгоритм працює приблизно в 3-5 разів швидше

за SIFT при аналогічних умовах [12]. Ця характеристика робить його особливо привабливим для систем реального часу, таких як відеоспостереження, робототехнічні комплекси та інтерактивні системи доповненої реальності. Крім того, метод демонструє хорошу стійкість до змін освітленості та невеликих афінних спотворень [2]. Однак аналіз точності SURF виявляє певні обмеження методу. Зокрема, дескриптори SURF мають меншу роздільну здатність для складних текстурних структур порівняно з SIFT, що може призводити до зниження точності зіставлення у випадках високодеталізованих зображень [12]. Це обмеження обумовлено використанням спрощених апроксимацій при побудові дескрипторів, що є компромісом для досягнення високої швидкості роботи.

Рисунок 1.6 – Характерні точки SURF [12]

Сучасні практичні застосування SURF охоплюють широкий спектр галузей. У системах біометричної ідентифікації він використовується для стабільного виявлення ключових точок обличчя незалежно від умов освітлення [12].

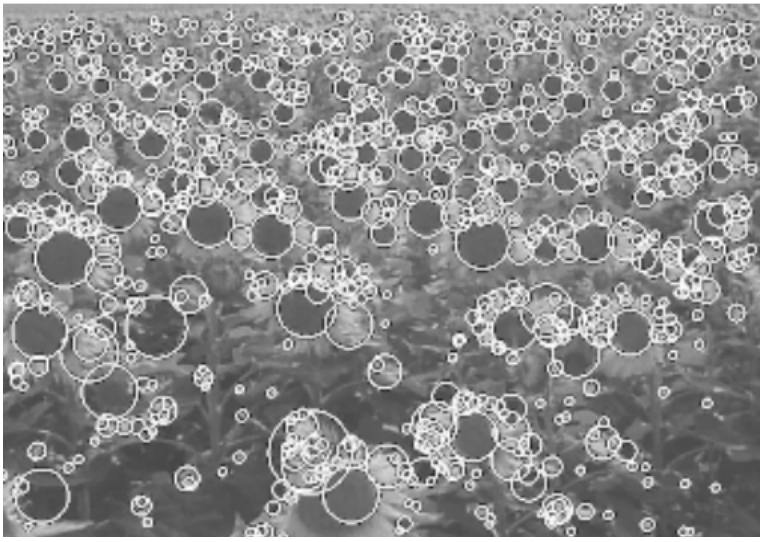

Рисунок 1.7 – Демонстрація SURF на полі соняшників [12]

У відеоаналітиці алгоритм застосовується для трекінгу об’єктів та аналізу їх руху. Для аналізу супутникових знімків SURF виявляється ефективним інструментом для зіставлення та реєстрації зображень [2]. У промислових системах контролю якості метод дозволяє виявляти дефекти поверхонь за рахунок аналізу локальних текстурних особливостей [12].

Традиційні методи детекції об’єктів, незважаючи на свої обмеження, продовжують застосовуватись у системах з обмеженими обчислювальними ресурсами та для вузькоспеціалізованих задач. Проте поява методів на основі глибокого навчання спричинила значний прорив у галузі комп’ютерного зору, забезпечивши якісно новий рівень точності розпізнавання.

Сучасні підходи до детекції об’єктів можна умовно поділити на три основні категорії: одноетапні детектори, двоетапні детектори та трансформери. Одноетапні детектори (Single-shot detectors) представляють особливий інтерес завдяки їх високій ефективності при обробці зображень і відеопотоків у реальному часі [3]. Відмінною рисою цих методів є здатність виявляти об’єкти без попередньої генерації регіонів пропозицій, що істотно знижує обчислювальну складність. Архітектура одноетапних детекторів базується на згорткових нейронних мережах, які одночасно прогнозують координати обмежувальних рамок та класову приналежність об’єктів. Ключовим елементом

таких систем є поділ зображення на просторову сітку, де кожна комірка відповідає за детекцію об’єктів у відповідній області. Такий підхід дозволяє одночасно ідентифікувати множину об’єктів різних категорій [5].

Найбільш відомим представником цього сімейства є модель YOLO (You Only Look Once), яка реалізує принцип єдиного проходу зображення через нейромережу. Ця особливість робить YOLO особливо ефективною для завдань, що вимагають обробки в реальному часі, таких як відеоспостереження чи автономні транспортні системи. Важливим аспектом є використання мультимасштабних карт ознак, які забезпечують точне виявлення об’єктів незалежно від їх розміру у кадрі [5]. Модель YOLO – це одна з найпопулярніших моделей для детекції об’єктів з високим показником швидкодії. Вперше її представив Joseph Redmon у 2016 році, і вона змінила підхід до розпізнавання, дозволяючи аналізувати зображення за один прохід нейромережі [5]. Завдяки цьому YOLO здатна миттєво знаходити десятки об’єктів одночасно, що робить її ідеальною для відеоспостереження, автономних систем та аналізу потокового відео.

Останні досягнення в розвитку YOLO-моделей знайшли своє втілення у версії YOLOv11, яка демонструє значний прогрес у порівнянні з попередніми ітераціями. За даними останніх досліджень, YOLOv11 досягає показника AP (Average Precision) 58.2% на наборі даних COCO при швидкості обробки 145 FPS на GPU NVIDIA A100, що становить значне покращення продуктивності порівняно з YOLOv10 [20]. Реалізація YOLOv11 через фреймворк Ultralytics пропонує комплексний підхід до роботи з моделями детекції об’єктів. Архітектура включає інноваційні механізми покращення якості детекції, такі як динамічне масштабування ознак, адаптивні механізми уваги та оптимізовані функції втрат. Важливою особливістю Ultralytics-реалізації є підтримка повного циклу розробки – від навчання на користувацьких даних до впровадження на різноманітних апаратних платформах [20]. Приклад застосування моделі Ultralytics YOLOv11 для виявлення та класифікації об’єктів показано на рис. 1.8.

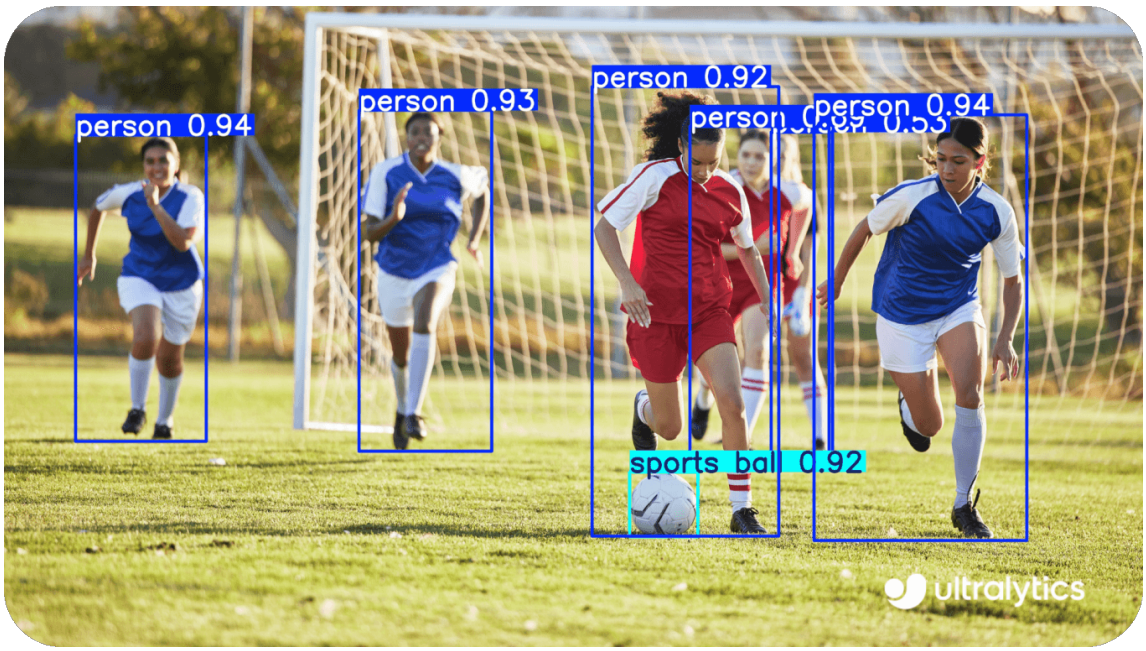

Рисунок 1.8 – Ultralytics YOLOv11 у виявлені об’єктів [19, 20]

Застосування моделі YOLO значно розширилися завдяки його покращеним характеристикам. У автономних транспортних системах модель демонструє високу ефективність у задачах детекції пішоходів, транспортних засобів та дорожньої інфраструктури в реальному часі. Системи відеоспостереження отримують вигоду від покращеної здатності моделі аналізувати складні сцени з високою щільністю об’єктів. У робототехніці YOLO дозволяє реалізовувати точну навігацію та маніпулювання об’єктами завдяки покращеній точності локалізації [20].

Архітектура SSD (Single Shot MultiBox Detector), запропонована Wei Liu та співавторами у 2016 році, представляє собою інноваційний підхід до задачі детекції об’єктів, який усуває необхідність у генерації регіонів пропозицій, характерну для двоетапних методів [16]. Відмінною рисою SSD є здатність виконувати локалізацію та класифікацію об’єктів за єдиний прохід зображення через нейромережу, що забезпечує значне підвищення швидкості обробки. Метод використовує багаторівневих карт ознак, отриманих від різних шарів згорткової нейромережі. Такий підхід дозволяє ефективно виявляти об’єкти різних масштабів – від дрібних до крупних – без необхідності додаткової обробки зображення [16]. Кожен рівень ознак відповідає за детекцію об’єктів

певного діапазону розмірів, що забезпечує високу адаптивність моделі до різноманітних умов зйомки.

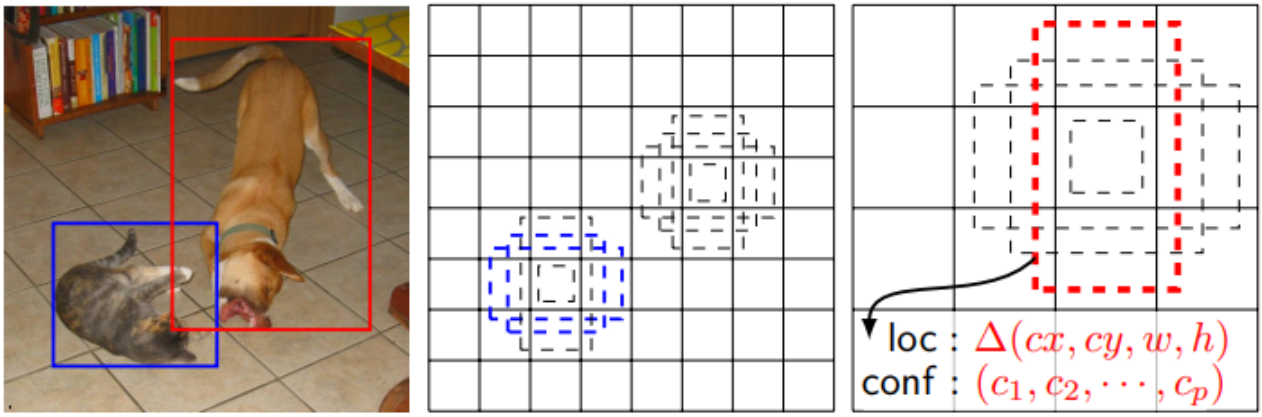

Рисунок 1.9 – Single Shot MultiBox Detector. 1) виявлення об’єктів; 2) Карта ознак 8x8; 3) Карта ознак 4x4; [16]

Дослідження показують, що SSD досягає значної продуктивності у порівнянні з традиційними методами, обробляючи зображення розміром 300×300 пікселів зі швидкістю 59 FPS на GPU Titan X при підтримці 20 класів об’єктів [14]. Метод SSD зазвичай використовують в таких галузях: системи автономного транспорту, де швидка та точна детекція учасників руху є критично важливою, у сфері відеоаналітики, використовується для моніторингу громадських просторів та аналізу транспортних потоків [16].

RetinaNet – це нейромережева модель для виявлення об’єктів, яка вирішує проблему дисбалансу між фоновими областями та реальними об’єктами. Вперше її представили Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He та Serge Belongie у 2017 році [2]. Головна особливість RetinaNet – використання функції втрат Focal Loss, спеціально розроблена для боротьби з переважанням негативних прикладів (фонових регіонів) у процесі навчання. Механізм Focal Loss реалізує динамічне зважування прикладів, зменшуючи внесок легко класифікованих негативних регіонів і концентруючи увагу на складних випадках, що призводить до значного підвищення точності детекції малих та складних об’єктів [2].

Результати тестування RetinaNet демонструють, високі показником середньої точності (AP) на стандартних наборах даних. Зокрема, на наборі COCO модель демонструє AP на рівні 39.1%, що є значним поліпшенням порівняно з попередніми одноетапними підходами [2]. При цьому швидкість обробки залишається достатньою для застосування в системах реального час

Рисунок 1.10 – Анотовані зображення в наборі даних RetinaNet [2]

RetinaNet охоплює широкий спектр задач, де необхідна точна детекція об’єктів різних масштабів. У системах відеоспостереження модель ефективно виявляє дрібні об’єкти на складному фоні, тоді як у медичній діагностиці вона застосовується для аналізу мікроскопічних зображень. Автономні транспортні системи отримують вигоду від здатності RetinaNet до надійної детекції віддалених об’єктів [2].

Головною перевагою одноетапних детекторів є швидкість обробки, що робить їх доречними для задач реального часу [14]. Вони широко застосовуються у сферах відеоспостереження, автономного транспорту та мобільних додатків. Однак їх точність може бути нижчою у випадках складних сцен або малих об’єктів, порівняно з двоетапними детекторами.

Двоетапні детектори, займають особливе місце серед сучасних підходів до виявлення об’єктів у комп’ютерному зорі. Відмінною рисою цих методів є поділ процесу детекції на дві послідовні фази: генерацію регіонів пропозицій та їх подальшу класифікацію, що забезпечує високий рівень точності локалізації та розпізнавання об’єктів [14].

Перший етап роботи таких алгоритмів полягає у генерації набору регіональних пропозицій – областей зображення, які з високою ймовірністю містять цікаві об’єкти. Цей процес може реалізовуватись за допомогою різних технік, включаючи селективний пошук (selective search) або спеціалізовані нейромережеві алгоритми генерації регіонів. Кожна з виявлених областей підлягає подальшому аналізу, що дозволяє зменшити простір пошуку та зосередити обчислювальні ресурси на перспективних ділянках зображення [14].

На другому етапі відбувається детальна обробка отриманих регіонів, яка включає точну класифікацію об’єктів та корекцію їх меж. Для цього застосовуються складні моделі, що аналізують комплекс візуальних ознак, включаючи текстуру, форму, колір та просторові взаємозв’язки між елементами зображення. Такий підхід дозволяє ефективно фільтрувати помилкові пропозиції та досягати високої точності визначення меж об’єктів [14].

Метод R-CNN (Regions with Convolutional Neural Networks) став важливим етапом у розвитку двоетапних детекторів об’єктів. Його архітектура передбачає початкову генерацію регіональних пропозицій за допомогою алгоритму селективного пошуку з подальшою обробкою кожної області згортковою нейромережею для класифікації. Подальші модифікації, такі як Fast R-CNN та Faster R-CNN, значно оптимізували цей процес, зменшивши обчислювальні витрати при збереженні високої точності детекції.

Найбільш досконалим представником цього сімейства є Faster R-CNN, запропонований дослідницькою групою під керівництвом Shaoqing Ren у 2015 році [6]. Ключовою інновацією даної архітектури стало впровадження Region Proposal Network (RPN) – спеціалізованої нейромережі для генерації регіонів інтересу. RPN аналізує особливості зображення на різних масштабах, що дозволяє ефективно виявляти потенційні об’єкти незалежно від їх розміру чи орієнтації у кадрі [6]. Механізм роботи Faster R-CNN ґрунтується на тісній інтеграції двох модулів: RPN для виявлення регіонів та Fast R-CNN для їх подальшої класифікації та уточнення меж. Така архітектура забезпечує значне підвищення продуктивності порівняно з попередніми версіями, зберігаючи при

цьому високу точність детекції навіть для малих або частково оклюдованих об’єктів [6].

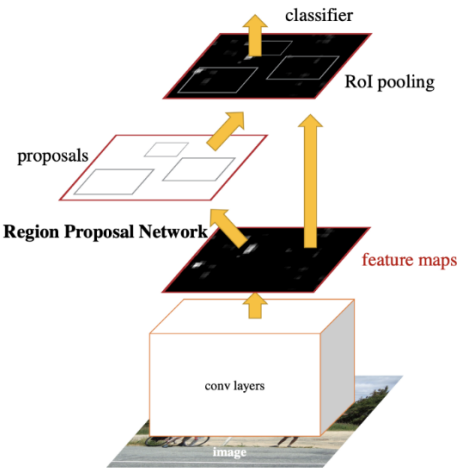

Рисунок 1.11 – Faster R-CNN [14]

Дослідження демонструють, що Faster R-CNN досягає показників середньої точності (mAP) на рівні 42.7% на наборі даних PASCAL VOC 2007, що значно перевершує результати одноетапних детекторів для аналогічних умов [6]. Особливо варто відзначити здатність архітектури до точного визначення меж об’єктів у складних сценах з високою щільністю.

Головною перевагою двоетапних детекторів, зокрема Faster R-CNN, є їхня здатність забезпечувати високоточну локалізацію об’єктів навіть у складних умовах. Ця характеристика робить їх незамінними для завдань, що вимагають деталізованого аналізу зображень, таких як медична діагностика, наукові дослідження або системи безпеки високого рівня [14]. Основним обмеженням архітектури є значна обчислювальна складність, що обумовлена необхідністю обробки множини регіонів інтересу. Це обмежує застосування Faster R-CNN у системах реального часу, особливо на пристроях з обмеженими обчислювальними ресурсами [14]. Сучасні дослідження спрямовані на подолання цього обмеження шляхом оптимізації архітектури мережі та впровадження ефективніших механізмів обробки регіонів.

Сучасний етап розвитку алгоритмів комп'ютерного зору ознаменувався появою моделей, заснованих на трансформерній архітектурі, які запропонували принципово новий підхід до задачі детекції об'єктів. На відміну від класичних згорткових нейронних мереж, трансформери використовують механізм самоконтрольованої уваги (self-attention), що дозволяє ефективно моделювати глобальні залежності між різними областями зображення [8].

Трансформерні детектори представляють зображення у вигляді послідовності патчів – фрагментів фіксованого розміру, кожен з яких обробляється як окремий елемент вхідних даних. Механізм уваги дозволяє моделі виявляти довгострокові залежності між об'єктами та їх контекстом незалежно від їх просторового розташування в кадрі [8]. Ця властивість принципово відрізняє трансформери від згорткових архітектур, чиє поле зору обмежене розміром фільтрів. Важливою перевагою трансформерних архітектур є їхня здатність до паралельної обробки даних, що значно підвищує ефективність роботи з великими зображеннями. На відміну від послідовних згорткових операцій, механізм уваги дозволяє одночасно аналізувати всі частини зображення, що призводить до зменшення часу обробки при зростанні роздільної здатності [8].

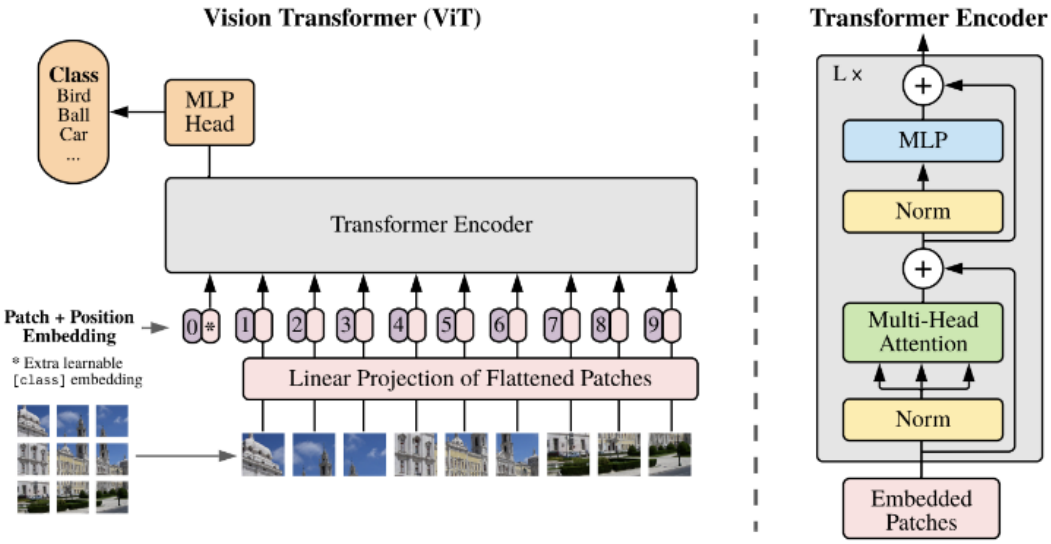

Рисунок 1.12 – Трансформерна архітектура [8]

Трансформерні архітектури демонструють принципові переваги у задачах детекції об’єктів завдяки унікальній здатності до глобального аналізу контексту сцени. Дослідження підтверджують їхню вищу точність у складних умовах, таких як наявність часткової оклюзії або неоднорідного фону, порівняно з традиційними згортковими підходами [8]. Важливою характеристикою цих моделей є також підвищена стійкість до масштабних перетворень і просторових деформацій об’єктів, що обумовлено механізмами самоконтрольованої уваги. Проте трансформерні архітектури мають суттєві обмеження, пов’язані з високими обчислювальними витратами. Квадратична складність алгоритмів відносно розміру вхідного зображення значно обмежує їх застосування на пристроях з обмеженими ресурсами або в системах, що вимагають обробки у строгому реальному часі [8].

Використання трансформерних детекторів охоплюють низку критично важливих галузей. У медичній діагностиці вони забезпечують точну ідентифікацію патологічних зон на рентгенографічних та томографічних знімках, системи автономного транспорту отримують вигоду від здатності цих моделей до надійної детекції об’єктів у складних умовах освітлення та при наявності візуальних перешкод [2].

Різні підходи до виявлення об’єктів мають суттєві відмінності у продуктивності, точності та обчислювальній складності. Аналіз цих характеристик дозволяє вибрати оптимальний метод для конкретного застосування. Сучасні тенденції спрямовані на створення гібридних архітектур, що поєднують переваги різних підходів. Наприклад, YOLO з механізмами уваги або трансформери з ефективними згортковими шарами [8]. Порівняльний аналіз розглянутих моделей продемонстровано в таблиці 1.1.

Загалом, кожен алгоритм має свої переваги та обмеження. Вибір залежить від конкретних вимог до точності, швидкості та доступних обчислювальних ресурсів. Сучасні дослідження спрямовані на поєднання переваг цих підходів у нових архітектурах [1, 2, 3, 4].

Таблиця 1.1 – Порівняльна таблиця для методів виявлення об’єктів [6, 16, 19, 20]

Модель	Тип	mAP	FPS	Переваги	Недоліки
YOLOv11	Одноетапний	72%	145	Висока швидкість	Вимагає значних обчислювальних ресурсів
Faster R-CNN	Двоетапний	75%	7	Висока точність	Низька швидкість
SSD	Одноетапний	72%	59	Добрий баланс швидкості/точності	Витрати пам’яті
RetinaNet	Одноетапний	80%	14	Краща точність	Високі обчислювальні витрати

1.3 Популярні підходи до реалізації та огляд платформ з готовими рішеннями для реалізації

Сучасний етап розвитку технологій комп’ютерного зору характеризується появою різноманітних програмних рішень та фреймворків, що значно спрощують процес розробки та впровадження алгоритмів обробки зображень. Серед найбільш поширених інструментів особливе місце займає бібліотека OpenCV (Open Source Computer Vision Library), яка стала стандартом у галузі комп’ютерного зору. OpenCV пропонує комплексне рішення для широкого спектра задач, включаючи аналіз зображень, детекцію об’єктів, розпізнавання облич та трекінг руху. Архітектура бібліотеки дозволяє ефективно використовувати як класичні алгоритми обробки зображень, так і сучасні методи глибокого навчання. Важливою перевагою OpenCV є її інтеграція з провідними фреймворками машинного навчання, такими як TensorFlow та PyTorch, що відкриває можливості для створення гібридних рішень [18]. Паралельно з OpenCV розвиваються спеціалізовані платформи для комп’ютерного зору. TensorFlow Object Detection API пропонує потужні інструменти для створення систем детекції на основі нейронних мереж. PyTorch Vision забезпечує комплексний набір функцій для роботи з візуальними даними в екосистемі PyTorch. Платформа MMDetection спеціалізується на реалізації алгоритмів детекції об’єктів з використанням глибокого навчання, тоді як Detectron2 від

Facebook AI Research орієнтований на підтримку наукових досліджень у галузі комп'ютерного зору.

Сучасна архітектура детекції об'єктів модель YOLO (You Only Look Once), яка реалізує інноваційний підхід до обробки зображень. Відмінною рисою цієї архітектури є здатність аналізувати зображення за один прохід через нейронну мережу, що забезпечує значну перевагу в швидкості роботи порівняно з традиційними методами, які використовують складні багатоетапні конвеєри обробки [5]. Аналіз останніх оновлень YOLOv11 виявляє ряд ключових вдосконалень. Серед них – оптимізація використання обчислювальних ресурсів, що дозволяє досягати високої продуктивності на різноманітних апаратних платформах, вдосконалення механізмів масштабування для роботи з об'єктами різних розмірів, а також підвищення стабільності роботи в умовах обмеженої яскравості або часткового перекриття об'єктів [20]. Аналіз останніх оновлень YOLOv11 виявляє ряд ключових вдосконалень. Серед них – оптимізація використання обчислювальних ресурсів, що дозволяє досягати високої продуктивності на різноманітних апаратних платформах, вдосконалення механізмів масштабування для роботи з об'єктами різних розмірів, а також підвищення стабільності роботи в умовах обмеженої яскравості або часткового перекриття об'єктів [20]. Ці характеристики роблять YOLOv11 особливо привабливим рішенням для завдань, що вимагають обробки у реальному часі, таких як системи відеоспостереження, автономні транспортні засоби та промислові системи контролю якості.

Одним із сучасних фреймворків для реалізації алгоритмів глибокого навчання особливе місце займає PyTorch, що став одним із найбільш поширених інструментів у наукових дослідженнях та промислових рішеннях. Цей фреймворк відзначається унікальним поєднанням гнучкості архітектури, високої продуктивності та інтуїтивного інтерфейсу програмування, що робить його особливо привабливим для розробки складних моделей комп'ютерного зору [21]. Архітектура PyTorch ґрунтується на динамічних обчислювальних графах, що надає дослідникам можливість ефективно експериментувати з різними

структурами нейронних мереж. Ця характеристика є особливо цінною для задач комп'ютерного зору, де часто виникає необхідність адаптувати моделі під специфічні вимоги завдання. Фреймворк забезпечує повний цикл розробки – від створення прототипів до промислового впровадження [21]. Torchvision значно спрощує процес створення та навчання моделей комп'ютерного зору, надаючи готові рішення для типових завдань і водночас дозволяючи гнучку модифікацію архітектур під конкретні потреби. Модуль постійно оновлюється, включаючи останні досягнення у галузі глибокого навчання, що робить його незамінним інструментом для сучасних досліджень [21].

Фреймворк TensorFlow, що використовують для реалізації алгоритмів машинного навчання, розроблений Google Brain Team. Цей фреймворк сформував цілісну екосистему для розробки та впровадження моделей штучного інтелекту, що включає спеціалізовані інструменти для комп'ютерного зору, обробки природної мови та інших напрямків AI [22]. Архітектура TensorFlow ґрунтується на концепції статичних обчислювальних графів, що забезпечує високу ефективність обчислень та масштабованість рішень. Для задач комп'ютерного зору особливе значення має TensorFlow Object Detection API - спеціалізований інструментарій, що надає готові реалізації сучасних архітектур детекції об'єктів. Серед них варто відзначити Faster R-CNN, який демонструє високу точність за рахунок використання регіонів пропозицій мереж; SSD (Single Shot MultiBox Detector), що забезпечує гарний баланс між швидкістю та точністю; та EfficientDet, який оптимізований для роботи на ресурсообмежених пристроях [22]. Важливою перевагою TensorFlow є підтримка розподілених обчислень, що дозволяє ефективно використовувати кластерні системи для навчання складних моделей [22].

Платформа Hugging Face – популярна платформа, яка пропонує бібліотеки та моделі для обробки природної мови, комп'ютерного зору та інших завдань штучного інтелекту (ШІ). Вона містить великий репозиторій попередньо навчених моделей, які можна легко адаптувати до конкретних потреб. Сервіс

Hugging Face також забезпечує інтеграцію з PyTorch і TensorFlow, що дозволяє розробникам швидко тестувати та розгортати моделі машинного навчання [23].

Отже, наведені платформи відіграють ключову роль у сучасному комп'ютерному зорі, забезпечуючи широкий спектр рішень для автоматичної обробки зображень та детекції об'єктів. Вибір конкретного інструменту залежить від вимог до продуктивності, точності та зручності інтеграції в проєкти машинного навчання. Hugging Face – платформа на якій доступні моделі YOLO, SSD, RetinaNet та Faster R-CNN, що дозволяє легко інтегрувати їх у розгортання нейромережевих сервісів [23]. Вона надає можливість використовувати тисячі публічно доступних датасетів для тренування моделей. Це особливо важливо для дослідників, які працюють над задачами класифікації, сегментації, генерації тексту та інших аспектів штучного інтелекту.

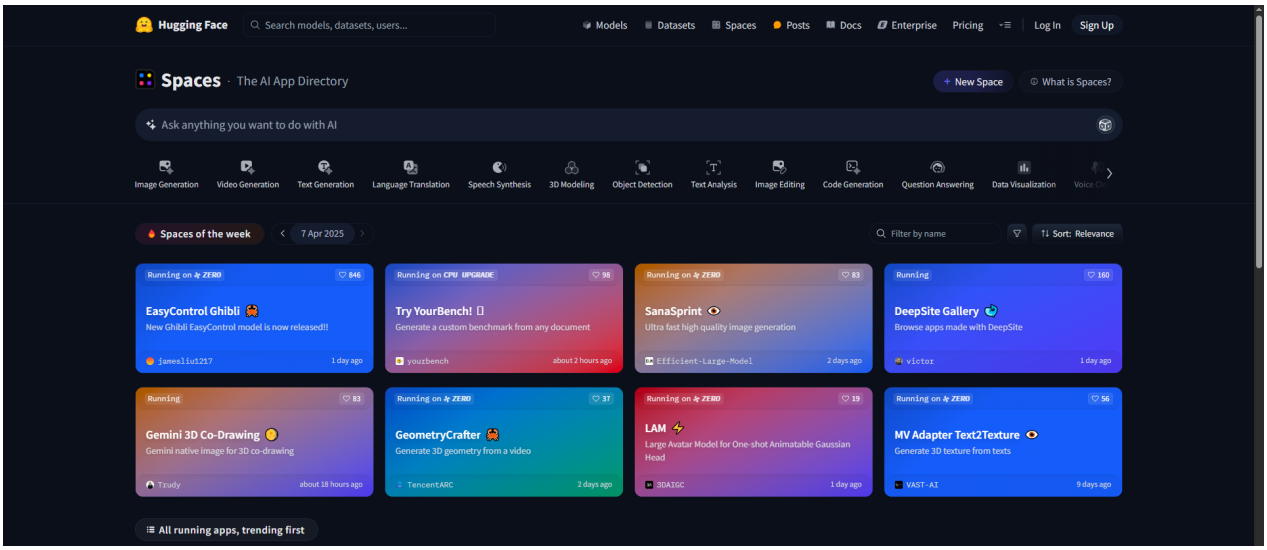

Рисунок 1.13 – Вигляд головної сторінки Hugging Face [23]

1.4 Постановка завдання на розробку

Відповідно до поставленої мети дослідження в якості об'єктів дослідження обрано дві сучасні архітектури нейронних мереж – YOLO та Faster R-CNN, а також два традиційні методи – HOG і SIFT. Основна увага приділяється вивченню характеристик цих алгоритмів у контексті їх практичного застосування для розв'язання задач комп'ютерного зору.

Для тестування ефективності детекції об’єктів у межах кваліфікаційної роботи було обрано дві згорткові нейромережі та два традиційних методи. Серед згорткових нейромереж розглядатимуться такі:

1. YOLO (You Only Look Once) – одна з найшвидших моделей одноетапного виявлення об’єктів, яка дозволяє розпізнавати об’єкти у реальному часі.

2. Faster R-CNN – двоетапний детектор, що забезпечує високу точність завдяки механізму генерації регіонів пропозицій.

За результатами огляду, для аналізу роботи традиційних методів було відібрано:

1. HOG (Histograms of Oriented Gradients) – класичний метод аналізу градієнтів зображення для виявлення контурів об’єктів.

2. SIFT (Scale-Invariant Feature Transform) – алгоритм, що знаходить ключові точки зображення, стійкі до змін масштабу та освітлення.

Розвиток технологій машинного бачення для автономного транспорту пройшов еволюційний шлях від традиційних методів виділення ознак до складних нейромережевих архітектур. Класичні алгоритми, такі як HOG та SIFT, заклали основу для аналізу візуальних даних, демонструючи хороші результати у статичних умовах. Однак їх обмежена адаптивність до змін у зовнішньому середовищі стимулювала розвиток глибокого навчання. Сучасні нейромережеві архітектури, зокрема YOLO та Faster R-CNN, відкрили нові можливості для систем автопілотування. YOLO, як представник одноетапних детекторів, забезпечує високу швидкість обробки, що є критично важливим для прийняття рішень у реальному часі. Faster R-CNN, будучи двоетапним детектором, досягає вищої точності за рахунок більш детального аналізу регіонів інтересу. Вибір цих конкретних методів для дослідження обумовлений їх широким застосуванням у промислових рішеннях автономного транспорту та наявністю перевірених реалізацій для роботи з дорожніми сценами. Аналіз їх продуктивності на стандартних наборах даних, таких як KITTI, дозволяє отримати об’єктивну оцінку придатності для вирішення задач детекції у реальних умовах.

РОЗДІЛ 2
ВИБІР ТА ОПТИМІЗАЦІЯ АЛГОРИТМІВ ДЛЯ ВИЯВЛЕННЯ ОБ'ЄКТІВ

2.1 Формування вимог та процедури підготовки вхідних даних

Для проведення дослідження ефективності методів детекції об'єктів було обрано міжнародно визнаний датасет KITTI, який є стандартом для оцінювання алгоритмів комп'ютерного зору в автономних транспортних системах. Вибір даного набору даних обґрунтований його комплексним характером, що включає зображення високої роздільної здатності, отримані в різних умовах освітлення та погодних умовах. Датасет містить детально анотовані об'єкти, такі як транспортні засоби, пішоходи та велосипедисти, що відповідає основним класам об'єктів, які необхідно розпізнавати в задачах автономного водіння.

Найперші демонстрації ефективності доповнення даних походять від простих перетворень, таких як горизонтальне віддзеркалення, доповнення колірного простору та випадкове кадрування. Ці перетворення кодують багато інваріантів, обговорюваних раніше, які створюють труднощі для завдань розпізнавання зображень. Доповнення, перелічені в цьому огляді, такі: геометричні перетворення, перетворення колірного простору, фільтрувальні ядра, змішування зображень, випадкове стирання (cropping), доповнення простору ознак [3]. Відзеркалювання горизонтальної осі набагато поширеніше, ніж відзеркалювання вертикальної осі. Порівняно з іншими видами аугментацій, віддзеркалення є однією з найпростіших для реалізації та виявилася корисною на таких наборах даних, як KITTI [3].

Дані цифрового зображення зазвичай кодуються як тензор з розмірністю (висота × ширина × колірні канали). Аугментація в кольоровому просторі – це ще одна практична та легка у реалізації стратегія. Найпростіші колірні перетворення включають ізоляцію одного каналу (R, G або B). Зображення можна швидко перетворити, залишивши лише один канал і додавши нульові матриці інших каналів. Крім того, значення RGB можна змінювати за

допомогою простих матричних операцій, щоб збільшити або зменшити яскравість. Більш складні колірні перетворення отримують шляхом побудови колірної гістограми, що описує зображення. Зміна значень інтенсивності в цих гістограмах призводить до змін освітлення, таких що використовується в програмах для редагування фотографій (рис. 2.1) [3].

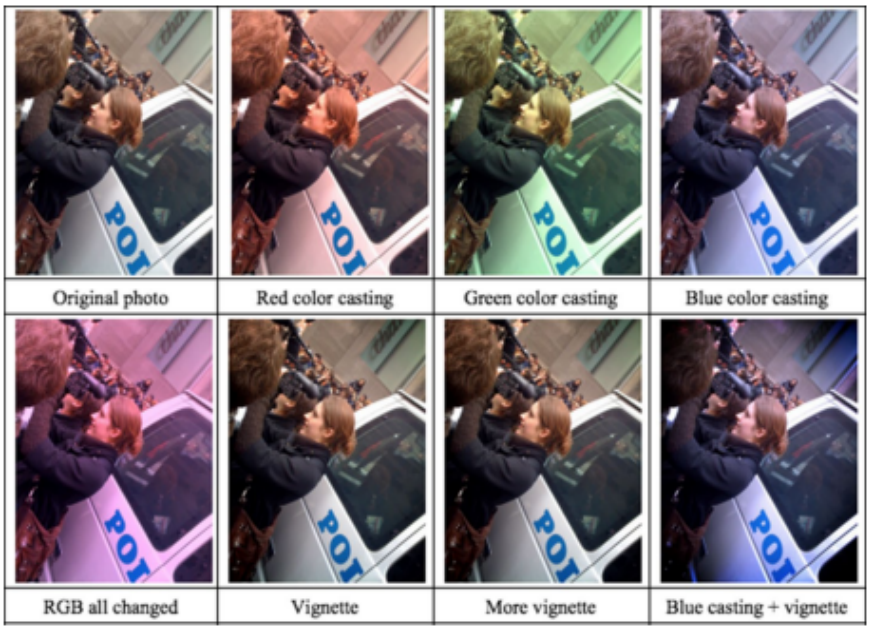

Рисунок 2.1 – Приклад аугментації в колірному просторі [3]

Обрізання зображень може бути використане як практичний крок обробки даних зображень зі змішаними розмірами висоти та ширини шляхом обрізання центральної ділянки кожного зображення. Крім того, випадкове обрізання може імітувати зсув. Різниця між цими двома підходами полягає в тому, що обрізання зменшує розмір зображення (наприклад, з (256×256) до (224×224)), тоді як зсув зберігає просторові розміри. Залежно від рівня обрізання, це перетворення може не зберігати мітку (рис. 2.2) [3].

Аугментація поворотом виконується обертанням зображення вліво або вправо на кут від 1° до 359°. Безпека цього методу сильно залежить від кута повороту [3].

Зміщення зображень ліворуч, праворуч, вгору або вниз може бути дуже корисним перетворенням, щоб уникнути позиційного зміщення в даних.

Наприклад, якщо всі зображення в наборі даних центровані, що є поширеним явищем у наборах даних для розпізнавання облич, це вимагатиме тестування моделі також на ідеально центрованих зображеннях. Оскільки оригінальне зображення переміщується в певному напрямку, решта простору може бути заповнена або постійним значенням, таким як 0 с або 255 с, або випадковим або гаусовим шумом. Це доповнення зберігає просторові розміри зображення після аугментації [3].

Введення шуму полягає у введенні матриці випадкових значень, зазвичай отриманих з гаусового розподілу. Додавання шуму до зображень може допомогти CNN вивчати більш надійні ознаки [3].

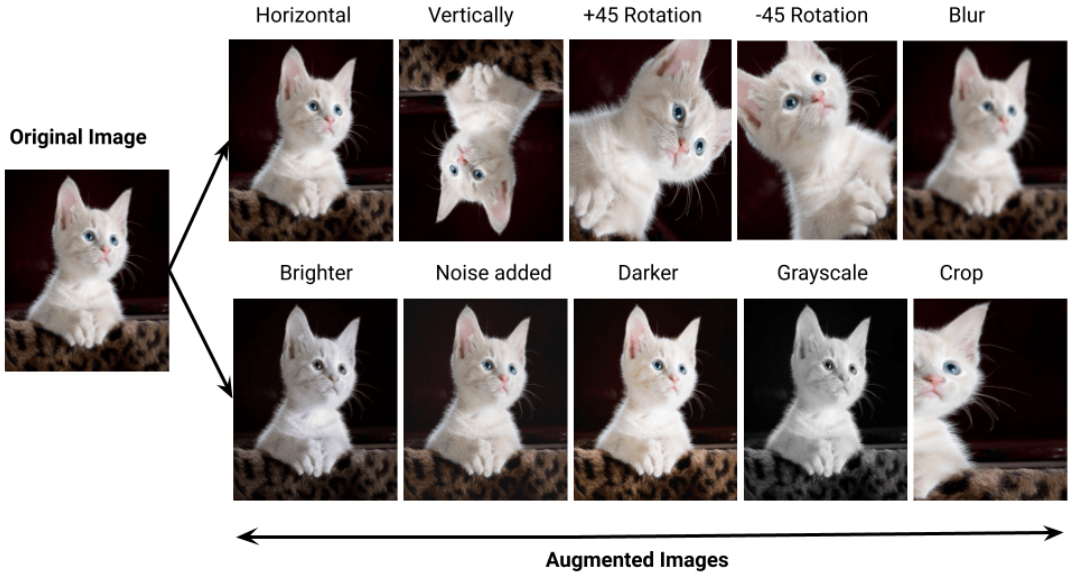

Рисунок 2.2 – Види аугментації даних

Геометричні перетворення є дуже хорошими рішеннями для позиційних зміщень, присутніх у навчальних даних. Існує багато потенційних джерел зміщення, які можуть відокремити розподіл навчальних даних від тестових даних. Існує багато бібліотек для обробки зображень, які роблять такі операції, як горизонтальне перевертання та обертання, безпроблемними для початку. Нарешті, у багатьох охоплених областях застосування, таких як аналіз медичних зображень, зміщення, що віддаляють навчальні дані від тестових даних, є

складнішими, ніж позиційні та трансляційні дисперсії. Тому сфера застосування геометричних перетворень є відносно обмеженою [3].

Дані зображення кодуються в 3 складені матриці, кожна з яких має розмір висота × ширина. Ці матриці представляють значення пікселів для окремого значення кольору RGB. Зміщення освітлення є одними з найчастіших проблем, що виникають у проблемах розпізнавання зображень. Отже, ефективність перетворень колірного простору, також відомих як фотометричні перетворення, досить інтуїтивно зрозуміла. Швидким виправленням надмірно яскравих або темних зображень є циклічний перегляд зображень та зменшення або збільшення значень пікселів на постійне значення. Ще однією швидкою маніпуляцією з колірним простором є сплайсинг окремих кольорових матриць RGB. Інше перетворення полягає в обмеженні значень пікселів певним мінімальним або максимальним значенням. Внутрішнє представлення кольору в цифрових зображеннях піддається багатьом стратегіям доповнення.

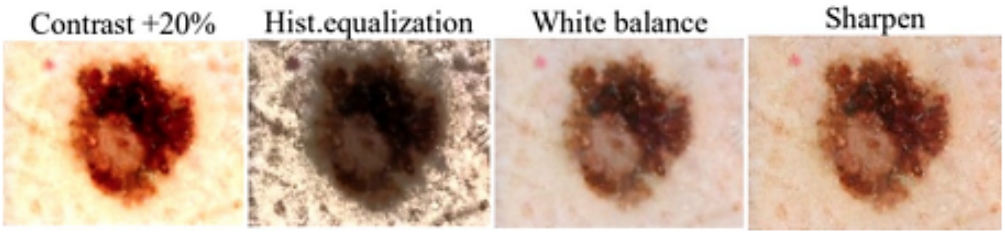

Рисунок 2.2 – Перетворення колірного простору [3]

Для підвищення узагальнюючої здатності моделі під час роботи з датасетом KITTI було застосовано аугментацію зображень за допомогою бібліотеки Albumentations. Нижче наведено докладний опис кожного використаного перетворення, його вплив на дані та обґрунтування параметрів. У лістингу 2.1 присутні такі аугментації:

– A.HorizontalFlip() – дзеркальне відображення зображення по горизонталі; це перетворення допомагає збільшити різноманітність даних, імітуючи зміну точки огляду;

– A.RandomBrightnessContrast() змінює яскравість та контрастність зображення, що імітує різні умови освітлення;

– A.MotionBlur() імітує ефект розмиття через рух камери або об’єктів у кадрі;

– A.GaussNoise() додає випадковий гаусів шум до зображення, що імітує артефакти камери або низьку якість запису;

– A.RandomFog() додає ефект туману, що імітує погані погодні умови.

На рисунках 2.4-2.6 показано приклади застосування аугментацій до зображень з набору даних KITTI.

а)

б)

в)

Рисунок 2.4 – а) оригінальне зображення з датасету KITTI [3]; б) після накладання шуму; в) після зміни контрастності, яскравості та відзеркалювання

Лістинг 2.1 – Уривок коду для аугментації

```
import albumentations as A

self.augmentation = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.3),
    A.MotionBlur(blur_limit=5, p=0.2),
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.2),
    A.RandomFog(fog_coef_lower=0.1, fog_coef_upper=0.3, p=0.1),
])
```

а)

б)

в)

Рисунок 2.5 – а) оригінальне зображення; б) гаусове розмиття; в) гаусове розмиття та відзеркалення

а)

б)

Рисунок 2.6 – а) оригінальне зображення; б) імітація туману

2.2 Алгоритми виявлення об’єктів на основі традиційних методів

Реалізація алгоритму SIFT у контексті сучасних технологій виявлення об’єктів для автомобілів з автопілотом має велике значення для підвищення точності системи комп’ютерного зору. Автономні транспортні засоби вимагають швидкої та надійної детекції об’єктів, таких як інші автомобілі, пішоходи, дорожні знаки та різні перешкоди. Саме алгоритм SIFT, що базується на виділенні стійких ознак та їх подальшому зіставленні, дозволяє досягти високої точності розпізнавання об’єктів незалежно від змін освітлення, повороту та масштабу [9, 19].

На етапі попередньої обробки даних з камер автомобіля здійснюється завантаження та підготовка зображень. Автопілот використовує камери з високої роздільної здатності для отримання даних у реальному часі, що потребує алгоритмів, здатних до оперативної обробки та аналізу. Попередня обробка зображень включає масштабування, фільтрацію шумів та перетворення в градації сірого для підвищення ефективності виділення ключових ознак. Це

дозволяє скоротити обсяг інформації та зосередитися на найважливіших особливостях сцени.

Застосування SIFT у системах автономного керування автомобілем здійснюється шляхом виділення ключових точок на зображенні. Використання функції cv2.SIFT_create() фреймворка OpenCV дозволяє розпізнати характерні елементи дорожнього середовища, такі як розмітка, дорожні знаки та перешкоди. Дескриптори, що обчислюються для кожного зображення, є ключовими в процесі порівняння з шаблонними даними, що зберігаються в базі знань автопілота. Це дає змогу з високою точністю визначати об’єкти, навіть якщо вони частково перекриті або дещо змінені за формою.

Критично важливим компонентом системи є модуль пошуку відповідностей між ознаками поточного кадру та еталонними шаблонами. FLANN Matcher забезпечує ефективне зіставлення ознак, що дозволяє швидко ідентифікувати інші транспортні засоби та перешкоди. Використання kD-дерев у FLANN Matcher значно прискорює процес знаходження відповідностей, а тест Лоу дозволяє відсіяти помилкові збіги, що могло б призвести до некоректної інтерпретації сцени автомобілем.

Щоб мінімізувати похибки розпізнавання, здійснюється оцінювання якості відповідностей за допомогою алгоритму RANSAC. Побудова гомографії між поточним кадром та шаблоном дозволяє встановити точне положення об’єкта в просторі. У контексті автомобілів із автопілотом цей метод допомагає уникнути неправильного розпізнавання об’єктів, таких як дорожні знаки, які можуть мати пошкодження або бути частково перекритими іншими об’єктами [9, 3].

Візуалізація та збереження аналітичних даних забезпечують можливість подальшого аналізу продуктивності алгоритму. Результати обробки кадрів у реальному часі дозволяють оцінювати точність розпізнавання ключових об’єктів у дорожньому середовищі, що сприяє покращенню роботи автономного керування. Дані про кількість знайдених відповідностей та виявлених класів об’єктів можуть бути використані для подальшого вдосконалення моделі комп’ютерного зору.

За допомогою представленої на рис. 2.7 діаграмі пакетів продемонстровано результати проєктування рішення для реалізації виявлення об’єктів на основі методу SIFT. Компоненти системи взаємодіють у цілісному процесі анотації, обробки та оцінки зображень з набору даних KITTI. Основний блок, який зібраний у межах пакету KITTISIFTTester, забезпечує послідовну обробку входів від датасету до кінцевої оцінки отриманих результатів, що дозволяє виявляти об’єкти за допомогою алгоритму SIFT.

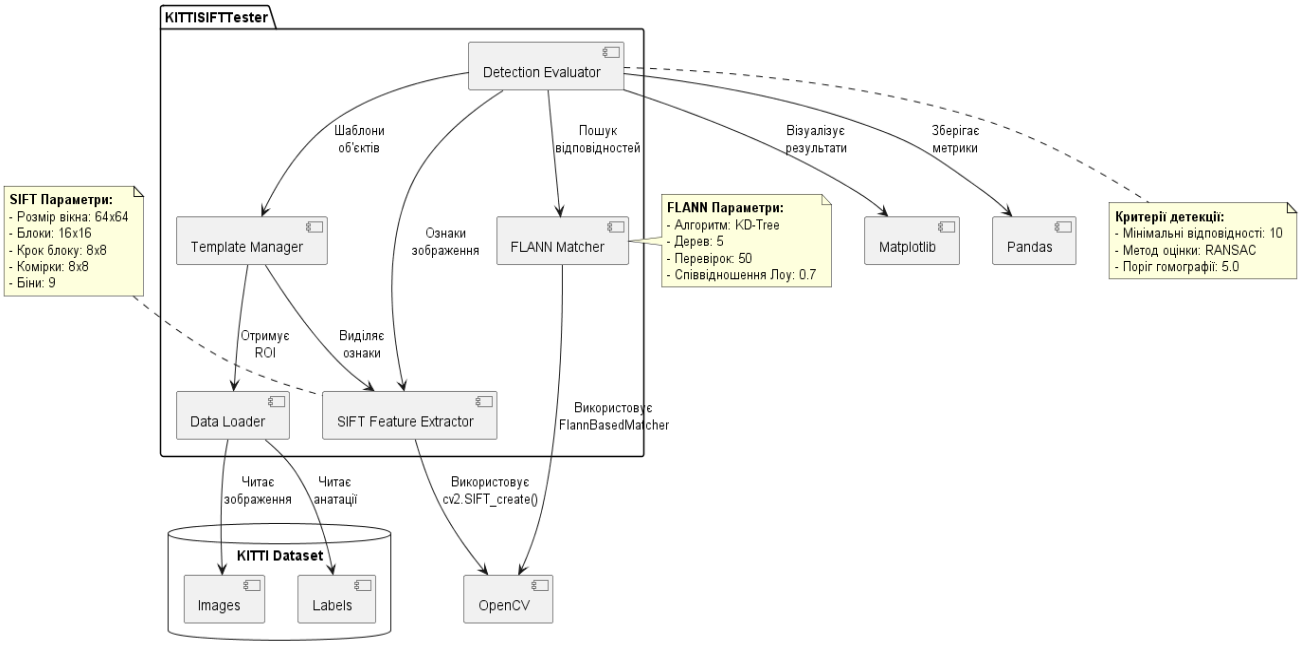

Рисунок 2.7 – Діаграма пакетів для системи виявлення об’єктів за допомогою SIFT-дескриптора

Першим елементом є Data Loader, що відповідає за читання зображень та відповідних анотацій із бази даних KITTI. Завдяки цьому модулю система отримує первинні дані, необхідні для подальшої обробки – реальні кадри та пов’язані з ними позначки, що відображають інформацію про об’єкти на зображеннях.

Далі Template Manager виконує роль попереднього аналізатора, який із завантажених даних відокремлює та виділяє регіони інтересу (ROI). Цей етап є критичним для подальшої обробки: отримані шаблони передаються до модуля

SIFT Feature Extractor, який базується на функції OpenCV cv2.SIFT_create(). Цей елемент системи здійснює виокремлення характерних ознак зображення, використовуючи спеціалізовані параметри, зокрема розмір вікна 64x64 пікселів, блоки розміром 16x16, з кроком 8 пікселів, комірки 8x8 та розбивку ознак на 9 бінів. Таке налаштування забезпечує високу стійкість виявлення ознак до змін масштабу та освітлення, що є важливим для подальшого аналізу. Відповідний уривок коду з виявленням характерних ознак наведено в лістингу 2.2.

Лістинг 2.2 – Уривок коду для виявлення характерних ознак у зображень

```
def extract_template_features(self, img_path, label_path):
    img = cv2.imread(str(img_path))
    if img is None:
        return None, []

    with open(label_path, 'r') as f:
        lines = f.readlines()

    templates = []
    for line in lines:
        parts = line.strip().split()
        cls = parts[0]
        if cls in self.classes:
            x1, y1, x2, y2 = map(float, parts[4:8])
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

            roi = img[y1:y2, x1:x2]
            if roi.size == 0:
                continue

            gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
            kp, des = self.sift.detectAndCompute(gray, None)

            if des is not None:
                templates.append({
                    'class': cls,
                    'keypoints': kp,
                    'descriptors': des,
                    'bbox': (x1, y1, x2, y2)
                })
    return img, templates
```

Наступний компонент, FLANN Matcher, інтегрується з OpenCV через модуль FlannBasedMatcher. Його завдання полягає у зіставленні виділених ознак, де використовується алгоритм KD-Tree. Параметри цього модуля, зокрема кількість дерев задається значенням 5, кількість перевірок – 50, а співвідношення Лоу встановлено на рівні 0.7, що сприяє більш точному та надійному визначенню

відповідностей між ознаками. Відповідний лістинг з уривком коду для зіставлення виокремлених ознак наведено в додатку А.

Важливу роль відіграє Detection Evaluator, який отримує шаблони об’єктів від Template Manager, ознаки зображень від SIFT Feature Extractor та інформацію про співпадіння від FLANN Matcher. Цей елемент проводить оцінку детекції із застосуванням методів, що включають критерії – мінімальна кількість відповідностей (мінімум 10), використання алгоритму RANSAC для оцінки, а також встановлення порогу гомографії на рівні 5. Результати оцінки зберігаються за допомогою пакету Pandas, а їх візуалізація здійснюється засобами бібліотеки Matplotlib, що дозволяє отримати як числові, так і графічні дані, необхідні для аналізу продуктивності моделі. Лістинг з уривком коду для виявлення характерних ознак для зображень представлено в додатку Б.

Метод HOG для систем автономного керування автомобілями також може бути інтегрований для побудови надійної системи детекції об’єктів, що працює у складних дорожніх умовах. У контексті автопілотів особливу увагу приділяють точності розпізнавання транспортних засобів, пішоходів та інших учасників руху. Реалізація такого підходу полягає в побудові модульної архітектури, що охоплює етапи завантаження даних, попередньої обробки, обчислення HOG-ознак, тренування класифікатора та подальшої детекції об’єктів на тестових зображеннях.

Архітектура програмного рішення для застосування методу HOG багато в чому перегукується з рішеннями для SIFT-декриптора. Першою складовою реалізації є ініціалізація обчислень за методом HOG за допомогою OpenCV. Основними параметрами, що задаються користувачем, є розмір вікна, розміри блоку, крок блоку, розмір комірки та кількість бінів. Завдяки відповідному налаштуванню цих параметрів система здатна екстрагувати локальні особливості зображення, що забезпечують точне представлення форми та орієнтації об’єктів навіть у випадку змін масштабу чи перспективи. Застосування стандартного вікна розміром 64×64 пікселів дозволяє привести вхідні регіони до єдиного формату, що критично для подальшого аналізу.

Надалі, дані завантажуються з датасету KITTI, що містить зображення з польотів автомобільних камер та відповідні анотації. Реалізація включає побудову механізму читання зображень із заданої директорії, одночасне зчитування інформації про об’єкти (кордонні координати) та формування регіонів інтересу (ROI). Отримані регіони за необхідності змінюються за розміром до встановленого параметру вікна, після чого для кожного регіону проводиться перетворення зображення в градації сірого, що є оптимальним для подальшого обчислення HOG-ознак [14].

Важливий етап реалізації полягає в обчисленні HOG-ознак за допомогою функцій HOGDescriptor(), що входять до складу OpenCV. Отриманий вектор ознак детально описує локальну орієнтацію градієнтів у кожному регіоні зображення, що дозволяє відтворити локальні характеристики форми об’єктів. Після цього, отримані ознаки використовуються для навчання класифікаційної моделі. У розглянутій реалізації як класифікатор застосовується лінійний SVM з використанням LinearSVC, а також стандартний масштабувач даних (StandardScaler) для нормалізації вхідних ознак. Навчання проводиться на зразках, що були попередньо оброблені з датасету KITTI, що дозволяє моделі врахувати специфіку різних класів транспортних засобів, пішоходів, а також інших об’єктів, важливих для автопілота.

Для реалізації режиму детекції на тестовому зображенні застосовується підхід скануючого вікна, коли зображення аналізується на декількох масштабах. Для кожного регіону проводиться обчислення HOG-ознаків та нормалізація отриманого вектора, після чого класифікатор розраховує значення функції рішення. Якщо отримане значення впевненості перевищує встановлений поріг, регіон вважається позитивним спрацьовуванням.

Завершальним етапом реалізації є візуалізація отриманих результатів для аналізу роботи моделі. За допомогою бібліотеки Matplotlib результати детекції відображаються в вигляді зображень із нанесеними прямокутниками, що охоплюють класифіковані об’єкти, а також позначками, що містять інформацію про класи та значення впевненості. Отримані дані, включаючи показники

точності та розподіли класифікації, можуть бути збережені для подальшого аналізу з використанням Pandas, що дозволяє проводити оцінку ефективності системи. Результати проєктування програмної системи зображено в вигляді діаграми пакетів на рис. 2.8.

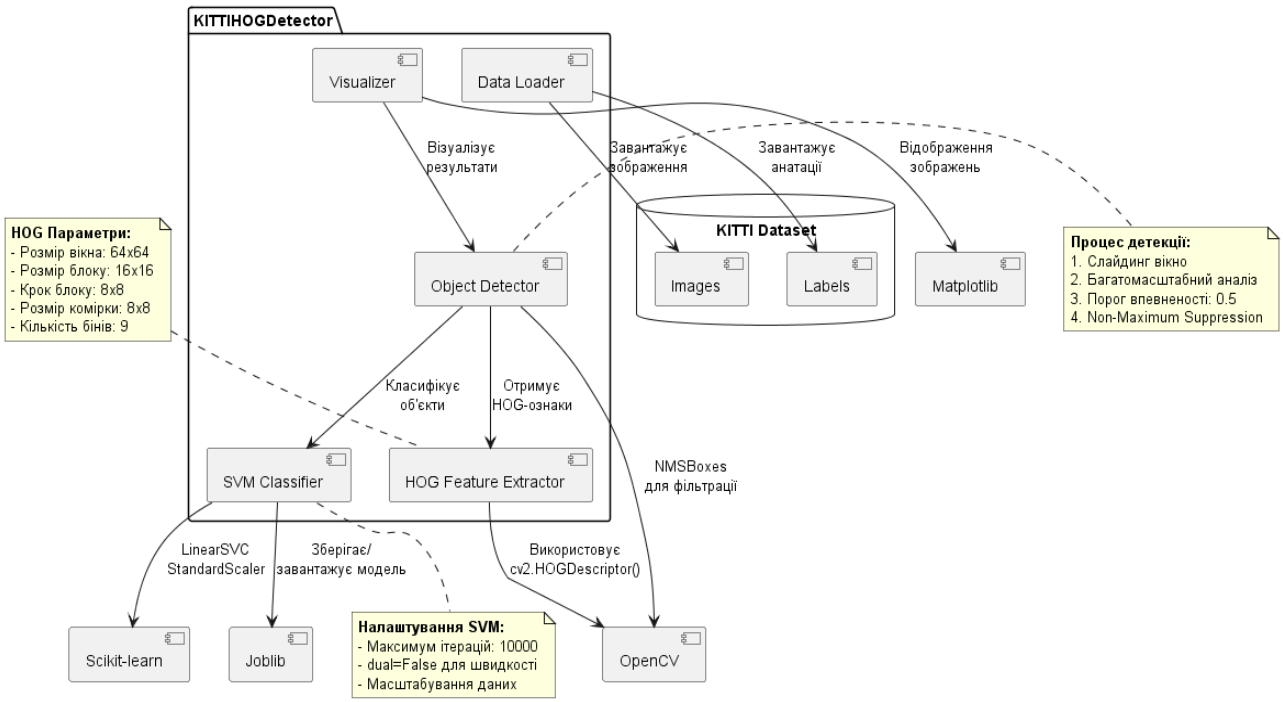

Рисунок 2.8 – Діаграма пакетів для системи виявлення об’єктів за допомогою метода HOG

Архітектура KITTIHOGDetector забезпечує повноцінний процес виявлення об’єктів за допомогою HOG-ознак та класифікатора SVM. Її основні компоненти взаємодіють у чітко визначеному конвеєрі, який охоплює завантаження даних, обчислення ознак, класифікацію та візуалізацію результатів.

Компонент Data Loader відповідає за завантаження зображень із KITTI Dataset та відповідних анотацій. Цей модуль читає файли, що містять інформацію про об’єкти на кадрах, включаючи їх розташування та класову приналежність. Завдяки цьому забезпечується правильна ініціалізація даних для подальшого аналізу. Відповідний уривок програмного коду, який займається завантаженням датасету, наведено в лістингу 2.3.

Лістинг 2.3 – Уривок коду для завантаження датасету

```
def load_kitti_data(self, data_dir, sample_limit=None):
    base_path = Path(data_dir)
    images = []
    labels = []
    bboxes = []

    img_dir = base_path / 'data_object_image_2' / 'training' / 'image_2'
    label_dir = base_path / 'data_object_label_2' / 'training' / 'label_2'

    if not img_dir.exists() or not label_dir.exists():
        raise FileNotFoundError("Не знайдено папки з даними KITTI")

    img_files = sorted(img_dir.glob('*.png'))
    label_files = sorted(label_dir.glob('*.txt'))

    if sample_limit:
        img_files = img_files[:sample_limit]
        label_files = label_files[:sample_limit]

    for img_path, label_path in zip(img_files, label_files):
        img = cv2.imread(str(img_path))
        if img is None:
            continue

        with open(label_path, 'r') as f:
            annotations = [line.strip().split() for line in f.readlines()]

        for ann in annotations:
            cls = ann[0]
            if cls in self.classes:
                bbox = list(map(float, ann[4:8]))
                x1, y1, x2, y2 = map(int, bbox)

                roi = img[y1:y2, x1:x2]
                if roi.size == 0:
                    continue

                roi = cv2.resize(roi, self.win_size)
                images.append(roi)
                labels.append(self.classes[cls])
                bboxes.append((x1, y1, x2, y2))

    return np.array(images), np.array(labels), bboxes
```

Модуль HOG Feature Extractor виконує обчислення ознак на основі гістограми направлених градієнтів, використовуючи cv2.HOGDescriptor(). Він отримує зображення та формує його дескриптор, який містить структурні особливості форми об'єкта. Налаштування таких параметрів, як розмір вікна, блоку та комірки, дозволяє виділити ключові особливості зображення, що робить його придатним для класифікації. Уривок відповідної програмної реалізації наведено в лістингу 2.4.

Лістинг 2.4 – Уривок коду виявлення характерних ознак методом HOG

```
def extract_hog_features(self, imgs):
    features = []
    for img in imgs:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        hog_features = self.hog.compute(gray)
        if hog_features is not None:
            features.append(hog_features.flatten())
    return np.array(features)
```

Компонент SVM Classifier здійснює навчання та класифікацію об’єктів за допомогою LinearSVC(), обробляючи отримані ознаки. Перед передачею даних у модель ознаки нормалізуються за допомогою StandardScaler, що покращує точність класифікації. Навчена модель зберігається у Joblib, що дозволяє використовувати її без необхідності повторного навчання. Відповідний процес відображено в лістингу 2.5.

Лістинг 2.5 – Уривок коду SVM Classifier

```
def train(self, data_dir, model_path='kitti_hog_svm.model'):
    print("Завантаження даних...")
    images, labels, _ = self.load_kitti_data(data_dir, sample_limit=2000)

    print("Виділення HOG ознак...")
    X = self.extract_hog_features(images)
    y = labels

    print("Масштабування даних...")
    self.scaler.fit(X)
    X = self.scaler.transform(X)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    print("Навчання SVM...")
    self.clf = LinearSVC(max_iter=10000, dual=False)
    self.clf.fit(X_train, y_train)
```

Object Detector отримує HOG-ознаки та використовує класифікатор SVM для визначення наявності об’єктів у кадрі. Процес детекції включає слайдинг-вікно, багатомасштабний аналіз та перевірку порогу впевненості. Для фільтрації непотрібних прямокутників застосовується метод Non-Maximum Suppression, що видаляє дублікати та залишає найбільш релевантні області.

Visualizer відповідає за візуалізацію результатів виявлення. Він отримує дані про знайдені об’єкти та використовує Matplotlib для побудови зображень з позначеними об’єктами. Таким чином, кінцевий користувач може оцінити якість роботи алгоритму та проаналізувати отримані результати.

2.3 Алгоритми виявлення об’єктів на основі згорткових нейронних мереж

Реалізація згорткової нейронної мережі YOLO для точного та швидкого розпізнавання транспортних засобів та інших учасників дорожнього руху ґрунтується на використанні сучасних моделей, що відрізняються від традиційних підходів, таких як SIFT чи HOG. Основна ідея цього підходу полягає у тому, що одна мережа, завантажена через бібліотеку Ultralytics, забезпечує одночасну локалізацію та класифікацію об’єктів, що дозволяє здійснювати детекцію в режимі реального часу навіть при змінних умовах освітлення та динамічному середовищі [3, 2].

Ініціалізація системи починається із завантаження попередньо натренованої моделі YOLO (наприклад, yolov8x або yolov11s) через Ultralytics. Цей компонент відповідає за адаптацію вхідних зображень із камер автомобіля до вимог моделі – автоматичне масштабування та стандартизацію даних дозволяють забезпечити точне розпізнавання. Важливою частиною є встановлення відповідності між класами, які визначені в моделі (наприклад, визначені індекси класів), та класами KITTI, що використовуються для оцінки якості детекції, зокрема, для категорій «Car», «Truck» і «Pedestrian» [19, 20].

Подальше використання функціональності системи забезпечує завантаження та аналіз анотацій із KITTI Dataset. Модуль завантаження даних відповідає за читання зображень та парсинг текстових файлів, у яких зберігається інформація про координати bounding boxes для кожного об’єкта. Завдяки цьому система отримує ground truth, що пізніше використовується для порівняння з результатами детекції. Для обчислення показників якості детекції, таких як Intersection over Union (IoU), precision, recall та F1-score, застосовуються функції з OpenCV разом із алгоритмом Non-Maximum Suppression для ефективного усунення дублювання результатів [20].

Ключовою особливістю реалізації є безпосередня обробка зображень через модель YOLO, яка генерує набір детекцій. Кожна детекція містить координати bounding box, значення довіри (confidence) та призначений клас об’єкта. Після цього отримані результати проходять етап візуалізації, де за допомогою бібліотек Matplotlib та Pandas сформовано графічні інтерпретації – зображення із накладеними прямокутниками, де реальні анотації позначені, наприклад, червоним, а детекції – зеленим, а також створено зведені таблиці з метриками продуктивності системи [19, 20].

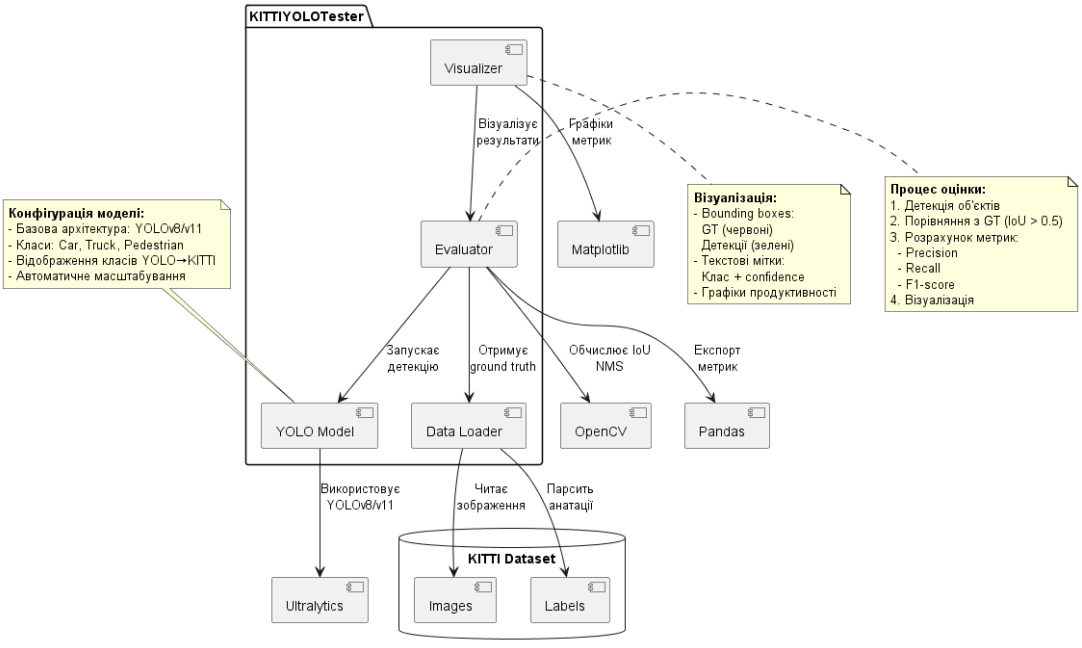

Рисунок 2.9 – Архітектура YOLO, реалізованого через Ultralytics

Першим і найважливішим компонентом є сама модель YOLO. Завдяки інтеграції з бібліотекою Ultralytics використовується один з варіантів сучасної архітектури (наприклад, YOLOv8 або YOLOv11). Ця модель виконує одночасне визначення (локалізацію) та класифікацію об’єктів на зображенні. Вона автоматично масштабує вхідні зображення, адаптуючи їх до внутрішніх параметрів моделі, та застосовує попередньо навчені ваги для точного розпізнавання класів, таких як автомобілі, вантажівки і пішоходи. Модель також містить механізми корекції, що дозволяють перевести вихідні індекси класів у відповідні категорії стандарту KITTI, забезпечуючи сумісність з даними, що використовуються для оцінки систем. Код для цього компоненту представлений в лістингу 2.6.

Лістинг 2.6 – Уривок коду моделі

```
def __init__(self, model_path='yolov8x.pt'):
        self.model = YOLO(model_path)
        self.class_names = self.model.names

        self.kitti_classes = {
            'Car': 0,
            'Truck': 1,
            'Pedestrian': 2,
        }

        self.yolo_to_kitti = {
            0: 'Pedestrian',
            1: 'Cyclist',
            2: 'Car',
            3: 'Motorcycle',
            5: 'Bus',
            7: 'Truck',

        }
```

Другим важливим компонентом є модуль завантаження даних. Він відповідає за отримання зображень та парсинг анотацій із KITTI Dataset. Цей блок системи організовано таким чином, що з одного боку здійснюється читання зображень, а з іншого — парсинг файлів з анотаціями, що містять координати bounding boxes для об’єктів. Коректне завантаження та обробка еталонних (ground truth) даних є критичним для можливості оцінювання роботи моделі на реальних даних.

Компонент оцінки (Evaluator) запускає процес детекції, використовуючи YOLO-модель для аналізу кожного зображення. Отримані результати порівнюються з ground truth анотаціями, що завантажуються модулем даних. Використовуючи функціонал OpenCV, здійснюється обчислення показника Intersection over Union (IoU) для кожної детекції, а алгоритм Non-Maximum Suppression (NMS) застосовується з метою усунення дублювань і вибору найбільш релевантних прямокутників. На основі цих розрахунків система визначає точність (precision), повноту (recall) та F1-міру детекції, що є ключовими метриками оцінки якості виявлення об’єктів. Крім того, результати оцінювання експортуються у форматі таблиць за допомогою Pandas для подальшого аналізу. Реалізація компонента показана в лістингу 2.7.

Лістинг 2.7 – Уривок коду оцінки моделі

```
def _calculate_image_metrics(self, gt_anns, detections, iou_thresh):
        metrics = {}
        matched_gt = set()

        for det in detections:
            cls = det['class']
            if cls not in metrics:
                metrics[cls] = {'tp': 0, 'fp': 0, 'fn': 0}

            best_iou = 0
            best_gt_idx = -1

            for i, gt in enumerate(gt_anns):
                if gt['class'] == cls and i not in matched_gt:
                    iou = self.calculate_iou(det['bbox'], gt['bbox'])
                    if iou > best_iou:
                        best_iou = iou
                        best_gt_idx = i

            if best_iou >= iou_thresh:
                metrics[cls]['tp'] += 1
                matched_gt.add(best_gt_idx)
            else:
                metrics[cls]['fp'] += 1

        for i, gt in enumerate(gt_anns):
            if i not in matched_gt and gt['class'] in metrics:
                metrics[gt['class']]['fn'] += 1

        return metrics
```

Останній блок системи – це модуль візуалізації, який відповідає за представлення кінцевих результатів. Завдяки використанню Matplotlib

створюються зображення, на яких накладаються bounding boxes. Візуально результати відрізняються: реальні анотації відображаються, наприклад, червоним кольором, а детекції – зеленим. Також до зображень накладаються текстові мітки, які містять назву класу та значення достовірності (confidence), що надає змогу швидко оцінити якість детекції та впевненість моделі у своїх прогнозах.

Реалізація алгоритму Faster R-CNN для систем автономного керування автомобілями є критичним етапом побудови сучасної системи детекції об’єктів, що дозволяє точно локалізувати і класифікувати транспортні засоби, пішоходів та інші об’єкти в режимі реального часу навіть у складних умовах дорожнього руху.

Першою складовою розробки є налаштування середовища та конфігурації навчання, що включає визначення апаратного пристрою (як правило, GPU для прискорення процесів навчання), встановлення параметрів, таких як кількість класів, розмір пакету даних (batch size) та кількість епох. Важливим етапом є також підготовка шляху до датасету KITTI, що містить зображення та відповідні анотації, які є необхідними для формування еталону (ground truth) при тренуванні моделі.

На наступному етапі реалізується модуль створення датасету. Для цього розробляється клас, який забезпечує завантаження зображень з файлової системи, читання анотацій з текстових файлів та формування регіонів інтересу (ROI) із заданням bounding boxes. Отримані координати та мітки перетворюються у відповідний формат, сумісний з входом мережі Faster R-CNN, що дозволяє ефективно обчислювати функцію втрат під час навчання. Далі, у модулі створення моделі, за допомогою бібліотеки PyTorch та Torchvision формується архітектура Faster R-CNN. До основних компонентів належать:

Backbone – мережа для екстракції ознак з зображення (у наведеній реалізації використовується MobileNet v2, який забезпечує збалансоване співвідношення між точністю і швидкістю обчислень),

Anchor Generator – модуль для генерації пропозицій регіонів з використанням набору фіксованих масштабів і співвідношень сторін,

ROI Pooler – блок для вибірки ознак із регіонів інтересу з подальшим їх нормуванням до стандартного розміру, та інші специфічні налаштування, що дозволяють моделі адаптуватися під датасет KITTI.

Особливу увагу приділено попередньому навчанню та оптимізації моделі. На цьому етапі застосовуються оптимізатори (наприклад, SGD чи Adam), планувальники швидкості навчання та інші стратегії, що дозволяють зменшити функцію втрат. Навчання моделі проводиться на попередньо оброблених даних з KITTI, що дозволяє повністю врахувати специфіку різних класів транспортних засобів, пішоходів, а також інших об’єктів, що важливі для автопілота.

Для реалізації режиму детекції на тестових зображеннях використовується підхід, за яким зображення подається через модель Faster R-CNN для отримання координат предиктивних bounding boxes, відповідних класів та значень достовірності (confidence). Отримані результати проходять постобробку, включаючи Non-Maximum Suppression, що дозволяє відфільтрувати дублікати детекцій, і порівнюються з ground truth для розрахунку метрик, таких як IoU, precision, recall та F1-score. Завершальним етапом реалізації є візуалізація отриманих результатів для подальшого аналізу роботи моделі. Для цього використовуються бібліотеки Matplotlib та Pillow, що дозволяють накладати на зображення bounding boxes з відповідними мітками класів і значеннями довіри. Результати розрахунків та метрики продуктивності експортуються для подальшої обробки з використанням Pandas, що сприяє глибшому аналізу ефективності системи виявлення об’єктів у реальному дорожньому середовищі.

У розробці системи об’єктного виявлення на базі Faster R-CNN для автономного керування транспортними засобами кожен елемент архітектури виконує свою спеціалізовану задачу, забезпечуючи ефективну обробку даних та отримання високоточних результатів. Компонент «Data Loader» відповідає за завантаження та підготовку даних із датасету KITTI. Він зчитує тренувальні зображення, відповідні їм анотації та тестові зображення, здійснює перетворення

анотацій у формат, сумісний із вхідними даними нейронної мережі, і проводить підготовку тензорів, що дозволяє забезпечити консистентність даних для подальшого навчання. Крім того, у цьому модулі реалізовано базову процедуру розширення даних, що сприяє підвищенню узагальнюючої здатності моделі.

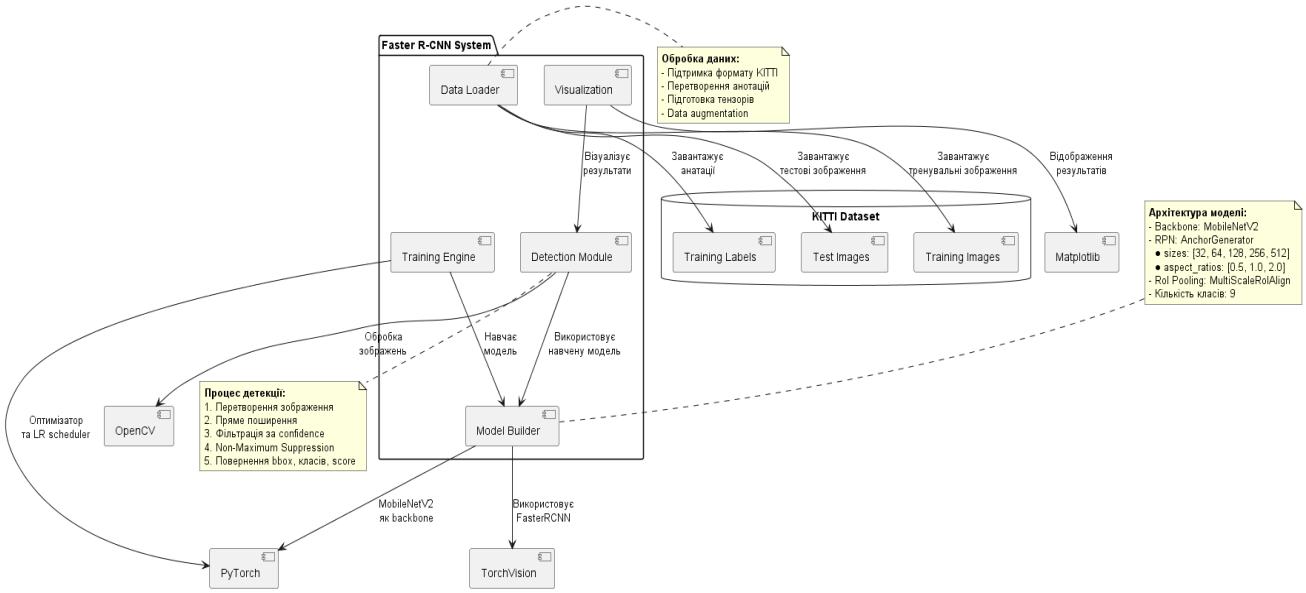

Рисунок 2.10 – Архітектура Faster R-CNN, реалізованого через PyTorch

«Model Builder» – це модуль, у якому формується сама модель Faster R-CNN. Використовуючи бібліотеку TorchVision, він створює архітектуру, засновану на використанні MobileNetV2 як backbone для екстракції ознак. Цей компонент включає генератор якорів (Anchor Generator) із заздалегідь визначеними розмірами та співвідношеннями сторін, а також метод MultiScaleRoIAlign для отримання фіксованих розмірів областей інтересу. Внутрішня конфігурація моделі дозволяє врахувати до 9 різних класів, що відповідає специфіці датасету KITTI, і забезпечити точну локалізацію та класифікацію об’єктів. Код Відповідний процес відображено в лістингу 2.8.

Після завершення навчання «Detection Module» використовує отриману навчену модель для виконання системної детекції об’єктів на тестових зображеннях. Цей компонент відповідальний за перетворення вхідного зображення та його подальше пряме поширення через нейронну мережу. Результати, що містять превалідовані bounding boxes, визначені класи та

значення confidence, фільтруються за допомогою алгоритму Non-Maximum Suppression, що виключає дублікати та знижує кількість помилкових спрацьовувань.

Лістинг 2.8 – Уривок коду Model Builder

```
    def create_model(num_classes, pretrained=True):
        backbone = mobilenet_v2(weights='DEFAULT').features
        backbone.out_channels = 1280

        anchor_generator = AnchorGenerator(
            sizes=((32, 64, 128, 256, 512),),
            aspect_ratios=((0.5, 1.0, 2.0),)
        )

        roi_pooler = torchvision.ops.MultiScaleRoIAlign(
            featmap_names=['0'],
            output_size=7,
            sampling_ratio=2
        )

        model = FasterRCNN(
            backbone,
            num_classes=num_classes,
            rpn_anchor_generator=anchor_generator,
            box_roi_pool=roi_pooler
        )

        if pretrained and os.path.exists(PRETRAINED_MODEL_PATH):
            checkpoint = torch.load(PRETRAINED_MODEL_PATH)

            if 'state_dict' in checkpoint:
                state_dict = checkpoint['state_dict']
                state_dict = {k.replace('module.', '').replace('backbone.', ''):
v
                              for k, v in state_dict.items()}
                model.load_state_dict(state_dict, strict=False)
            else:

                model.load_state_dict(checkpoint)

            print("Loaded pretrained model from", PRETRAINED_MODEL_PATH)

        return model
```

Останній елемент архітектури – «Visualization» – займається візуалізацією отриманих результатів. За допомогою бібліотеки Matplotlib цей модуль відображає результати детекції, накладаючи bounding boxes з зазначенням класових міток і значень confidence безпосередньо на оригінальні зображення. Такий підхід дозволяє не лише отримати наглядну оцінку роботи моделі, але й

зберегти дані й метрики у форматах, придатних для подальшого аналізу з використанням Pandas. Відповідна реалізація наведено в лістингу 2.9.

Лістинг 2.9 – Уривок коду Visualization

```
    def visualize_detections(image, boxes, labels, scores, class_names,
threshold=0.5):
        img_pil = Image.fromarray(image)
        draw = ImageDraw.Draw(img_pil)

        try:
            font = ImageFont.truetype("arial.ttf", 20)
        except:
            font = ImageFont.load_default()

        for box, label, score in zip(boxes, labels, scores):
            if score >= threshold:
                class_name = class_names[label]
                color = COLORS.get(class_name, (255, 255, 255))

                draw.rectangle([(box[0], box[1]), (box[2], box[3])],
                               outline=color, width=3)

                text = f"{class_name}: {score:.2f}"
                text_bbox = draw.textbbox((box[0], box[1] - 5), text, font=font)
                text_width = text_bbox[2] - text_bbox[0]
                text_height = text_bbox[3] - text_bbox[1]

                draw.rectangle([(box[0], box[1] - text_height - 5),
                                (box[0] + text_width, box[1])], fill=color)
                draw.text((box[0], box[1] - text_height - 5), text, fill=(0, 0,
0), font=font)

        return np.array(img_pil)
```

Отже, в цьому розділі було здійснено комплексний аналіз сучасних підходів до виявлення об’єктів, зокрема для завдань автономного транспорту з використанням датасету KITTI. На етапі формування вимог та підготовки вхідних даних було розроблено спеціалізовані методики аугментації, що враховують специфіку дорожніх сцен, включаючи трансформації, що імітують різні умови освітлення та погодні явища. Це дозволило значно розширити різноманітність навчальної вибірки та підвищити стабільність роботи алгоритмів.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

3.1 Метрики та процедури оцінки якості алгоритмів виявлення об’єктів

Середня точність (mean Average Precision, mAP) є ключовою метрикою для оцінки якості детекції об’єктів. Вона обчислюється як середнє значення точності (AP) для всіх класів об’єктів при різних порогах IoU (Intersection over Union). Для задач автопілотування особливо важливим є mAP, який враховує діапазон IoU від 50% до 95% [2]. Дослідження показують, що сучасні моделі YOLO досягають mAP близько 57-60% на датасеті KITTI [5], тоді як Faster R-CNN демонструє показники на рівні 70-75% [6].

Точність (Precision) та повнота (Recall) є фундаментальними метриками для оцінки ефективності алгоритмів. Точність визначається як відношення істинно позитивних виявлень до загальної кількості виявлених об’єктів (P = TP/(TP+FP)), тоді як Recall показує частку виявлених об’єктів від загальної кількості присутніх [6].

Частота кадрів (FPS) є критично важливою метрикою для систем реального часу[6]. Для автопілотування рекомендованим мінімумом є 25-30 кадрів на секунду [5,17].

Інтерсекція по об’єднанню (IoU) є ключовим показником точності локалізації об’єктів. Вона обчислюється як відношення площі перетину передбаченого та реального обмежувальних прямокутників до площі їх об’єднання [4,12]. Для критичних систем, таких як автопілотування, часто використовують поріг IoU на рівні 70-75% [5,17].

Стійкість до умов навколишнього середовища є особливо важливою для автопілотування. Дослідження показують, що сучасні алгоритми демонструють зниження точності на 15-20% у складних умовах, таких як дощ або туман [5,17]. Для традиційних методів, таких як HOG, це зниження може досягати 30-

40% [8,20]. Інтерсекція по об’єднанню (Intersection over Union, IoU) є ключовою метрикою для оцінки точності локалізації об’єктів. Вона обчислюється як відношення площі перетину передбаченого та реального обмежувальних прямокутників до площі їх об’єднання. Для більшості застосувань мінімально прийнятним вважається поріг у 50%, проте для критичних систем, таких як автопілотування, часто використовують більш жорсткі критерії на рівні 70-75%.

3.2 Тестування алгоритмів

Тестування алгоритмів детекції об’єктів проводилося на наборі даних KITTI з використанням метрик, описаних у попередньому розділі: точність (precision), повнота (recall), F1-міра, середнє значення IoU (Intersection over Union), а також середня частота кадрів (FPS) та mAP (mean Average Precision). Нижче наведено результати для кожного з алгоритмів.

```
=== Результати оцінки ===
Середня точність (Precision): 0.1079
Середня повнота (Recall): 0.0873
Середній F1-score: 0.0951
Середній IoU: 0.1013
Середня частота кадрів (FPS): 2.60
mAP: 0.7740
```

Рисунок 3.1 – Оцінка роботи SIFT

Рисунок 3.2 – Результат роботи методу SIFT

Алгоритм SIFT показав низькі значення точності та повноти, що свідчить про його недостатню ефективність для задачі детекції об’єктів у реальному часі. Однак високий mAP (0.7740) вказує на хорошу здатність знаходити об’єкти на зображенні, але з великою кількістю помилкових спрацювань. Низька швидкість (2.60 FPS) обмежує його застосування в системах, що вимагають обробку в реальному часі.

	precision	recall	f1-score	support
Car	0.84	0.86	0.85	1493
Van	0.41	0.45	0.43	151
Truck	0.64	0.66	0.65	56
Pedestrian	0.59	0.66	0.62	236
Person_sitting	0.20	0.12	0.15	17
Cyclist	0.58	0.47	0.52	96
Tram	0.64	0.75	0.69	24
Misc	0.34	0.36	0.35	47
DontCare	0.72	0.66	0.69	646
accuracy			0.74	2766
macro avg	0.55	0.55	0.55	2766
weighted avg	0.74	0.74	0.74	2766

Рисунок 3.3 – Оцінка роботи HOG

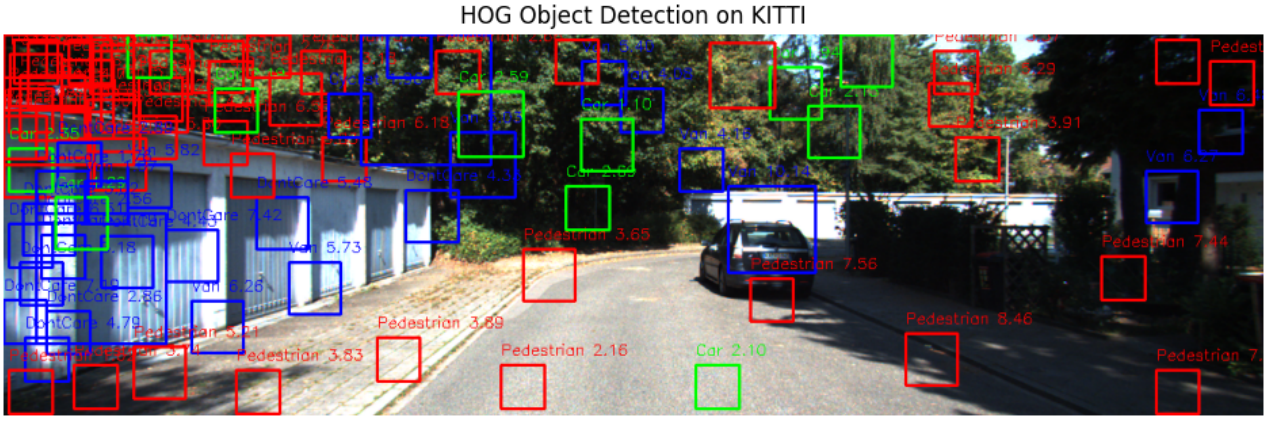

Рисунок 3.3 – Демонстрація роботи HOG

HOG демонструє допустимі результати для класів «Car» (F1 = 0.85) та «Truck» (F1 = 0.65), але менш ефективний для класів із малою кількістю зразків, наприклад «Van» (F1 = 0.43) або «Person_sitting» (F1 = 0.15). Загальна точність

74% свідчить про стабільну роботу, але алгоритм може бути чутливим до варіацій у зовнішності об’єктів.

```
Evaluation Summary:
            precision    recall  f1_score  mean_iou     tp     fp    fn
Car          0.641330  0.805970  0.714286  0.810362  270.0  151.0  65.0
Pedestrian   0.400000  0.739130  0.519084  0.747413   34.0   51.0  12.0
Truck        0.384615  0.714286  0.500000  0.813048   10.0   16.0   4.0
```

Рисунок 3.4 – Оцінка моделі YOLOv11

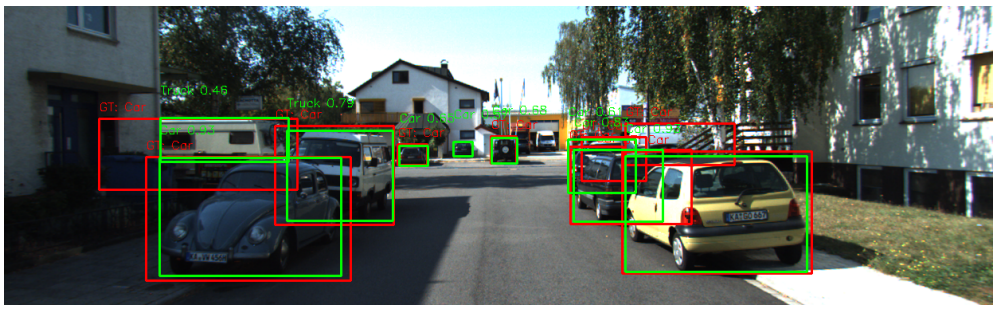

Рисунок 3.5 – Детекція об’єктів YOLO

YOLOv11 досягає високої повноти для класів «Car» (0.806) та «Truck» (0.714), що свідчить про хороше виявлення цих об’єктів. Однак низька точність для «Pedestrian» (0.400) що вказує на значну кількість помилкових спрацювань. Однак «Truck» (0.385) має такий результат через те що YOLO не має такого класу, для моделі «Car» та «Truck» об’єднані під один клас, а саме – «Car». Високий IoU (понад 0.74 для всіх класів) підтверджує точність локалізації об’єктів.

```
=== Результати оцінки ===
Середня точність (Precision): 0.5847
Середня повнота (Recall): 0.6723
Середній F1-score: 0.6053
Середній IoU: 0.7110
Середня частота кадрів (FPS): 1.5
mAP: 0.7119
```

Рисунок 3.7 – Оцінка моделі Faster R-CNN

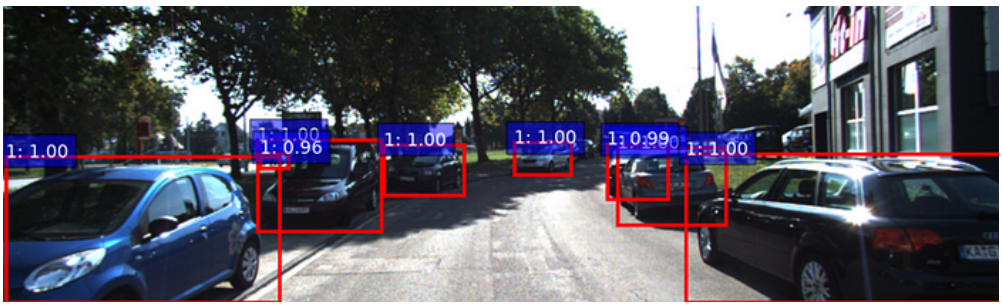

Рисунок 3.6 – Виявлення об’єктів Faster R-CNN

Faster R-CNN демонструє збалансовані результати з точністю 0.5847 та повнотою 0.6723, що робить його надійним для завдань детекції. Високий IoU (0.711) свідчить про точну локалізацію об’єктів, але низька швидкість (1.5 FPS) обмежує його використання у системах реального часу.

3.3 Аналіз результатів експериментів

Проведене дослідження ефективності алгоритмів детекції об’єктів (SIFT, HOG, YOLOv11 та Faster R-CNN) на наборі даних KITTI дозволило отримати комплексну оцінку їх роботи за ключовими показниками. Результати демонструють суттєві відмінності в продуктивності кожного з розглянутих підходів, що обумовлено їх архітектурними особливостями та сферою застосування. У таблиці 3.1 зібрано отримані аналітичні показники.

Таблиця 3.1 – Порівняльна таблиця методів виявлення об’єктів

	SIFT	HOG	YOLOv11	Faster R-CNN
mAP	0.774	0.5511	0.4752	0.5847
Precision	0.1079	0.5511	0.4752	0.5847
Recall	0.0873	0.5344	0.753	0.6723
IoU	0.1013	0.1013	0.7902	0.711
FPS (No CUDO)	2.6	0.8	11.4	1.5

Алгоритм SIFT показав незадовільні результати з точністю 0.1079 та повнотою 0.0873, що свідчить про його низьку ефективність для сучасних задач детекції об’єктів. Хоча показник mAP досягає 0.7740, що вказує на здатність знаходити об’єкти, дуже низька швидкість обробки (2.60 кадрів за секунду) та

мінімальне значення F1-міри (0.0951) роблять його малопридатним для практичного використання.

HOG-детектор продемонстрував хороші результати для поширених класів, зокрема «Car» (точність 0.84, повнота 0.86, F1-середнє 0.85). Проте його ефективність значно знижується для рідкісних категорій, таких як «Van» (F1 = 0.43) чи «Person_sitting» (F1 = 0.15). Загальна точність алгоритму на рівні 74% дозволяє використовувати його в базових системах детекції, проте для складних сценаріїв він виявляється недостатньо надійним.

Модель YOLOv11 показала вражаючу продуктивність, поєднуючи високу точність детекції з відмінною швидкодією (11.5 FPS). Для класу «Car» досягнуто показників повноти 0.806 при точності 0.641, що свідчить про здатність ефективно виявляти ці об’єкти. Високі значення IoU (понад 0.74 для всіх категорій) підтверджують точність локалізації. Особливо варто відзначити, що модель демонструє збалансованість між швидкістю роботи та якістю виявлення.

Faster R-CNN показав стабільні результати з точністю 0.5847 та повнотою 0.6723. Високий IoU (0.7110) та mAP (0.7119) свідчать про його надійність, проте низька швидкість обробки (1.5 FPS) суттєво обмежує сферу його застосування.

Порівняльний аналіз виявив, що YOLOv11 є оптимальним вибором для більшості практичних завдань, поєднуючи високу швидкість (11.5 FPS) з доброю якістю детекції. HOG може бути корисним для швидкісних, але менш вимогливих до точності систем. Faster R-CNN демонструє гарну якість, але через низьку швидкість підходить лише для задач, де час обробки не є критичним. SIFT показав себе неефективним для сучасних завдань комп’ютерного зору.

ВИСНОВКИ

Порівняльний аналіз архітектур YOLO та Faster R-CNN виявив їх суттєві переваги перед традиційними методами (HOG, SIFT) у задачах детекції об’єктів для автономного водіння. Особливо варто відзначити здатність сучасних нейромереж забезпечувати високу точність розпізнавання при збереженні можливості роботи в реальному часі, що є критично важливим для систем безпеки дорожнього руху.

Дослідження реалізації алгоритмів у популярних фреймворках машинного навчання (TensorFlow, PyTorch, OpenCV, Ultralytics) показало, що кожен з них має свої переваги для конкретних сценаріїв використання.

Результати роботи підтверджують, що розвиток технологій комп’ютерного зору продовжує прискорюватися під впливом нових архітектур глибокого навчання та зростаючої потужності обчислювальних систем. Отримані висновки можуть стати основою для подальших досліджень у напрямку оптимізації продуктивності систем детекції об’єктів та їх адаптації до різних умов експлуатації.

Перспективи подальших досліджень полягають у розробці нових гібридних архітектур, що поєднуватимуть переваги трансформерних моделей з ефективністю згорткових нейромереж, а також у вдосконаленні методів оптимізації для роботи на ресурсообмежених пристроях. Це дозволить розширити сферу застосування систем комп’ютерного зору в автономному транспорті та інших критично важливих галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Davies E. R. Computer Vision: Principles, Algorithms, Applications, Learning. Elsevier Science & Technology Books, 2017. 900 с.

2. Lin T.-Y., Maire M. Microsoft COCO: Common Objects in Context. arXiv.org. URL: https://arxiv.org/abs/1405.0312 (дата звернення: 28.02.2025).

3. Shorten C., Khoshgoftaar T. M. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 2019. Т. 6, № 1. URL: https://doi.org/10.1186/s40537-019-0197-0 (дата звернення: 20.03.2025).

4. LeCun Y., Bengio Y., Hinton G. Deep learning. Nature. 2015. Т. 521, № 7553. С. 436–444. URL: https://doi.org/10.1038/nature14539 (дата звернення: 28.02.2025)

5. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv.org. URL: https://arxiv.org/abs/1804.02767 (дата звернення: 28.02.2025).

6. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / S. Ren та ін. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. Т. 39, № 6. С. 1137–1149. URL: https://doi.org/10.1109/tpami.2016.2577031 (дата звернення: 02.02.2025).

7. Bojarski M., Zieba K. End to End Learning for Self-Driving Cars. arXiv.org.URL:https://arxiv.org/abs/1604.07316 (дата звернення: 28.03.2025).

8. Ouaknine A. Review of deep learning algorithms for object detection. URL: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852 (дата звернення: 02.02.2025).

9. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick et al. arXiv.org. URL: https://arxiv.org/abs/1311.2524 (дата звернення: 10.03.2025).

10. Lowe D. G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. 2004. Т. 60. № 2. С. 91-110.

URL: https://doi.org/10.1023/b:visi.0000029664.99615.94 (дата звернення: 10.03.2025).

11. Bandara R. CodeProject. CodeProject - For those who code. URL: https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O (дата звернення: 10.03.2025).

12. Speeded-Up Robust Features (SURF) H. Bay et al. Computer Vision and Image Understanding. 2008. Т. 110, № 3. С. 346–359. URL: https://doi.org/10.1016/j.cviu.2007.09.014 (дата звернення: 10.03.2025).

13. Zhang W., Tong R., Dong J. Boosted cascade of scattered rectangle features for object detection. Science in China Series F: Information Sciences. 2009. Т. 52. № 2. С. 236–243. URL: https://doi.org/10.1007/s11432-009-0034-8 (дата звернення: 20.03.2025).

14. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick et al. arXiv.org. URL: https://arxiv.org/abs/1311.2524 (дата звернення: 20.03.2025).

15. Mittal K. A gentle introduction into the histogram of oriented gradients. URL: https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa (дата звернення: 20.03.2025).

16. Liu W., Reed S. SSD: Single Shot MultiBox Detector. arXiv.org. URL: https://arxiv.org/abs/1512.02325 (дата звернення: 30.03.2025).

17. End-to-End object detection with transformers / N. Carion et al. arXiv.org. URL: https://arxiv.org/abs/2005.12872 (дата звернення: 20.03.2025).

18. OpenCV. OpenCV: introduction to SURF (speeded-up robust features). OpenCV documentation index. URL: https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html (дата звернення: 20.03.2025).

19. YOLO Object Detection Explained: A Beginner’s Guide | DataCamp. datacamp. URL: https://www.datacamp.com/blog/yolo-object-detection-explained (дата звернення: 25.03.2025).

20. GitHub – ultralytics/ultralytics: Ultralytics YOLO11. GitHub. URL: https://github.com/ultralytics/ultralytics (дата звернення: 25.03.2025).

21. PyTorch documentation – PyTorch 2.7 documentation. Redirecting... URL: https://docs.pytorch.org/docs/stable/index.html (дата звернення: 25.03.2025).

22. TensorFlow 2 Object Detection API – documentation. URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/ (дата звернення: 30.03.2025).

23. Spaces - Hugging Face. Hugging Face – The AI community building the future. URL: https://huggingface.co/spaces (дата звернення: 25.03.2025).

24. Towards a better understanding of annotation tools for medical imaging: a survey / M. Aljabri et al. *Multimedia Tools and Applications*. 2022. URL: https://doi.org/10.1007/s11042-022-12100-1 (дата звернення: 25.03.2025).

ДОДАТКИ

Додаток А – Уривок коду для зіставлення виокремлених ознак

```
def match_features(self, img, templates):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kp_img, des_img = self.sift.detectAndCompute(gray, None)

    if des_img is None:
        return []

    detections = []
    for template in templates:
        matches = self.flann.knnMatch(
            template['descriptors'],
            des_img,
            k=2
        )

        good_matches = []
        for m, n in matches:
            if m.distance < 0.7 * n.distance:
                good_matches.append(m)

        if len(good_matches) > self.min_match_count:
            src_pts = np.float32(
                [template['keypoints'][m.queryIdx].pt for m in good_matches]
            ).reshape(-1, 1, 2)
            dst_pts = np.float32(
                [kp_img[m.trainIdx].pt for m in good_matches]
            ).reshape(-1, 1, 2)

            M, mask = cv2.findHomography(
                src_pts, dst_pts, cv2.RANSAC, 5.0
            )

            if M is not None:
                h, w = img.shape[:2]
                pts = np.float32([
                    [0, 0], [0, h-1], [w-1, h-1], [w-1, 0]
                ]).reshape(-1, 1, 2)
                dst = cv2.perspectiveTransform(pts, M)

                detections.append({
                    'class': template['class'],
                    'matches': len(good_matches),
                    'bbox': cv2.boundingRect(dst)
                })

    return detections
```

Додаток Б – Уривок коду для виявлення характерних ознак методом SIFT для зображень

```
    def evaluate(self, sample_size=50, output_dir='sift_results'):
        Path(output_dir).mkdir(exist_ok=True)

        img_files, label_dir = self.load_kitti_data()
        img_files = img_files[:sample_size]

        results = []
        for img_path in tqdm(img_files, desc="Тестування SIFT"):
            label_path = label_dir / f"{img_path.stem}.txt"
            img, templates = self.extract_template_features(img_path,
label_path)

            if img is None or not templates:
                continue

            detections = self.match_features(img, templates)

            result_img = img.copy()
            for det in detections:
                x, y, w, h = det['bbox']
                cv2.rectangle(result_img, (x, y), (x+w, y+h), (0, 255, 0), 2)
                label = f"{det['class']} ({det['matches']})"
                cv2.putText(
                    result_img, label, (x, y-10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1
                )

            output_path = Path(output_dir) / f"result_{img_path.name}"
            cv2.imwrite(str(output_path), result_img)

            for det in detections:
                results.append({
                    'image': img_path.name,
                    'class': det['class'],
                    'matches': det['matches']
                })

        if results:
            df = pd.DataFrame(results)
            print("\nРезультати тестування SIFT:")
            print("Розподіл виявлених об'єктів:")
            print(df['class'].value_counts())

            print("\nСередня кількість відповідностей:")
            print(df.groupby('class')['matches'].mean())

            df.to_csv(Path(output_dir) / 'sift_metrics.csv', index=False)
            print(f"\nРезультати збережено у: {output_dir}")
        else:
            print("Не знайдено жодного відповідника, що відповідає критеріям")

        return results
```

Додаток В – Уривок коду для виявлення об’єктів моделі Faster R-CNN

```
def detect_and_visualize(model, image_dir, dataset, confidence_threshold=0.5,
max_images=5):
        model.eval()
        test_images = [f for f in os.listdir(image_dir) if
f.endswith('.png')][:max_images]

        for img_name in test_images:
            img_path = os.path.join(image_dir, img_name)
            image = cv2.imread(img_path)
            if image is None:
                print(f"Could not read image {img_path}")
                continue

            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image_tensor = F.to_tensor(image).unsqueeze(0).to(DEVICE)

            with torch.no_grad():
                prediction = model(image_tensor)

            boxes = prediction[0]['boxes'].cpu().numpy()
            labels = prediction[0]['labels'].cpu().numpy()
            scores = prediction[0]['scores'].cpu().numpy()

            keep = scores >= confidence_threshold
            boxes = boxes[keep]
            labels = labels[keep]
            scores = scores[keep]

            vis_image = visualize_detections(
                image, boxes, labels, scores,
                dataset.classes, confidence_threshold
            )

            plt.figure(figsize=(16, 10))
            plt.imshow(vis_image)
            plt.axis('off')
            plt.title(f'Detection results: {img_name}')
            plt.tight_layout()
    plt.show()
```

Додаток Г – Репозиторій на GitHub

https://github.com/lame-over/Algorithms-and-neural-networks