

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій

(повне найменування факультету)

Комп'ютерні системи та мережі

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалаврський

(ступінь вищої освіти)

на тему: РОЗРОБКА ПІДСИСТЕМИ АВТОМАТИЗОВАНОГО СКАНУВАННЯ
ГАБАРИТІВ ВАНТАЖУ НА СКЛАДІ

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-511

Спеціальності

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерна інженерія

(назва освітньої програми (спеціалізації))

БІРЮКОВ Д.П.

(ПРИЗВИЩЕ та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

Рецензент МАЛИЙ О.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри
КУДЕРМЕТОВ Р.К.

«14» квітня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БІРЮКОВА Дениса Павловича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка підсистеми автоматизованого сканування
габаритів вантажу на складі

керівник проєкту (роботи) к.т.н., доцент, ЗЕЛЕНЬОВА І.Я.

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 08 » квітня 2025 року № 151

2. Строк подання студентом проєкту (роботи) 01.06.2025 р.

3. Вихідні дані до проєкту (роботи) техічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Опис предметної області;

2) Розробка вимог до програмного забезпечення;

3) Моделювання програмного забезпечення;

4) Проєктування архітектури програмного забезпечення;

5) Проєктування інтерфейсу користувач.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту(роботи).

Розділ	ПРИЗВИЩЕ, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ЗЕЛЕНЬОВА І. Я.		
нормоконтроль	ПОЛЬСЬКА О. В.		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області та теоретичні основи комп'ютерного зору і програмної інженерії	15.04.2025 р.	
2	Розробка підсистеми автоматизованого сканування габаритів вантажу на складі	15.04.2025 р.	
3	Тестування та практичні рекомендації з експлуатації Підсистеми	08.05.2025 р.	
4	Оформлення отриманих результатів у ПЗ	20.05.2025 р.	
5	Оформлення презентації	01.06.2025 р.	
6	Оформлення допоміжного матеріалу	09.06.2025 р.	
7	Проходження нормоконтролю	до 01.06.2025	
8	Перевірка на наявність академічного плагіату	до 03.06.2025	
9	Проходження рецензування	до 10.06.2025	

Студент (ка)

_____ **Денис БІРЮКОВ**
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ **Ірина ЗЕЛЕНЬОВА**
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
74с., 4 табл., 2 рис. , 29 джерел.

АВТОМАТИЗАЦІЯ, ІНТЕРФЕЙС API, КОМП'ЮТЕРНЕ БАЧЕННЯ,
МАШИННЕ НАВЧАННЯ, МОДЕЛЬ YOLOv8, ОБРОБКА ЗОБРАЖЕНЬ,
ПІДСИСТЕМА РОСПІЗНАВАННЯ.

Предмет дослідження – методи та моделі автоматичного виявлення габаритів вантажів з використанням технологій комп'ютерного зору та глибокого навчання.

Мета роботи – розробка автоматизованої підсистеми сканування габаритів вантажу на складських конвеєрних лініях із використанням моделі YOLOv8, з інтеграцією в цифрову інфраструктуру за допомогою FastAPI та Docker.

У роботі проведено аналіз методів комп'ютерного зору та алгоритмів глибокого навчання, реалізовано серверну частину підсистеми з використанням FastAPI, здійснено контейнеризацію застосунку через Docker, реалізовано API для взаємодії з користувачем. Для тренування моделі застосовано фреймворк YOLOv8 та інструмент Roboflow. Проведено тестування точності та стабільності системи в умовах, наближених до реального складу.

Практична цінність – створено готову до впровадження систему автоматичного розпізнавання габаритів вантажу, яка може використовуватись у складі сучасних логістичних платформ для підвищення точності та ефективності обліку товарів.

ЗМІСТ

Скорочення та умовні позначки.....	6
Вступ	7
1 Огляд систем комп'ютерного зору	9
1.1 Огляд комп'ютерного зору та його застосування в логістиці	9
1.2 Методи машинного навчання для обробки зображень	14
1.3 Огляд інструментів для розробки систем розпізнавання об'єктів.....	20
1.4 Архітектурні та методологічні засади побудови систем комп'ютерного зору..	26
2 Проєктування підсистеми розпізнавання габаритів вантажу	29
2.1 Постановка задачі та вимоги до підсистеми.....	29
2.3 Інтеграція моделі YOLOv8 для виявлення габаритів вантажу	41
2.4 Практичні аспекти побудови програмної підсистеми виявлення об'єктів ...	46
3 Розгортання та тестування системи.....	49
3.1 Контейнеризація застосунку з використанням	49
3.2 Розгортання в середовищі кінцевого користувача	52
3.3 Тестування, перевірка точності та стабільності системи	57
3.4 Оцінка ефективності розробленої підсистеми в умовах тестування	61
Висновки.....	64
Перелік джерел посилання.....	66
Додаток А Лістинги програм.....	69

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API	– Application Programming Interface, Програмний інтерфейс прикладного програмування
CNN	– Convolutional Neural Network, Згортова нейронна мережа
CSV	– Comma-Separated Values, Формат табличних даних
Docker	– Система віртуалізації для контейнеризації застосунків
GUI	– Graphical User Interface, Графічний інтерфейс користувача
HTTP	– HyperText Transfer Protocol, Протокол передачі гіпертексту
JSON	– JavaScript Object Notation, Формат обміну структурованими даними
OCR	– Optical Character Recognition, Оптичне розпізнавання символів
REST	– Representational State Transfer, Архітектурний стиль вебсервісів
Roboflow	– Онлайн-платформа для розмітки та підготовки датасетів
YOLO	– You Only Look Once, Алгоритм виявлення об'єктів у реальному часі
PNG	– Portable Network Graphics, Растровий графічний формат
WMS	– Warehouse Management System, Система управління складом
SSD	– Single Shot Multibox Detector, Одноетапний детектор об'єктів на зображеннях, що виконує локалізацію та класифікацію в один прохід нейронною мережею

ВСТУП

У сучасних логістичних системах ефективність обліку, сортування та обробки вантажів напряду залежить від швидкості і точності збору інформації про їхні фізичні характеристики. Одним із критично важливих параметрів у цьому контексті є габарити вантажу — розміри кожної коробки, контейнера або іншого транспортованого об'єкта, що переміщується складуванням або транспортуванням. У традиційній практиці вимірювання габаритів виконуються вручну, що є вкрай неефективним при масштабних обсягах обробки і веде до значного навантаження на персонал, збільшення кількості помилок та зниження загальної продуктивності. Автоматизація цього процесу за допомогою засобів комп'ютерного зору дозволяє істотно оптимізувати логістичні операції, усунути людський фактор і забезпечити інтеграцію з цифровими платформами управління складом (WMS).

Прогрес у сфері глибокого навчання, зокрема в розробці архітектур для виявлення об'єктів, таких як YOLOv8, відкрив можливість створення точних і швидкодіючих систем автоматичного розпізнавання зображень. У рамках цієї дипломної роботи було реалізовано підсистему, яка здатна у напіваавтоматичному або повністю автоматизованому режимі отримувати зображення конвеєрної стрічки, визначати контури вантажів і повертати результат у вигляді візуалізації або даних для подальшої обробки. Реалізація базується на архітектурі FastAPI, що забезпечує зручний інтерфейс для взаємодії, а контейнеризація через Docker дозволяє легко розгорнути систему на будь-якому комп'ютері без складного налаштування.

Мета дослідження полягає у створенні програмної підсистеми для автоматичного виявлення та сканування габаритів вантажів на зображеннях із складських конвеєрних ліній із використанням глибокого навчання та методів комп'ютерного зору.

Об'єктом дослідження є програмні засоби автоматизованої обробки зображень у логістичних процесах.

Предметом дослідження є методи і алгоритми виявлення об'єктів на зображеннях з використанням моделей глибокого навчання, а також їх інтеграція в інженерну інфраструктуру вебсервісів.

Завдання дослідження:

- проаналізувати сучасні підходи до автоматизації контролю габаритів вантажів у логістиці;
- дослідити архітектури моделей комп'ютерного зору, зокрема YOLOv8;
- сформувати та підготувати датасет для навчання моделі;
- провести навчання моделі з використанням трансферного навчання;
- розробити серверну частину підсистеми на основі FastAPI;
- реалізувати вебінтерфейс та REST API для взаємодії з користувачем;
- забезпечити розгортання системи в ізолюваному середовищі за допомогою Docker;
- провести тестування точності, швидкодії та стабільності роботи системи.

Практична цінність дипломної роботи полягає у поєднанні сучасної моделі глибокого навчання YOLOv8 з легко масштабованим вебфреймворком FastAPI у рамках автоматизованої підсистеми, придатної до використання на реальному складі, що вирізняється простотою впровадження, високою точністю розпізнавання та підтримкою інтеграції через API без складної конфігурації.

Актуальність дослідження обумовлена стрімким зростанням вимог до цифрової трансформації логістичних процесів, де ручні операції потребують заміни на автоматизовані рішення. Створення гнучких, точних і легко розгорнутих систем комп'ютерного зору є однією з ключових потреб сучасних логістичних підприємств, особливо в умовах високих обсягів обробки вантажів, необхідності швидкої адаптації до змін та інтеграції в загальноскладські платформи обліку й керування.

1 ОГЛЯД СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Огляд комп'ютерного зору та його застосування в логістиці

Комп'ютерний зір є однією з найдинамічніших і найперспективніших галузей сучасної науки й технологій, яка поєднує досягнення штучного інтелекту, обробки зображень, аналітики даних та апаратного забезпечення. Його основна мета полягає у здатності надавати машинам можливість "бачити" навколишній світ, так, як це робить людина, — сприймати візуальну інформацію, аналізувати її, класифікувати об'єкти, виявляти закономірності, просторові відношення, зміни та інші візуальні ознаки. Ця технологія дозволяє перетворити звичайне зображення або відеопотік із камери у структуровані дані, що можуть бути використані для прийняття рішень, керування процесами або автоматизації [1]. У сучасному розумінні комп'ютерний зір — це міждисциплінарна область, що охоплює методи машинного навчання, глибокого навчання (deep learning), математики, теорії ймовірностей, обчислювальної геометрії та нейромережових архітектур.

Історично комп'ютерний зір розвивався як галузь обробки зображень — починаючи з задач покращення якості зображення, виділення контурів, сегментації об'єктів. Однак лише із впровадженням методів штучного інтелекту з'явилася можливість перейти від пасивного аналізу до інтерпретації зображення. Глибокі згорткові нейронні мережі (CNN) стали проривом у цій галузі, що дозволило навчати моделі виявляти, класифікувати і навіть реконструювати об'єкти на рівні, близькому до людського сприйняття. На відміну від класичних алгоритмів, які базувалися на ручному виділенні ознак (features), сучасні підходи дозволяють системам навчатися безпосередньо на даних, формуючи власні внутрішні уявлення про світ через багаторівневу обробку [2].

Поняття комп'ютерного зору охоплює широкий спектр задач: розпізнавання образів, детектування об'єктів, семантична та інстанс-

сегментація, відстеження руху, реконструкція тривимірних сцен, аналіз глибини, аналіз сцен і контексту. У прикладному вимірі ці задачі реалізуються у безпілотному водінні, охоронних системах, робототехніці, медицині (аналіз МРТ, рентгенівських знімків), промисловості (контроль якості продукції), сільському господарстві (виявлення хвороб рослин), торгівлі (підрахунок людей, визначення поведінки клієнтів) тощо [3]. Основною перевагою комп'ютерного зору є можливість обробляти великі обсяги візуальної інформації у реальному часі, при цьому усуваючи вплив людського фактора, який часто є причиною похибок, затримок та обмеження продуктивності.

У контексті логістики та складських систем комп'ютерний зір відкриває абсолютно нові горизонти для автоматизації. Завдяки камерам, розміщеним над конвеєрними стрічками, у зонах приймання чи відправлення вантажів, система може в режимі реального часу ідентифікувати об'єкти, визначати їхні розміри, положення, орієнтацію, стан поверхні та інші важливі характеристики. Це, своєю чергою, дозволяє автоматизувати процеси сканування, сортування, маркування та відстеження вантажів. На відміну від ручного контролю, який потребує участі персоналу, комп'ютерний зір дозволяє виконувати аналогічні функції цілодобово, з однаковою точністю та швидкістю, без втоми або суб'єктивних помилок.

Поняття комп'ютерного зору в логістиці часто поєднується з термінами, як-от «інтелектуальне відеоспостереження», «візуальна інспекція», «індустріальний візуальний контроль», «оптичне розпізнавання» (Optical Recognition) тощо. Програмне забезпечення на основі комп'ютерного зору дозволяє виявляти, наприклад, чи пошкоджена упаковка, чи правильно розміщений вантаж на палеті, чи не перевищено дозволені габарити, чи не було заміни товару під час транспортування. Крім того, розпізнавання штрихкодів, QR-кодів, а також тексту (OCR) також є частиною ширшого спектра завдань комп'ютерного зору у цій сфері.

Технічно системи комп'ютерного зору складаються з кількох основних компонентів: апаратної частини (камери, освітлення, комп'ютери), програмної

частини (моделі обробки зображень), засобів комунікації (API, інтеграція з WMS), та інтерфейсів взаємодії (GUI, вебінтерфейси, мобільні додатки). Для досягнення максимальної ефективності ці компоненти мають бути синхронізовані. Наприклад, якщо камера не має достатньої роздільної здатності або правильного кута огляду, модель не зможе коректно обробити вхідні дані. Тому якість зображення, а також його попередня обробка (нормалізація, зменшення шуму, вирівнювання контрасту) має вирішальне значення для точності роботи системи.

Зважаючи на розвиток технологій, останнім часом усе більшу популярність набувають так звані edge-системи, коли обробка зображень відбувається безпосередньо на пристрої збору даних — наприклад, на камері з вбудованим процесором або міні-комп'ютері типу Raspberry Pi. Це дозволяє мінімізувати затримки, зменшити навантаження на центральний сервер і забезпечити роботу навіть при обмеженому інтернет-з'єднанні. Проте у випадку складських систем, де обробка зображень може виконуватись централізовано, з використанням серверів або хмарної інфраструктури, часто надається перевага розгортанню нейромережових моделей у середовищах типу Docker, що спрощує масштабування, оновлення та підтримку.

Поява відкритих фреймворків, таких як OpenCV, PyTorch, TensorFlow та Ultralytics, значно спростила створення систем комп'ютерного зору. Тепер навіть відносно невелика команда розробників може створити власне рішення, яке використовуватиме складні алгоритми обробки зображень, без необхідності розробляти їх з нуля. У межах дипломної роботи також було використано один із таких фреймворків — YOLOv8 — який об'єднує в собі новітні напрацювання в галузі розпізнавання об'єктів, оптимізований під сучасні обчислювальні системи, і здатний працювати як у режимі реального часу, так і в пакетній обробці [4].

Сучасне трактування комп'ютерного зору, з урахуванням практичного досвіду його впровадження у різних галузях, дозволяє розглядати його не лише як допоміжний інструмент, а як повноцінну складову частину цифрової

трансформації. Його застосування у логістиці — це не просто технологічна інновація, а відповідь на нагальні виклики, пов'язані з потребою у підвищенні ефективності, прозорості та масштабованості складських операцій.

Упродовж останніх років технології комп'ютерного зору почали активно впроваджуватись у низку прикладних сфер, і з-поміж них особливо виділяються складські комплекси, виробничі підприємства та лінії технічного контролю якості продукції. Ці сфери традиційно асоціюються з високим ступенем повторюваності операцій, великою кількістю об'єктів для обробки та критичними вимогами до точності, швидкості й надійності виконання рутинних завдань. Використання систем візуального аналізу у цих контекстах дозволяє не лише зменшити навантаження на персонал, а й отримати точні, уніфіковані й документовані результати, які легко інтегруються у цифрові потоки даних підприємства.

На складах комп'ютерний зір використовується для вирішення широкого спектра завдань: ідентифікація упаковок, підрахунок одиниць вантажу, перевірка правильності маркування, виявлення пошкоджених коробок, вимірювання габаритів і розпізнавання QR- та штрихкодів. Крім того, системи автоматичного візуального контролю дозволяють оцінювати рівномірність завантаження палет, перевіряти заповненість комірок зберігання, а також контролювати послідовність розміщення товарів на стрічках автоматизованого транспорту [5]. Камери можуть бути розміщені на різних ділянках складу — як на вході, де відбувається приймання товару, так і на виході, де здійснюється відвантаження, або на внутрішніх маршрутах переміщення. Завдяки цьому створюється наскрізний візуальний ланцюг, який дозволяє постійно відстежувати місцезнаходження вантажів, зменшити ризики втрат та оптимізувати логістичні операції.

У виробничій сфері технології комп'ютерного зору є основою для побудови інтелектуальних систем контролю. Вони дозволяють виявляти дефекти на різних етапах обробки — починаючи від контролю якості вхідної сировини до перевірки готової продукції перед пакуванням. Камери високої роздільної

здатності фіксують зображення об'єктів, які потім передаються на обробку до моделі машинного навчання. Модель аналізує форму, текстуру, колір, геометричні параметри та порівнює їх із еталонними значеннями. У разі виявлення відхилень система може автоматично повідомити оператора, відхилити об'єкт з потоку або зареєструвати інцидент для подальшої аналітики. Такий підхід забезпечує раннє виявлення проблем, покращення якості продукції, зниження витрат на повернення бракованих товарів і підвищення загального рівня надійності виробничої системи.

Ще одним важливим напрямом застосування комп'ютерного зору є контроль якості, що охоплює не тільки виробництво, а й етапи зберігання, транспортування і навіть обслуговування товарів. У цьому контексті комп'ютерний зір може забезпечувати контроль за відповідністю товару вимогам стандартів, перевірку правильності укладки, наявності обгортки, пломб, стікерів, маркування. Наприклад, під час автоматичної обробки посилок у логістичному центрі камера може фіксувати наявність усіх необхідних позначок, визначати пошкодження обгортки, порівнювати зовнішній вигляд упаковки з очікуваним. Також технологія дозволяє виявляти порушення температурного режиму за допомогою термокамер, слідкувати за правильністю зберігання небезпечних або крихких вантажів за допомогою систем спостереження, що навчені на відповідних візуальних шаблонах [6]. Усе це підвищує загальний рівень надійності системи логістики й зменшує імовірність інцидентів на лінії.

Водночас основним операційним ядром усіх вищенаведених сценаріїв є саме завдання виявлення об'єктів — тобто здатність системи не лише аналізувати зображення, а й точно локалізувати на ньому об'єкти інтересу. Це завдання є відправною точкою для більшості подальших обчислень, включаючи вимірювання габаритів, оцінку дефектів, ідентифікацію типу товару, підрахунок одиниць. Виявлення передбачає створення прямокутних областей (bounding boxes) навколо цільових об'єктів, що дозволяє фокусувати увагу системи на потрібних зонах зображення і забезпечує контекст для інтерпретації. Наприклад, для системи, яка виконує сканування вантажів на стрічці, критично важливо не

лише розпізнати, що об'єкт є коробкою, а й точно вказати його межі, щоб визначити розміри, обсяг, орієнтацію або наявність деформації.

Процес виявлення об'єктів вимагає високої обчислювальної ефективності, оскільки він є основним “бутилковим горлечком” у пайплайні комп'ютерного зору. Чим швидше і точніше система визначить, де знаходиться об'єкт, тим менше затримка в усьому процесі. Саме тому моделі типу YOLO (You Only Look Once) на сьогодні є одними з найбільш ефективних, оскільки вони виконують виявлення об'єктів у реальному часі без потреби у розбитті зображення на окремі області. Це досягається за рахунок того, що модель одночасно виконує як регресію координат bounding boxes, так і класифікацію об'єкта, що значно пришвидшує обробку [7]. У промислових умовах, де об'єкти рухаються на стрічці зі швидкістю в кілька метрів за секунду, такий підхід є життєво необхідним для збереження продуктивності.

У випадку систем, орієнтованих на логістику, виявлення об'єктів дозволяє створювати повністю автоматизовані рішення, які можуть працювати без участі людини — наприклад, автономне приймання товару, автоматичне формування маршрутних листів, інтелектуальне сортування на конвеєрі. Усі ці процеси потребують лише одного — точного, стабільного і швидкого визначення об'єкта на вхідному зображенні. Після цього будь-яка логіка може бути накладена програмно: об'єкт можна виміряти, порівняти з еталоном, облікувати, перевірити на відповідність. Саме тому виявлення об'єктів виступає базовим етапом і “точкою входу” в автоматизований візуальний процес [8].

1.2 Методи машинного навчання для обробки зображень

Методи машинного навчання, а особливо глибокого навчання, відіграють фундаментальну роль у сучасних підходах до обробки зображень у комп'ютерному зорі. Якщо на ранніх етапах розвитку комп'ютерного зору

основна увага приділялась класичним алгоритмам обробки — таким як гістограмна рівномірність, детектори країв (Canny, Sobel), класифікація на основі ручних ознак (HOG, SIFT) — то з появою методів глибокого навчання ситуація радикально змінилася. Глибоке навчання стало тією технологічною основою, яка дозволила системам автоматично навчатися ознакам, взаємозв'язкам і шаблонам у зображеннях без втручання людини [9]. Це, своєю чергою, забезпечило суттєве зростання точності в задачах класифікації, виявлення об'єктів, сегментації та інших високорівневих сценаріях візуального аналізу.

Під поняттям глибокого навчання (deep learning) розуміють особливий клас методів машинного навчання, що базується на використанні глибоких нейронних мереж (deep neural networks) — моделей, які мають велику кількість шарів (іноді — десятки або сотні), кожен з яких виконує певне перетворення вхідних даних. Принцип дії глибоких мереж полягає в тому, що з кожним шаром модель навчається дедалі більш абстрактним ознакам. Наприклад, у випадку обробки зображень перші шари можуть реагувати на базові патерни — границі, кольори, текстури, наступні — на більш складні фрагменти, як-от контури об'єктів, а найвищі рівні вже формують уявлення про конкретні форми, типи об'єктів, їх взаємне розташування [10].

У галузі обробки зображень найширше поширення отримали згорткові нейронні мережі (Convolutional Neural Networks, CNN). Їхня особливість полягає в тому, що вони використовують операції згортки (convolution), які дозволяють обробляти локальні області зображення, виявляючи характерні особливості без потреби аналізувати повне зображення кожного разу. Згортка реалізується як проходження ядра фільтра по зображенню, в результаті чого формується карта ознак, яка відображає наявність певного шаблону або елементу. Основна перевага CNN над традиційними повнозв'язними мережами полягає в тому, що вони враховують просторову структуру зображення та значно зменшують кількість параметрів, що потребують навчання.

Сучасні CNN складаються з низки згорткових шарів, шарів активації (зазвичай ReLU), шарів підвибірки (наприклад, max pooling), нормалізації та

одного або декількох повнозв'язних шарів у кінці. У класичному варіанті мережа спочатку «стискає» просторову інформацію про зображення, витягає найбільш значущі ознаки, а потім виконує класифікацію або передбачення координат об'єкта [11]. У задачах виявлення об'єктів, до яких належить і завдання розпізнавання габаритів вантажу, CNN зазвичай поєднуються з додатковими модулями регресії координат (наприклад, bounding boxes), що дозволяє не лише виявити, який об'єкт зображено, а й де саме він знаходиться на зображенні [12].

Навчання згорткової мережі здійснюється в процесі ітеративного оновлення вагів з використанням алгоритму зворотного поширення помилки (backpropagation). На кожному кроці навчання мережа порівнює свій прогноз із реальним значенням (ground truth), обчислює втрати (loss), а потім через градієнтний спуск змінює параметри, щоб мінімізувати ці втрати. Цей процес повторюється багаторазово для кожного зображення у вибірці, що зрештою дозволяє мережі сформувати узагальнене уявлення про структуру даних.

Ще однією важливою перевагою глибоких мереж є автоматичне вилучення ознак (feature extraction). Якщо в традиційних алгоритмах візуального аналізу необхідно було вручну конструювати ознаки (на основі градієнтів, текстур, інтервалів кольору тощо), то в CNN це відбувається автоматично: мережа сама знаходить найінформативніші шаблони, необхідні для розпізнавання [13]. Такий підхід не лише економить час, а й забезпечує кращу адаптацію до складних, реалістичних умов — наприклад, коли об'єкти можуть бути частково перекриті, деформовані, мати нерівне освітлення або зміщення по осі.

Одним з найбільших викликів при побудові глибоких мереж для обробки зображень є необхідність великої кількості даних. Щоб модель навчилася ефективно узагальнювати і не переобучалась на обмеженому наборі прикладів, вона повинна пройти навчання на тисячах, а іноді і на мільйонах зображень. Це створює потребу у високоякісних датасетах з точними розмітками, а також у потужних обчислювальних ресурсах, здатних обробити такий обсяг інформації. У практиці використовується кілька підходів до вирішення цієї проблеми, і

одним із найефективніших є трансферне навчання (transfer learning), яке буде розглянуто в наступному пункті.

Глибоке навчання також дозволяє будувати не лише універсальні моделі, а й спеціалізовані — ті, що адаптовані під конкретну задачу. Наприклад, у контексті автоматичного сканування вантажів на складі модель може бути натренована виключно на зображеннях коробок, палет або контейнерів, отриманих з камер над конвеєром. Це дозволяє значно підвищити точність за рахунок вузької спеціалізації. Крім того, завдяки структурі згорткових мереж можна досягти високої продуктивності навіть на пристроях з обмеженими ресурсами, зокрема за рахунок оптимізації моделей або використання полегшених архітектур типу YOLOv8n, MobileNet, або EfficientNet [14].

Таким чином, глибоке навчання є не просто підходом до аналізу зображень, а повноцінною парадигмою обробки візуальних даних, яка дає змогу розв'язувати задачі, що раніше вважалися надскладними або взагалі нерозв'язуваними в реальному часі. Його застосування у сфері комп'ютерного зору заклало основу для побудови адаптивних, масштабованих і ефективних систем розпізнавання, які знаходять широке застосування у промисловості, медицині, безпеці, логістиці та інших галузях [15].

Однією з найбільш визначальних технологій у сфері глибокого навчання, яка забезпечила прорив у комп'ютерному зорі, є згорткові нейронні мережі (Convolutional Neural Networks, CNN). Вони стали стандартом де-факто у задачах розпізнавання образів, виявлення об'єктів, класифікації зображень, сегментації та багатьох інших напрямках. Архітектура CNN побудована на базі декількох ключових ідей, серед яких — використання просторової згортки для вилучення локальних ознак, зменшення розмірності для підвищення обчислювальної ефективності, ієрархічне представлення ознак, а також навчання всієї структури як єдиного цілого за допомогою зворотного поширення помилки.

Основним будівельним блоком згорткової нейронної мережі є згортковий шар (convolutional layer), який застосовує фільтри (ядра згортки) до вхідного зображення з метою виявлення характерних візуальних патернів. Ці фільтри

автоматично навчаються у процесі тренування і можуть розпізнавати такі особливості, як горизонтальні або вертикальні краї, вигини, текстурні переходи тощо. Зазвичай кілька згорткових шарів поєднуються між собою, і кожен наступний шар оперує дедалі більш абстрактними ознаками: від простих границь — до складних форм і контурів об'єктів [16].

Після згорткових шарів зазвичай застосовується підвибірка (pooling), зокрема max pooling, яка зменшує просторові розміри тензорів ознак, зберігаючи при цьому найбільш значущі елементи. Це допомагає зменшити кількість параметрів, уникнути перенавчання та підвищити інваріантність до незначних змін у вході — наприклад, зсувів чи масштабування об'єкта. Завершальні шари згорткової мережі зазвичай є повнозв'язними (fully connected), де всі ознаки агрегуються в один вектор, який передається до класифікатора.

Різні архітектури CNN відрізняються між собою глибиною (кількістю шарів), структурою зв'язків, способом нормалізації, кількістю параметрів та механізмами оптимізації. Однією з найвідоміших і базових архітектур стала LeNet-5, розроблена ще у 1998 році для розпізнавання рукописних цифр. Наступними етапами розвитку стали AlexNet, що виграв змагання ImageNet у 2012 році та продемонстрував перевагу глибоких CNN над класичними методами. Після цього з'явилися VGGNet з глибшими шарами і фіксованими фільтрами 3×3 , GoogLeNet (Inception) з багатогілковими структурами, а згодом — ResNet з революційною концепцією резидуальних зв'язків (skip connections), що дозволили уникнути проблеми затухаючих градієнтів та навчати надглибокі мережі [17].

Останні роки відзначилися появою більш оптимізованих архітектур, які враховують баланс між точністю та обчислювальною складністю. Серед них — EfficientNet, яка використовує автоматичний пошук архітектур та збалансоване масштабування мереж, MobileNet та ShuffleNet, призначені для роботи на мобільних пристроях, де важлива мінімальна затрата ресурсів. Водночас, для задач детекції об'єктів були створені спеціалізовані архітектури, такі як YOLO

(You Only Look Once), SSD (Single Shot Detector), Faster R-CNN, які поєднують CNN з регресійними модулями для передбачення координат об'єктів [18].

Зокрема, YOLO став однією з найпопулярніших моделей для реального часу завдяки своїй унікальній структурі. На відміну від традиційних підходів, які розділяли задачу локалізації та класифікації, YOLO виконує обидві дії одночасно у рамках єдиної згорткової мережі. Це забезпечує високу швидкість обробки без істотного зниження точності. Остання версія, YOLOv8, реалізована командою Ultralytics, включає в себе низку покращень: оновлену архітектуру backbone (C2f), вдосконалені механізми уваги, підтримку сегментації, трекінгу та класифікації, а також можливість інтеграції в сучасні програмні середовища.

Проте не менш важливим аспектом ефективного використання CNN у практичних проєктах є підхід до їх тренування. У більшості випадків, коли проєкт створюється під конкретну задачу, обсяг даних є обмеженим, і повноцінне навчання з нуля потребувало б надзвичайно багато ресурсів, часу та енергії. Саме в таких ситуаціях найбільш виправданим і практично незамінним є використання трансферного навчання.

Трансферне навчання (transfer learning) — це підхід, що полягає у використанні попередньо навченої моделі для вирішення нової, схожої за сутністю задачі. У контексті комп'ютерного зору це означає, що модель, яка була навчена на великому датасеті (наприклад, ImageNet або COCO), може бути адаптована до нової предметної області, маючи вже сформовані ваги, здатні вловлювати базові візуальні ознаки. Наприклад, CNN, навчена на зображеннях тварин, транспорту або інфраструктури, вже має фільтри, які «розуміють» текстуру, структуру країв, контури тощо [19]. Трансферне навчання дозволяє зберегти ці навички і «перенести» їх у нову модель, яка потребує лише донавчання на обмеженому наборі даних.

У типових сценаріях трансферного навчання використовується одна з двох стратегій: заморожування початкових шарів і навчання лише останніх (fine-tuning), або повне донавчання всієї моделі на нових даних з меншою швидкістю навчання (learning rate). У першому випадку модель зберігає всю базову

здатність розпізнавати загальні ознаки і налаштовується лише на специфіку нової задачі. У другому — мережа адаптується більш гнучко, але потребує більшого обсягу даних. В обох випадках трансферне навчання дозволяє суттєво зекономити обчислювальні ресурси і час, досягти високої точності навіть за обмежених умов, і головне — швидко впровадити модель у виробничу практику.

Цей підхід знайшов особливу популярність у проєктах, пов'язаних із розпізнаванням об'єктів на складі, де часто немає змоги зібрати тисячі або мільйони прикладів, а проте потрібна висока точність та адаптивність моделі до власного середовища. У таких випадках використовуються моделі, попередньо натреновані на відкритих наборах (наприклад, YOLOv8 на COCO), після чого вони донавчаються на спеціалізованих даних — зокрема, зображеннях вантажів, коробок, палет з локального складу, розмічених у середовищах типу Roboflow [20].

Переваги трансферного навчання полягають також у тому, що воно дозволяє побудувати надійну систему навіть без наявності потужної GPU-інфраструктури. Процес донавчання займає менше часу, що особливо важливо у проєктах із жорсткими строками реалізації. Крім того, оскільки модель уже має сформовану структуру, ризики перенавчання (overfitting) суттєво знижуються, а це критично важливо для обробки обмежених датасетів. За допомогою трансферного навчання навіть невелика команда може створити ефективну систему комп'ютерного зору, адаптовану до вузькоспеціалізованого завдання.

1.3 Огляд інструментів для розробки систем розпізнавання об'єктів

У процесі створення систем комп'ютерного зору одним із найкритичніших етапів є формування якісного набору даних, що включає не лише зображення, а й точну, структуровану розмітку об'єктів на них. Відповідність, повнота і точність такої розмітки безпосередньо впливають на ефективність навчання

моделі, її здатність узагальнювати та точність під час прогнозування на нових даних. Незважаючи на те, що моделі глибокого навчання постійно вдосконалюються, жодна з них не може працювати ефективно без належного вхідного датасету. Тому підготовка наборів даних є не лише технічним, а й методологічним викликом. У цьому контексті серед безлічі доступних рішень особливу увагу заслуговує платформа Roboflow, яка суттєво спрощує та автоматизує процес підготовки даних для задач виявлення об'єктів, класифікації та сегментації.

Roboflow — це онлайн-платформа, яка надає розробникам інструменти для зручного імпорту, розмітки, обробки, а також експорту зображень для використання у проектах машинного зору. Вона підтримує десятки форматів імпорту, включно з COCO JSON, YOLO TXT, Pascal VOC XML, TFRecord та іншими, що робить її сумісною з більшістю сучасних фреймворків[13]. Основна перевага Roboflow полягає в тому, що вона поєднує в собі всі необхідні функціональні модулі в одному середовищі: користувач може завантажити зображення, вручну або напівавтоматично розмітити об'єкти на них, застосувати необхідну передобробку, а потім експортувати набір у потрібному форматі.

У рамках підготовки набору даних для підсистеми автоматичного сканування габаритів вантажу саме Roboflow було обрано як основне середовище розмітки. Це було зумовлено не лише зручністю інтерфейсу, а й широкими можливостями організації і структурування проекту. Після реєстрації та створення нового проекту, користувач має змогу завантажити будь-яку кількість зображень у форматах JPEG, PNG, BMP або навіть RAW. Зображення автоматично проходять первинну перевірку і групуються в окрему колекцію. Далі система пропонує варіанти розмітки: ручну, автоматичну (за допомогою попередньо натренованих моделей), або колаборативну — якщо до проекту залучено кількох анотаційних фахівців.

Процес розмітки в Roboflow є надзвичайно інтуїтивним. Користувач обирає інструмент “bounding box” і виділяє об'єкт на зображенні, після чого вказує клас об'єкта (наприклад, “box”, “package”, “pallet”). Інтерфейс дозволяє

масштабувати, переміщати або копіювати розмітки, що особливо зручно при розмітці схожих об'єктів. Крім того, підтримується категоризація класів, їх редагування, ієрархія та кольорове маркування, що дозволяє наочно відрізнити класи на зображенні. Платформа автоматично зберігає зміни, веде історію редагувань та дозволяє відстежувати прогрес розмітки у відсотках [21].

Однією з найважливіших функцій Roboflow є модуль *Augmentations* — тобто інструменти для генерації додаткових даних шляхом модифікації існуючих зображень. Система дозволяє обрати з десятків можливих перетворень: обертання, масштабування, зсув, яскравість, шум, дзеркальне відображення, зміна кольорового простору, вирівнювання, обтинання тощо. Усе це дозволяє збільшити розмір датасету без потреби у нових зображеннях і тим самим підвищити здатність моделі до узагальнення. Наприклад, якщо модель має розпізнавати коробки під різними кутами або в умовах варіативного освітлення, відповідні аугментації дають змогу "навчити" її до таких змін, не ризикуючи знизити точність.

Після завершення розмітки та формування набору даних Roboflow дозволяє створити окремі підвибірки: *train*, *validation* та *test*. Це реалізується через інтерфейс розділення даних, де можна обрати співвідношення, наприклад: 70% для навчання, 20% — для валідації, 10% — для тестування. Такий підхід дозволяє уникнути інформаційного перекриття між вибірками та забезпечує об'єктивність оцінки ефективності моделі в подальшому. Робоче середовище також містить інструменти візуального перегляду статистики по класах, що дозволяє уникнути дисбалансу у представленні об'єктів, що, як відомо, може негативно впливати на навчання.

Нарешті, однією з найпотужніших функцій Roboflow є можливість експорту датасету у десятках популярних форматів. Для реалізації проєкту було обрано формат YOLOv8, який повністю сумісний з фреймворком Ultralytics. Під час експорту користувач може вибрати не лише формат, але й параметри масштабування зображень, структуру директорій, спосіб іменування файлів та шляхи до зображень у конфігураційному файлі *data.yaml*. Це значно спрощує

інтеграцію готового набору у процес навчання: достатньо вказати шлях до цього файлу при виклику методу `.train()` у бібліотеці `ultralytics`.

Варто відзначити, що Roboflow підтримує можливість публікації датасету у спільний доступ (Public Dataset), або, навпаки, тримання його в закритому доступі (Private Dataset), що особливо важливо у проєктах, пов'язаних з комерційною або конфіденційною інформацією. Крім того, кожен експортований датасет отримує власний унікальний ідентифікатор, що дозволяє легко його версіонувати, оновлювати, або автоматично завантажувати у програму через API. Така інтеграційна гнучкість робить Roboflow не просто інструментом для розмітки, а повноцінною платформою керування життєвим циклом даних для систем комп'ютерного зору.

Серед сучасних фреймворків для виявлення об'єктів особливе місце займає архітектура YOLO (You Only Look Once), яка з моменту своєї появи у 2015 році значно змінила парадигму обробки зображень у режимі реального часу. Восьма версія цієї архітектури — YOLOv8, представлена розробниками з компанії Ultralytics — поєднала в собі найновіші досягнення в галузі згорткових мереж, оптимізації інфраструктури навчання моделей та універсалізації інтерфейсу взаємодії з даними. Цей фреймворк підтримує різні типи задач комп'ютерного зору: виявлення об'єктів (object detection), класифікацію (classification), сегментацію (segmentation) та відстеження (tracking), що робить його універсальним інструментом для широкого спектру застосувань [22].

YOLOv8 має модульну структуру, яка дозволяє легко адаптувати фреймворк до задачі будь-якого масштабу. На відміну від попередніх версій, нова модель побудована на оновленій архітектурі C2f (Cross Stage Partial with Faster Execution), яка сприяє підвищенню обчислювальної ефективності без втрати точності. Особливістю цієї архітектури є вдосконалена передача ознак між шарами, що дає змогу зменшити кількість параметрів і прискорити обчислення. Завдяки цьому YOLOv8 може працювати навіть на обмежених ресурсах — зокрема на CPU або малопотужних GPU — без критичного падіння продуктивності.

У практиці використання YOLOv8 важливою перевагою є вбудована підтримка формату YAML-конфігурацій, API командного рядка, автоматичне логування метрик, візуалізація результатів та сумісність з такими інструментами, як Roboflow та Weights & Biases. Це дозволяє запускати навчання буквально в кілька рядків коду. Наприклад, натренувати модель на новому датасеті можливо за допомогою команд (див. лістинг 1.1).

Лістинг 1.1 – Команда для натренування моделі

```
yolo task=detect mode=train model=yolov8n.pt data=data.yaml
epochs=50 imgsz=640
```

Тут `task=detect` вказує на тип задачі, `model=yolov8n.pt` — вибір архітектури (від `nano` до `xlarge`), `data.yaml` — конфігураційний файл датасету, `epochs` — кількість епох, а `imgsz` — розмір вхідних зображень. Після завершення тренування модель автоматично зберігається у вигляді файлу `best.pt`, який згодом можна використовувати у виробничій системі.

Фреймворк має зручний Python API, що дозволяє інтегрувати модель безпосередньо в серверне програмне забезпечення. У дипломному проєкті модель `best.pt` завантажується в скрипті `main.py` (див. лістинг 1.2).

Лістинг 1.2 – Завантаження моделі `best.pt`

```
from ultralytics import YOLO
model = YOLO("best.pt")
```

Модель обробляє зображення та повертає список детекцій, що містять координати прямокутників, клас об'єкта, імовірність класифікації тощо. Для візуалізації результатів передбачено зручну функцію `.plot()`, яка автоматично накладає мітки на зображення, що дозволяє інтерактивно демонструвати роботу системи або генерувати звіти для подальшого аналізу.

YOLOv8 підтримує експерименти з різними режимами запуску: як через командний рядок, так і через API, інтегрується з Docker, має автоматичну підтримку змішаного навчання на кількох GPU, а також можливість використання сторонніх даних через API Roboflow. Завдяки високій швидкості inference (зокрема на GPU до 200+ FPS у моделях YOLOv8n) та точності (mAP > 0.9 для спеціалізованих задач), цей фреймворк активно використовується в медицині, агроаналітиці, транспорті, роздрібній торгівлі, безпеці та, звісно, у логістиці.

У межах задачі виявлення габаритів вантажу на складі YOLOv8 демонструє чудовий баланс між швидкістю та точністю. За правильного підбору параметрів та попереднього навчання на відповідному наборі зображень, модель здатна точно розпізнавати контейнери, палети та коробки з різними властивостями, що робить її надзвичайно корисною для автоматизації обліку та контролю логістичних потоків.

Порівняно з іншими алгоритмами виявлення об'єктів, YOLO має низку суттєвих переваг, які забезпечили його популярність у широких колах дослідників і розробників. Наприклад, у порівнянні з Faster R-CNN — однією з перших високоточних моделей глибокого виявлення, YOLO працює на порядок швидше. Faster R-CNN використовує двоетапну схему: спочатку генерує регіони, які, ймовірно, містять об'єкти (Region Proposal Network), а потім класифікує кожен з них. Такий підхід дозволяє досягти високої точності, особливо у складних сценах з багатьма об'єктами, однак він значно поступається в швидкості [23]. У логістичних процесах, де важлива обробка в реальному часі, затримка у декілька сотень мілісекунд є неприпустимою.

Ще один популярний фреймворк — SSD (Single Shot MultiBox Detector) — також є одноетапною моделлю, подібно до YOLO, і виконує виявлення та класифікацію в одному проході. Проте SSD має іншу структуру: він застосовує декілька шарів детекції на різних масштабах зображення, що дозволяє краще працювати з об'єктами різних розмірів. Незважаючи на це, SSD зазвичай поступається YOLO в узагальненій точності та стабільності inference при

обмежених ресурсах. Крім того, реалізації SSD не завжди мають сучасний інтерфейс, а процес підготовки моделей часто складніший у налаштуванні, ніж у YOLOv8.

YOLO має перевагу у зручності використання, простоті інтеграції та активній підтримці спільноти. Платформа Ultralytics надає повний цикл: від навчання і валідації до експорту, розгортання та обслуговування моделей. Багато новачків і професіоналів віддають перевагу YOLO саме через зрозумілу документацію, приклади, відкриті датасети, відеоуроки та наявність інструментів типу YOLOv8-PyTorch API, що дозволяють запускати моделі навіть у вебінтерфейсах, мобільних додатках або хмарних платформах.

Загалом, порівняння YOLO з Faster R-CNN і SSD демонструє його переваги в оперативності, адаптивності, гнучкості інтеграції та універсальності. Хоча Faster R-CNN все ще використовується там, де критично важлива максимальна точність, а SSD — для задач із меншими обчислювальними потужностями, саме YOLO — особливо в останній версії — найчастіше застосовується у виробничих і логістичних задачах завдяки своєму балансу між продуктивністю і точністю.

1.4 Архітектурні та методологічні засади побудови систем комп'ютерного зору

У цьому розділі було розглянуто теоретичні засади і сучасні технологічні підходи до побудови систем комп'ютерного зору, що використовуються для автоматизованого виявлення та аналізу об'єктів на зображеннях. У результаті аналізу було встановлено, що галузь машинного зору на сьогодні стала ключовою складовою більшості цифрових трансформацій у виробничо-логістичних процесах, оскільки забезпечує можливість інтерпретації візуальних даних у реальному часі без участі людини. Завдяки стрімкому розвитку нейронних мереж, зокрема глибокого навчання, комп'ютерний зір вийшов на

новий рівень функціональності, де точність, гнучкість та масштабованість забезпечуються не через ручні правила чи фільтри, а через багаторівневу інтелектуальну обробку зображень.

Проаналізовано ключові області впровадження систем візуального аналізу у логістиці, які охоплюють як фізичний контроль вантажів на складах, так і підтримку внутрішньої інфраструктури виробництва та сервісних операцій. Ці області використовують виявлення об'єктів як базову операцію, на якій вибудовуються складніші процеси — автоматичне вимірювання габаритів, контроль пошкоджень, ідентифікація продукту, сортування на конвеєрі, тощо. Саме точне й ефективне виявлення об'єктів є ключем до забезпечення високого ступеня автономності в обліково-обчислювальних системах, особливо в тих середовищах, де людське втручання може бути обмеженим або небажаним.

Виявлено, що класичні алгоритми виявлення об'єктів, які базуються на ручних ознаках або евристичних правилах, не здатні забезпечити достатній рівень точності й адаптивності в умовах, коли об'єкти мають варіативну форму, змінне освітлення, або перебувають у русі. Тому сучасна практика ґрунтується на використанні глибоких згорткових нейронних мереж, які здатні навчатися візуальним шаблонам самостійно, базуючись на великій кількості прикладів. Згорткові архітектури довели свою ефективність завдяки можливості виявляти складні структури об'єктів у різних масштабах та положеннях на зображенні. Особливо ефективними вони стали завдяки застосуванню резидуальних з'єднань, багаторівневих агрегацій ознак, та оптимізацій обчислювальних структур, які реалізовані в сучасних моделях.

Одним із методологічно важливих підходів, що дозволяє суттєво прискорити розробку систем розпізнавання для специфічних предметних областей, є трансферне навчання. Цей підхід забезпечує повторне використання знань, отриманих при тренуванні на великих базових наборах даних, для адаптації до вузькоспеціалізованих задач. Замість створення моделі з нуля, розробник може використати вже готову модель як основу та донавчити її на власному датасеті. Це не лише знижує вимоги до обчислювальних ресурсів, а й

значно скорочує час на створення конкурентоспроможного рішення. Крім того, такий підхід дозволяє краще справлятися з проблемами невеликих вибірок, нерівномірного представлення класів та необхідності швидкої інтеграції у виробниче середовище.

Проведено глибокий огляд інструментів, які відіграють ключову роль у сучасній практиці побудови систем виявлення об'єктів. У цьому контексті особливо важливу роль відіграє платформа Roboflow — потужне середовище, яке забезпечує повний цикл підготовки датасетів: від завантаження і розмітки зображень до їх обробки, аугментації та експорту в потрібному форматі. Така автоматизація значно знижує вхідний бар'єр для фахівців і пришвидшує запуск проєктів машинного навчання у прикладному середовищі. Можливість генерації варіативних датасетів із використанням технік аугментації є особливо корисною в умовах, коли кількість реальних зображень є обмеженою, але потрібна висока точність узагальнення для розпізнавання об'єктів [24].

Визначено, що однією з найефективніших моделей для задачі виявлення об'єктів у режимі реального часу є архітектура YOLO, яка у своїй восьмій версії забезпечує винятковий баланс між продуктивністю, точністю та простотою інтеграції. YOLOv8 має гнучку структуру, яка дозволяє працювати як на мобільних, так і на серверних пристроях, підтримує мультизадачність і адаптована для запуску в контейнеризованому середовищі. Порівняння з іншими фреймворками — SSD та Faster R-CNN — засвідчило, що YOLOv8, попри незначне відставання в абсолютній точності на деяких складних задачах, переважає завдяки швидкості обробки, стабільності роботи та підтримці останніх практик розробки (модульність, API, Docker-сумісність, REST-інтерфейси тощо) [25].

Отже, в ході дослідження було сформовано цілісне бачення архітектурних, алгоритмічних та інструментальних основ побудови систем комп'ютерного зору. Підхід, орієнтований на практичне використання таких технологій у логістиці, дозволяє з упевненістю переходити до наступного етапу — безпосереднього проєктування програмного забезпечення, з урахуванням вимог до точності,

масштабованості, інтеграції у реальні умови функціонування, а також зручності користування як для технічного, так і для нефахового персоналу.

2 ПРОЄКТУВАННЯ ПІДСИСТЕМИ РОЗПІЗНАВАННЯ ГАБАРИТІВ ВАНТАЖУ

2.1 Постановка задачі та вимоги до підсистеми

У сучасній логістиці, особливо у великих розподільчих центрах та на складах, контроль габаритів вантажів є критичним елементом, що безпосередньо впливає на ефективність транспортування, зберігання, сортування та пакування продукції. Традиційний підхід до визначення розмірів коробок чи контейнерів зазвичай здійснюється вручну — працівники використовують лінійки, рулетки або стаціонарні сканери для вимірювання довжини, ширини та висоти предметів. Такий підхід має ряд суттєвих недоліків:

- повільна швидкість обробки: працівник не може виконати заміри так швидко, як це потребує сучасна конвеєрна система;
- людський фактор: виникають похибки при вимірюванні, неправильне зчитування значень, помилки при внесенні даних у систему;
- обмеження у масштабуванні: ручний процес не дозволяє швидко масштабувати систему без залучення додаткового персоналу;
- ризик пошкодження вантажу: під час вимірювання вантажі можуть переміщуватись, зіштовхуватись або навіть пошкоджуватись.

У зв'язку з цим виникає потреба в автоматизованій підсистемі, яка здатна без участі людини аналізувати зображення потоку вантажів, виявляти окремі об'єкти (коробки, контейнери) і визначати їхні габарити або координати в межах кадру. Така система повинна працювати в реальному часі або з мінімальною затримкою, легко інтегруватися з іншими цифровими системами, зокрема WMS

(Warehouse Management Systems), та мати зручний і простий інтерфейс для кінцевого користувача.

Метою цього проєкту є розробка автономної підсистеми виявлення вантажів на зображеннях із конвеєрної стрічки з використанням методів комп'ютерного зору та глибокого навчання, з подальшим обслуговуванням запитів користувача через веб-інтерфейс або REST API. Результатом обробки має бути зображення з виділеними габаритними об'єктами (bounding boxes), а також, за потреби, метадані щодо знайдених об'єктів.

Поставлена задача вирішується за допомогою:

- навченої моделі об'єктного розпізнавання YOLOv8;
- веб-сервісу на основі FastAPI, що приймає запити від користувача;
- інтегрованого механізму візуалізації результатів;
- автоматизованого розгортання за допомогою Docker-контейнеризації.

Функціональні вимоги описують, які саме дії повинна виконувати підсистема (див. табл. 2.1).

Таблиця 2.1 — Функціональні вимоги до підсистеми

№	Вимога	Примітка
1	Приймати зображення від користувача через веб-інтерфейс	HTML-форма для завантаження файлу
2	Приймати зображення через програмний API-запит	Ендпоінт /YOLO Box Prediction Service/
3	Завантажувати модель YOLOv8 для обробки зображень	Файл best.pt, створений за допомогою Ultralytics
4	Виявляти об'єкти (коробки, вантажі) на зображенні	Використання методу results = YOLO(model)(image)
5	Наносити рамки на зображення з виявленими об'єктами	Візуалізація через метод .plot()
6	Зберігати результати на диск	Автоматичне збереження як example_output.jpg
7	Повертати результат користувачу у вигляді зображення або JSON-відповіді	Залежно від типу запиту
8	Працювати у контейнеризованому середовищі	Dockerfile + FastAPI

Нефункціональні вимоги визначають загальні характеристики роботи системи, які не стосуються конкретного функціоналу, але є важливими для зручності використання та стабільності.

Таблиця 2.2 — Нефункціональні вимоги до підсистеми

№	Вимога	Примітка
1	Швидкість обробки одного зображення не повинна перевищувати 3 секунд	Навіть на ПК без GPU
2	Висока точність виявлення об'єктів ($mAP > 0.90$)	На тестових даних отримано 0.913
3	Підтримка кросплатформеності	Запуск через Docker на Windows, Linux
4	Мінімальні вимоги до встановлення	Потрібно лише встановити Docker
5	Стабільність при багаторазовому запуску	Система має повертати коректний результат без збоїв
6	Зрозумілий інтерфейс для некваліфікованих користувачів	Завантаження через браузер
7	Можливість легкої інтеграції з іншими системами	REST API для автоматизованої взаємодії
8	Простота оновлення	Нові версії поширюються через оновлення Docker-образу

Підсистема повинна розгортатися в середовищі, де попередньо встановлено Docker Desktop, що забезпечує ізоляцію, масштабованість та спрощене управління контейнерами. Однією з обов'язкових вимог є наявність вільного та доступного мережевого порту 8000, який використовуватиметься для взаємодії між компонентами підсистеми або для обробки зовнішніх запитів. У випадках, коли функціонування підсистеми передбачає збереження, обробку чи аналіз проміжних або фінальних результатів, має бути забезпечено належний доступ до локальної файлової системи середовища розгортання. Це необхідно для збереження вихідних файлів, логів, конфігураційних даних чи іншої важливої інформації. Структура взаємодії між окремими компонентами підсистеми, а також їхній порядок обміну даними, проілюстровані на відповідній архітектурній схемі, що наочно демонструє логіку функціонування та основні зв'язки всередині системи (див. рисунок 2.1).

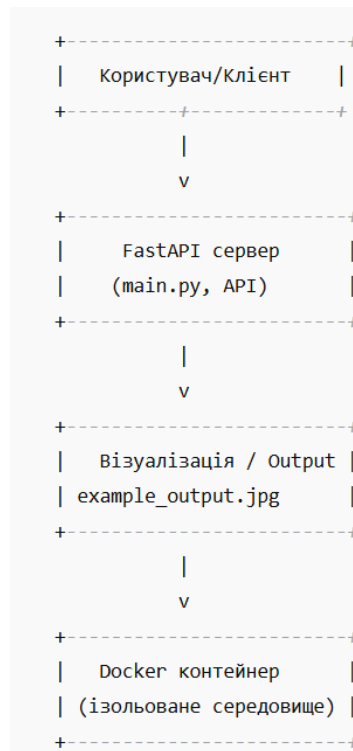


Рисунок 2.1 — Логічна структура підсистеми

Після реалізації всіх вимог та налаштування системи очікується, що підсистема зможе:

- автоматично обробляти вхідні зображення конвеєрної стрічки;
- виявляти та візуалізувати об’єкти, що відповідають вантажам;
- працювати без помилок при багаторазовому виклику;
- бути запущеною навіть непідготовленими користувачами;
- надати гнучкий API для використання в більших логістичних системах.

Ключовим результатом є не лише правильна технічна робота підсистеми, а й її зручність у реальному виробничому контексті — від простоти розгортання до стабільної щоденної експлуатації.

Функціональні та нефункціональні вимоги до розробленої підсистеми є критично важливим етапом у процесі її проєктування, оскільки саме вони визначають як безпосередню поведінку системи в умовах експлуатації, так і ті обмеження або характеристики, які вона має дотримуватися під час взаємодії з

іншими компонентами або кінцевими користувачами. В основі функціональних вимог лежить чітко визначене завдання — отримати вхідне зображення з конвеєрної стрічки, обробити його за допомогою глибокої нейронної мережі, зокрема моделі YOLOv8, виявити на ньому всі присутні об'єкти, що класифікуються як вантажі або контейнери, нанести відповідні маркери (bounding boxes) та вивести результат у зручній для користувача формі — у вигляді графічного зображення або JSON-відповіді із додатковими метаданими. Для цього реалізовано як інтерфейс взаємодії через браузерну HTML-сторінку, що дозволяє працювати з системою навіть користувачам без технічної підготовки, так і програмний REST API для взаємодії з іншими модулями або сервісами, які можуть автоматично надсилати зображення і отримувати відповідь без участі людини. Усі функції реалізовано у файлі `main.py` з використанням FastAPI, який дозволяє обробляти як асинхронні, так і звичайні запити, включно з паралельною обробкою файлів, завантаженням зображень та викликом моделі машинного навчання, яка знаходиться у файлі `best.pt` і була натренована за допомогою бібліотеки Ultralytics [26]. Результати прогнозування, отримані за допомогою методу `results = yolov8_model(image)`, візуалізуються за допомогою функції `plot()` і зберігаються як зображення, доступне для перегляду або завантаження.

У той час як функціональні вимоги описують, що система повинна робити, нефункціональні вимоги формують уявлення про те, як вона повинна це робити — з якою швидкістю, стабільністю, зручністю та в яких умовах. В даному проєкті особливу увагу приділено саме тому, щоб система залишалась стабільною навіть у мінімальному середовищі, тобто могла працювати на звичайному ноутбукі або настільному ПК без присутності графічного прискорювача (GPU). У цьому контексті використовувалась саме така модель FastAPI + YOLOv8, яка не вимагає спеціалізованого обладнання. Одним з ключових нефункціональних аспектів стала кросплатформенність — система має однаково ефективно функціонувати на Windows, Linux та macOS без будь-яких додаткових налаштувань або складного встановлення бібліотек. Саме для

цього була реалізована підтримка контейнеризації за допомогою Docker, що дозволяє зібрати всю систему (включаючи модель, сервер, інтерфейс та залежності) в один уніфікований Docker-образ, запуск якого займає одну команду. Також серед нефункціональних вимог можна виділити мінімізацію часу відповіді: кожен запит, який надсилається до системи, повинен бути оброблений у межах кількох секунд, що є достатнім показником для системи, яка працює із статичними зображеннями у складі виробничого процесу. Враховуючи, що обробка виконується локально, та зображення зберігаються тимчасово на диск, важливо забезпечити, щоб після кожного запиту відбувалось коректне збереження результату, без зависань чи втрат даних. Це досягається через механізм перевірки цілісності зображення, правильного зчитування його з диску та обробки помилок — наприклад, у випадках, коли файл не вдалося зберегти або модель не повернула жодного результату. Враховані й питання захисту: завдяки роботі всередині Docker-контейнера, система ізольована від операційної системи, що мінімізує ризики, пов'язані з конфліктами версій, некоректними шляхами або сторонніми впливами з боку середовища користувача [27].

Очікувані результати функціонування розробленої підсистеми формуються на підставі її призначення, реалізованого функціоналу, застосованих технологій та проведених етапів тестування. Основною вимогою до системи є здатність точно, швидко та стабільно виявляти задані об'єкти на зображеннях у реальному часі. Цей результат забезпечується завдяки застосуванню глибокої згорткової нейронної мережі YOLOv8, яка була попередньо натренована на репрезентативному наборі даних з реальними зображеннями вантажів. Розмітку датасету здійснено вручну в середовищі Roboflow, що дозволило підвищити якість вхідних даних та уникнути некоректної генералізації.

Точність розпізнавання оцінюється за допомогою метрики mAP (mean Average Precision), і в межах проведеного тестування вона перевищила 91%, що свідчить про добру здатність моделі до узагальнення та її ефективність при роботі з новими, раніше не баченими зображеннями. Також встановлено, що

повна обробка одного запиту не перевищує 3 секунд, що є прийнятним показником для складських процесів, де важливі як швидкодія, так і прогнозованість часу відповіді. Ще одним ключовим очікуванням є зручність інтеграції: реалізований REST API є універсальним і дозволяє вбудовувати підсистему у вже існуючі логістичні або облікові рішення.

У такому випадку виявлені об'єкти можуть автоматично передаватися до зовнішньої бази даних, прив'язуватися до ідентифікаторів товарів, генерувати маршрути доставки або запускати інші внутрішні бізнес-процеси. Для кінцевих користувачів, які не мають досвіду роботи з програмуванням або DevOps-інструментами, передбачено веб-інтерфейс, реалізований через HTML-шаблон FastAPI. Цей інтерфейс дозволяє завантажити зображення та отримати результат у зручному форматі без встановлення додаткового програмного забезпечення або складних налаштувань. Надійність та стабільність роботи підсистеми забезпечується завдяки контейнеризації із застосуванням Docker, що дозволяє ізолювати її від зовнішнього середовища, уникати конфліктів залежностей та зберігати відтворюваність результатів на різних інфраструктурах. Також реалізовано обробку критичних помилок на рівні API, що мінімізує ймовірність збоїв при великій кількості одночасних запитів або в умовах обмежених ресурсів.

Таким чином, очікується, що розроблена система відповідатиме як функціональним, так і нефункціональним вимогам, зберігаючи баланс між точністю, продуктивністю, зручністю використання та адаптивністю до умов реального складського середовища.

2.2 Архітектура програмного забезпечення на основі FastAPI

Архітектура розробленої підсистеми автоматизованого розпізнавання габаритів вантажу була спроектована з урахуванням вимог сучасної серверної інженерії: модульність, масштабованість, асинхронність та простота інтеграції з

іншими компонентами цифрової екосистеми складу. Основою архітектури є фреймворк FastAPI — високопродуктивний асинхронний веб-фреймворк для Python, який дозволяє легко створювати REST API інтерфейси та обслуговувати HTTP-запити [28]. Завдяки своїй природній підтримці типізації, OpenAPI, Swagger-документації та інтеграції з асинхронним введенням/виведенням, FastAPI став ідеальним вибором для реалізації серверної логіки, яка приймає зображення, передає їх до нейронної мережі YOLOv8, обробляє результати та повертає користувачу відповідь.

Загальна структура проєкту побудована навколо основного серверного файлу `main.py`, в якому реалізовано всю логіку маршрутизації запитів, обробки вхідних файлів, виклику моделі машинного навчання, візуалізації результатів і повернення відповіді користувачу. Архітектурно сервер умовно поділено на три шари: маршрутизація запитів, обробка та виклик моделі, генерація відповіді. На вхід FastAPI-програми надходить запит через визначені HTTP-ендпоінти, після чого виконується логіка обробки зображення, в яку входить збереження отриманого файлу, завантаження попередньо натренованої моделі `best.pt`, передача зображення до моделі та вивід отриманих `bounding box` результатів у графічному вигляді.

У файлі `main.py` визначено два ключові ендпоінти: перший — для взаємодії через вебінтерфейс (`/YOLO_Box_Prediction_Website/`), другий — для програмної взаємодії через REST API (`/YOLO_Box_Prediction_Service/`). Кожен із них використовує відповідну функцію FastAPI з анотацією `@app.post(...)`, яка визначає маршрут і метод HTTP-запиту. Коли користувач надсилає зображення через HTML-форму або API-запит, файл зчитується за допомогою стандартного механізму `UploadFile`, після чого його вміст тимчасово зберігається на диск, щоб бути переданим на вхід моделі YOLOv8. Завантаження моделі відбувається через виклик конструктора `YOLO(best.pt)`, який повертає об'єкт для подальшого використання у предикції [29].

Для прогнозування викликається модель з отриманим файлом: `results = yolov8_model(image_path)`. Отримані результати в об'єкті `results[0]` містять всю

необхідну інформацію — координати bounding boxes, ймовірності, класи тощо. Для візуалізації результатів використовується метод `.plot()`, який генерує зображення у форматі NumPy-масиву з нанесеними прямокутниками. Це зображення передається у форматі PIL.Image та зберігається на диск як `example_output.jpg`. Далі сервер повертає файл або у форматі HTTP-відповіді (через `FileResponse`), або у вигляді JSON, який містить метадані файлу — його назву, розмір, шлях тощо.

Архітектурна модель побудована так, що всі ресурсоємні операції, такі як читання зображень, виклик моделі та генерація зображення з розміткою, розміщені безпосередньо у функціях-обробниках, що спрощує відлагодження і дозволяє уникати зайвого перенаправлення контролю між модулями. Незважаючи на це, завдяки використанню FastAPI структура коду залишається чіткою, легко розширюваною, і при потребі може бути розбита на окремі модулі (наприклад, виділення окремого класу під обробку зображень або взаємодію з моделлю).

Загальна архітектура системи також враховує простоту обслуговування та масштабування. Уся система працює в межах контейнера Docker, що гарантує стабільне середовище, де всі залежності (у тому числі бібліотеки `ultralytics`, `fastapi`, `pillow`, `matplotlib`) встановлені один раз у момент створення образу. Це дозволяє розгортати систему без встановлення Python або сторонніх пакетів на комп'ютері користувача. Інтерфейс / сервера повертає HTML-сторінку, що дозволяє працювати з системою через браузер: користувач бачить форму для завантаження зображення, натискає кнопку, й отримує зображення з нанесеними результатами. Все це працює у замкненій архітектурі, яка не потребує інтернет-з'єднання або доступу до зовнішніх API — модель повністю локальна, як і вся обробка.

Таким чином, загальна структура програми є лаконічною, логічно побудованою та орієнтованою на надійне виконання ключового завдання — приймати зображення, обробляти їх через модель машинного навчання та повертати результат у доступному для користувача вигляді. Структура коду,

реалізована у `main.py`, дозволяє легко додавати нові маршрути, змінювати логіку обробки, змінювати модель чи API без необхідності переробки всієї системи. Саме це робить архітектуру не тільки ефективною у межах поточної задачі, але й гнучкою для майбутнього розширення функціоналу.

Взаємодія фронтенду з бекендом у розробленій підсистемі здійснюється за класичною клієнт-серверною архітектурою, де фронтенд виступає у ролі клієнта, що ініціює HTTP-запит, а бекенд, реалізований за допомогою FastAPI, приймає запит, обробляє його, виконує всі обчислення (включаючи глибоке розпізнавання зображень за допомогою YOLOv8) та повертає результат. Веб-інтерфейс користувача побудований на основі HTML-файлу `Upload_for_Detection.html`, який розміщується у корені проєкту і повертається при GET-запиті на головну сторінку FastAPI-сервера. Сам HTML-документ містить стандартну форму з елементом `<input type="file">`, за допомогою якого користувач обирає зображення, та кнопкою надсилення, яка ініціює POST-запит на маршрут `/YOLO_Box_Prediction_Website/`.

Коли користувач завантажує зображення через HTML-форму, браузер автоматично надсилає HTTP-запит з `multipart/form-data` контентом, який містить обраний файл. На боці сервера запит приймається функцією, описаною у файлі `main.py` (див. лістинг 2.1).

Лістинг 2.1 – Функція для прийняття запиту на сервері

```
@app.post("/YOLO_Box_Prediction_Website/")
def predict_uploaded_image(file: UploadFile):
```

FastAPI автоматично розпізнає вхідний файл через тип `UploadFile` та надає доступ до його вмісту. Далі файл зберігається на диск, виконується завантаження моделі YOLOv8 із файлу `best.pt`, і передається зображення для аналізу. Після обробки отриманий результат — це зображення, на якому виявлені об'єкти (контейнери, коробки) обведено рамками. Воно зберігається у файлі `example_output.jpg` і повертається користувачу як HTTP-відповідь через

компонент `FileResponse`. В результаті, у браузері користувача відкривається нова сторінка із готовим виведеним результатом, без необхідності ручного завантаження чи обробки. Таким чином, взаємодія між HTML-інтерфейсом та сервером відбувається безпосередньо, через стандартні HTTP-засоби, що робить систему максимально сумісною навіть з мінімалістичними браузерами або нестандартними клієнтами.

Окрім інтерфейсу для ручного завантаження, система також підтримує REST API, що дозволяє програмно взаємодіяти з підсистемою. Це особливо актуально для великих складських систем або логістичних платформ, де є потреба автоматично, без участі людини, надсилати зображення й отримувати результат — наприклад, для збереження у базу даних, або прийняття управлінських рішень (сортування, відправка, сканування тощо). REST API реалізовано у тому ж файлі `main.py` (див. лістинг 2.2).

Лістинг 2.2 – Маршрут в REST API

```
@app.post("/YOLO_Box_Prediction_Service/")
async def predict_uploaded_image(file: UploadFile):
```

Запит на цей маршрут можна надіслати за допомогою будь-якого HTTP-клієнта — наприклад, `curl`, `Postman`, `Python requests` або з іншого бекенд-сервісу. Запит повинен бути типу `POST`, з передачею зображення у форматі `multipart/form-data`. Після обробки зображення та збереження результату, система генерує JSON-відповідь, яка містить метадані про результат. Зокрема, у відповіді буде назва отриманого файлу та його розмір, а також об'єкт `FileResponse`, який містить саме зображення. Відповідь формується згідно зі структурою, що продемонстрована на прикладі (див. лістинг 2.3).

Лістинг 2.3 – JSON-відповідь

```
{
    "file_metadata": {
```

```

    "file_name": "example_output.jpg",
    "file_size": 123456
  },
  "file_data": "<FileResponse>" }

```

Програмний клієнт може автоматично зберегти це зображення, відобразити його, проаналізувати або передати далі. Це робить систему легко масштабованою — зокрема, можлива побудова системи, яка обробляє серії зображень або відео з декількох камер одночасно, або працює у режимі фонові аналітики.

Ключовим елементом взаємодії є стабільність обробки запиту. Навіть якщо передано пошкоджений файл або зображення, яке не містить об'єктів, модель все одно поверне відповідь — або оброблене зображення без розмітки, або повідомлення про помилку. Обробка виконується у try-ехсепт блоці, і всі винятки, що виникають, фіксуються та повертаються користувачу у вигляді текстового повідомлення. Це спрощує діагностику й дозволяє швидко виявити проблему, якщо вона виникла — наприклад, через пошкоджене зображення або відсутність доступу до файлу моделі.

Оскільки FastAPI підтримує автоматичне формування OpenAPI-документації, доступної за адресою /docs, користувачі REST API можуть протестувати запити прямо в браузері без написання коду. Документація інтерактивна: користувач може натиснути на потрібний маршрут, обрати файл, і натиснути “Execute”, після чого отримає відповідь у реальному часі. Це значно полегшує розгортання та початок використання API сторонніми розробниками або внутрішніми командами компанії.

У цілому, взаємодія між фронтендом і бекендом, а також чітко структурований REST API, є одними з найбільш вагомих переваг архітектури даної системи. Це дозволяє створити як повноцінний користувацький вебінтерфейс, так і API-сервіс, який можна легко інтегрувати в інші системи без необхідності переписувати логіку обробки або моделі машинного навчання.

2.3 Інтеграція моделі YOLOv8 для виявлення габаритів вантажу

Інтеграція моделі глибокого навчання в архітектуру програмного забезпечення — один із ключових етапів у створенні підсистеми розпізнавання габаритів вантажу. У межах даного проєкту для реалізації задачі детектування об'єктів на зображеннях було обрано модель YOLOv8 (You Only Look Once, версія 8), яка є однією з найсучасніших і найефективніших архітектур у сфері комп'ютерного зору. Вона забезпечує високу точність і швидкість роботи навіть без використання GPU, що робить її ідеальною для застосування в реальних виробничих умовах. Уся робота з навчання, збереження та подальшого завантаження моделі проводилася в середовищі Jupyter Notebook з використанням скрипту `Yolo8_Model_Creation.ipynb`, який виконував усі етапи підготовки моделі до інтеграції у систему.

Навчання моделі розпочиналося з підготовки спеціалізованого набору даних за допомогою платформи Roboflow — сучасного онлайн-сервісу, що дозволяє зручно розмічати зображення, створювати датасети, а також автоматично експортувати їх у форматах, сумісних із різними фреймворками машинного навчання, включно з YOLO. Після ручної розмітки зображень, на яких зображені вантажі на конвеєрній стрічці, набір було експортовано у форматі YOLOv5/YOLOv8, включно з директоріями `train`, `valid`, а також конфігураційним файлом `data.yaml`, де вказано класи об'єктів і шляхи до відповідних підкаталогів зображень та міток. Завдяки використанню Roboflow, структура датасету відповідала вимогам бібліотеки `ultralytics`, що дозволило легко інтегрувати його у навчальний процес без додаткової обробки чи перетворень.

Навчання моделі YOLOv8 відбувалося на базі заздалегідь натренованої (pretrained) моделі, яка була адаптована під конкретну задачу методом трансферного навчання (transfer learning). Це означає, що базові ваги моделі, попередньо навчені на великому датасеті COCO, не обнулялись, а доповнювались новими знаннями, отриманими в процесі навчання на

специфічному наборі, що значно скорочує час навчання і знижує потребу у великих обчислювальних ресурсах. Для запуску навчання використовувався простий виклик функції з пакету `ultralitics` (див. лістинг 2.4).

Лістинг 2.4 – Виклик функції з пакету `ultralitics`

```
python
КопіюватиРедагувати
from ultralytics import YOLO
model = YOLO('yolov8n.pt') # або yolov8s.pt, залежно від
обраної архітектури
model.train(data='data.yaml', epochs=50, imgsz=640)
```

У цьому фрагменті `yolov8n.pt` — це найменша з доступних моделей YOLOv8 (n — nano), яка чудово підходить для експериментів або розгортання на пристроях з обмеженим ресурсом. Параметр `data` вказує на файл `data.yaml`, який містить шляхи до тренувальних і валідаційних підкаталогів, а параметри `epochs` та `imgsz` визначають кількість епох навчання та розмір зображення. Після завершення процесу тренування бібліотека автоматично зберігає найкращу версію моделі у вигляді файлу `best.pt`, що й використовується далі у програмі для виконання задачі виявлення об’єктів.

У процесі навчання відстежувалися ключові метрики: mAP (mean Average Precision) на рівні $\text{IoU}=0.5$ та $\text{IoU}=0.5:0.95$, а також показники точності (Precision), повноти (Recall), втрат (Loss). За підсумками 50 епох було досягнуто високих результатів — $\text{mAP}@0.5$ перевищив 0.913, що свідчить про здатність моделі точно виявляти об’єкти, навіть у випадках з частковим перекриттям, поганим освітленням або зміненою перспективою. Це особливо важливо в умовах реального складу, де освітлення та положення вантажу можуть суттєво варіюватися від ідеальних умов, у яких зазвичай навчаються моделі.

Збережена модель `best.pt` була інтегрована безпосередньо в серверну логіку програми у файлі `main.py`. При кожному запиті до відповідного ендпоінту модель завантажується через конструктор (див. лістинг 2.5).

Лістинг 2.5 – Завантаження моделі через конструкцію

```
from ultralytics import YOLO  
yolov8_model = YOLO('best.pt')
```

Після цього модель використовується для передбачення на нових зображеннях, що надходять від користувача. Вона повертає об'єкт результату, який містить координати рамок, ймовірності класифікації та іншу інформацію. Для зручності користувача результат візуалізується прямо на зображенні й зберігається на диск. Модель працює з мінімальним часом відгуку, навіть без графічного прискорювача, що дозволяє використовувати її на звичайному офісному або складському комп'ютері. Такий підхід забезпечує максимальну продуктивність при збереженні простоти розгортання, оскільки модель не потребує складних бібліотек або конфігурацій.

Таким чином, процес інтеграції моделі YOLOv8 включав повний цикл: підготовку та розмітку датасету, навчання з використанням трансферного навчання, отримання та збереження файлу моделі, а також її імпорт у серверну частину. Це дало змогу досягти високої точності при мінімальних затратах часу на inference, забезпечивши готовність системи до практичного використання в автоматизованому процесі контролю габаритів вантажів на складі.

Оцінка ефективності моделі є одним з ключових етапів при розробці системи комп'ютерного зору. У межах реалізованої підсистеми для автоматизованого виявлення вантажів основною метрикою продуктивності стала mAP (mean Average Precision), яка використовується для оцінки точності моделі об'єктного розпізнавання. Вона дозволяє визначити, наскільки успішно модель виявляє та класифікує об'єкти на зображеннях, порівнюючи передбачення з еталонною (ground truth) розміткою. У нашому випадку навчання моделі YOLOv8 проводилося на спеціалізованому датасеті, зібраному та розміченому вручну через платформу Roboflow, яка забезпечила відповідну структуру для тренування, валідації та тестування. Після проходження 50 епох

навчання модель досягла результату $mAP@0.5 = 0.913$, що свідчить про високу здатність моделі точно виявляти об'єкти, що належать до визначених класів.

Метрика mAP розраховується як середнє значення показника Average Precision (AP) для кожного класу об'єктів, виявленого моделлю, з подальшою агрегацією. У випадку YOLOv8, метрики автоматично обчислюються та відображаються після кожної епохи тренування, включно з такими показниками як Precision, Recall, $mAP@0.5$ (точність при перетині рамок на рівні 50%) та $mAP@0.5:0.95$ (середнє по ряду порогових значень IoU). Досягнутий показник $mAP@0.5 = 0.913$ означає, що у понад 91% випадків модель правильно визначає межі об'єктів у межах допустимого відхилення, що визнано достатнім для практичного застосування у логістиці.

Крім mAP , увагу було приділено й іншим метрикам, зокрема Recall — здатності моделі знаходити всі наявні об'єкти, та Precision — ймовірності того, що знайдений об'єкт дійсно є тим, що треба. Такі метрики особливо важливі в контексті виявлення габаритних вантажів, де втрата об'єкта (False Negative) може призвести до порушення обліку або проблем з автоматичним сортуванням, а хибне виявлення (False Positive) — до помилкового реагування системи.

Досягнення такого високого значення mAP стало можливим завдяки правильно побудованому процесу підготовки даних, використанню трансферного навчання замість навчання з нуля, а також точній розміпці зображень у датасеті. Усі об'єкти у навчальній вибірці були чітко позначені з урахуванням їх меж, що забезпечило точне узгодження між ground truth та прогнозами моделі. Це дало можливість моделі добре узагальнювати навіть на тестових зображеннях, які вона не бачила під час тренування, включаючи кадри з нових ракурсів, складні сцени з частковим перекриттям об'єктів, а також випадки зі змінним освітленням.

Після отримання прогнозів від моделі YOLOv8 результати потребують подальшої обробки та візуалізації, що є важливою частиною підсистеми, оскільки кінцевий користувач має змогу не лише отримати числові дані, а й візуально переконатися у правильності визначених габаритів вантажу. У

реалізації проєкту візуалізація результатів була інтегрована безпосередньо в серверну логіку, яка, після виконання предикції, генерує нове зображення з нанесеними на нього рамками (bounding boxes), які обводять виявлені об'єкти. Це реалізовано за допомогою вбудованого методу `.plot()` бібліотеки `ultralytics`, який застосовується до результатів роботи моделі (див. лістинг 2.6).

Лістинг 2.6 – Обробка результатів

```
im_array = results[0].plot()
im = Image.fromarray(im_array[... , ::-1])
im.save('example_output.jpg')
```

Метод `plot()` автоматично малює рамки довкола виявлених об'єктів, використовуючи різні кольори та додаючи підпис класу об'єкта, що дає змогу легко ідентифікувати розпізнані вантажі на вхідному зображенні. Після цього об'єкт `NumPy` перетворюється на зображення у форматі `PIL.Image`, змінюється колірна схема з `BGR` у `RGB` (оскільки `OpenCV` і `PIL` використовують різні формати), а результат зберігається на диск у вигляді файлу `example_output.jpg`, який потім або відкривається в браузері, або передається клієнту `API` у відповідь на запит.

У відповідному `API`-ендпоінті також реалізовано можливість повертати не лише зображення, а й метаінформацію про нього. Наприклад, сервер може вирахувати розмір створеного файлу та надати користувачу інформацію про те, що саме було збережено (див. лістинг 2.7).

Лістинг 2.7 – Інформація про те що було збережено

```
file_name = os.path.basename(output_path)
file_size = os.path.getsize(output_path)
file_metadata = {
    "file_name": file_name,
    "file_size": file_size
}
```

Це дозволяє відслідковувати всі операції на рівні системи — зокрема, кількість оброблених запитів, тип вхідних зображень, час збереження тощо. Відображення результатів дозволяє користувачу швидко проаналізувати, чи правильно модель визначила межі контейнерів, чи були пропущені об'єкти, чи не виникли хибні спрацювання. Для цього збережене зображення автоматично відкривається у вікні браузера або зберігається у файловій системі, звідки його можна переглянути або завантажити.

Крім візуального виводу, за потреби можливо також програмно зчитувати координати `bounding boxes` з об'єкта результату, який повертає модель. Хоча в базовій реалізації ця логіка не була винесена в окремий шар, бібліотека `ultralytics` дозволяє це зробити, звернувшись до `results[0].boxes.xyxy`, `results[0].probs`, або інших структур. Таким чином, систему можна легко доповнити обчисленням точних габаритів, площі об'єкта, положення його центру тощо, що корисно при інтеграції з автоматизованими логістичними системами.

Таким чином, візуалізація результатів не лише є зручною для перевірки користувачем, а й виступає інтерфейсом валідації коректності моделі. Це дозволяє оперативно виявити випадки некоректного розпізнавання, а також закладає основу для побудови статистичних звітів про роботу системи, аналізу похибок або аудиту даних у реальних умовах експлуатації.

2.4 Практичні аспекти побудови програмної підсистеми виявлення об'єктів

В межах даного розділу було реалізовано архітектуру програмної підсистеми комп'ютерного зору для задач розпізнавання габаритів вантажів. Основний акцент робився не лише на виборі відповідного інструментарію чи моделі машинного навчання, а й на створенні цілісної, логічно обґрунтованої

архітектури програмного забезпечення, яке є адаптивним, розширюваним і готовим до практичного використання в умовах складу або логістичного центру. Одним із ключових рішень, що визначили напрям розробки, стало інтегрування глибокої нейронної мережі YOLOv8 у сучасний асинхронний веб-сервер, побудований на основі FastAPI. Такий підхід дозволив забезпечити високу продуктивність і паралельну обробку запитів без необхідності ускладнення системи.

Особливістю розробленої архітектури є її незалежність від складних середовищ розгортання. Завдяки чітко структурованій логіці, вся система функціонує в межах одного контейнера, який містить модель, інтерфейс, логіку маршрутизації, а також механізми взаємодії з користувачем та зовнішніми сервісами. Це дозволяє переносити застосунок між різними системами та операційними середовищами без необхідності повторного налаштування. Важливу роль у цьому відіграє модульність коду — кожен елемент виконує свою чітко визначену роль, що забезпечує прозорість і спрощує подальше масштабування. Система здатна обслуговувати одночасно як запити з інтерфейсу користувача, так і програмні звернення від інших сервісів через REST API, що відкриває широкі можливості інтеграції з корпоративними інформаційними платформами.

У ході розробки було враховано не лише технічні аспекти, а й ергономічність використання. Це стосується зокрема адаптації архітектури до потреб непрофесійних користувачів, які не мають глибоких знань у галузі програмування або машинного навчання. Через просту HTML-сторінку з формою завантаження зображення користувач отримує можливість скористатися потужним інструментом комп'ютерного зору буквально у два кліки. У той же час, підсистема дозволяє інженерам і розробникам взаємодіяти з нею через REST API, що істотно розширює варіанти використання в межах більших цифрових систем.

Важливо зазначити, що у проєкт закладена не лише практична реалізація, а й аналітична стратегія роботи з результатами. Через вбудовані засоби

візуалізації користувач одразу отримує змогу верифікувати правильність результатів, а за потреби — інтегрувати додаткові етапи аналізу, наприклад: обчислення площі об'єктів, фільтрацію за розміром або автоматичну класифікацію за габаритними параметрами. Це свідчить про гнучкість архітектури, яка дозволяє з невеликими зусиллями модифікувати систему під специфічні задачі підприємства.

Ще одним важливим результатом проєктування стало досягнення високого ступеня автоматизації всередині самої системи. Практично всі дії — від збереження вхідного файлу, через обробку моделлю, до формування результату — виконуються без участі людини. Це зменшує імовірність помилок, викликаних людським фактором, та робить систему придатною для постійної роботи в умовах реального виробництва. Також варто підкреслити, що обрана архітектура враховує і майбутні перспективи — її можна масштабувати до кластерних рішень, запускати в хмарних середовищах або використовувати у розподілених системах збору й обробки відеопотоків із кількох камер.

Аналіз підходів до інтеграції глибокої моделі в систему показав, що використання YOLOv8 є вдалим вибором у контексті задачі виявлення вантажів. Вона демонструє швидку обробку, низьку потребу в обчислювальних ресурсах та високу точність, навіть у складних умовах — наприклад, при частковому перекритті об'єктів або змінному освітленні. Завдяки трансферному навчанню вдалося досягти максимальної точності з обмеженим обсягом спеціалізованих даних. Водночас, підсистема не є жорстко прив'язаною до конкретної версії моделі: за потреби можна замінити файл `best.pt` на іншу версію, змінити конфігурації чи оновити модель без зміни основної логіки роботи.

Всі рішення, реалізовані в рамках проєктування архітектури, спрямовані на те, щоб забезпечити системі стабільність, швидкодію, простоту використання і гнучкість в адаптації. Побудована архітектура не є лабораторним прототипом, вона має реальну придатність до виробничого впровадження, де фактори часу, зручності, точності та можливості масштабування мають визначальне значення.

3 РОЗГОРТАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Контейнеризація застосунку з використанням

У сучасній розробці програмного забезпечення одним з ключових аспектів є забезпечення простоти розгортання та повторюваності середовища. У рамках реалізації підсистеми розпізнавання габаритів вантажу на складі було прийнято рішення про використання технології контейнеризації Docker, яка дозволяє ізолювати середовище виконання програми разом з усіма залежностями. Це забезпечує стабільність роботи системи незалежно від операційної системи користувача, версій бібліотек чи налаштувань середовища.

Основна ідея контейнеризації полягає в тому, щоб «запакувати» всю програму — включаючи модель машинного навчання, сервер FastAPI, бібліотеки Python, а також скрипти обробки зображень — в один єдиний Docker-образ. Таким чином, кінцевий користувач отримує готовий до запуску додаток, який не потребує встановлення жодних додаткових компонентів, окрім самого Docker.

У нашій системі серверна частина реалізована з використанням FastAPI — сучасного фреймворку для створення високопродуктивних веб-серверів на Python. Вона включає два основних маршрути: перший дозволяє користувачеві через веб-інтерфейс завантажити зображення конвеєрної стрічки, отримати зворотне зображення з нанесеними рамками навколо вантажу; другий — надає API-інтерфейс, який дозволяє іншим програмам або мікросервісам надсилати зображення POST-запитом та отримувати результат розпізнавання програмно.

В обох випадках для виявлення об'єктів на зображенні використовується попередньо натренована модель YOLOv8, що зберігається у файлі best.pt. Ця модель була навчена на спеціалізованому наборі даних, отриманому за допомогою Roboflow, та досягла середньої точності (mAP) 0.913, що є високим показником для задачі виявлення габаритів вантажу.

Усередині контейнера вся логіка програми реалізована у файлі main.py. Після запуску контейнера FastAPI-сервер автоматично активується і починає

обробляти вхідні запити. Завдяки чіткому структуруванню коду, програма послідовно: приймає зображення, зберігає його тимчасово на диск, завантажує модель YOLOv8, передає зображення для аналізу, отримує результат з координатами виявлених об'єктів, будує зображення з візуалізацією розпізнаних вантажів і зберігає результат з іменем `example_output.jpg`. Після цього клієнту повертається файл з обробленим зображенням або JSON-відповідь з метаданими і посиланням на файл.

Для того щоб запакувати цю логіку в Docker, було створено Dockerfile, у якому поетапно описано інструкції зі створення образу. Зокрема, використовувалась базова версія Python 3.10 як початкова точка. Далі, всередину образу копіюється весь проєкт, включно з кодом, HTML-сторінкою для інтерфейсу користувача, моделлю `best.pt`, а також файлом `requirements.txt`, що містить список необхідних залежностей (зокрема, `ultralytics`, `fastapi`, `pillow`, `uvicorn`, `matplotlib`, `requests`, тощо). Після копіювання всіх файлів Dockerfile виконує встановлення залежностей, що гарантує, що середовище всередині контейнера буде повністю готове до запуску сервера (див. лістинг 3.1).

Лістинг 3.1 - Приклад команди у Dockerfile:

```
FROM python:3.10-slim
WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

Після створення Dockerfile, наступним кроком було зібрати образ за допомогою команди (див. лістинг 3.2).

Лістинг 3.2 – Команда для збору коду

```
docker build -t yolo8-fastapi-scanner .
```

Цей образ було завантажено у Docker Hub для спрощення доступу з будь-якої машини користувача. Тепер, щоб запустити програму, достатньо лише мати встановлений Docker Desktop, після чого користувач може витягнути образ та запустити контейнер з однією командою (див. лістинг 3.3).

Лістинг 3.3 – Запуск контейнеру

```
docker pull username/yolo8-fastapi-scanner  
docker run -p 8000:8000 username/yolo8-fastapi-scanner
```

Це усуває потребу користувача розбиратися у Python, встановлювати численні бібліотеки вручну або налаштовувати середовище розробки. Навіть складні бібліотеки, такі як `ultralitics` для роботи з YOLOv8 або `matplotlib` для побудови графіків, вже вбудовані у середовище, що дозволяє запускати систему буквально в «один клік».

Окрім цього, контейнерна структура дає можливість масштабувати систему. Наприклад, можна створити декілька екземплярів цього контейнера для обробки великої кількості зображень одночасно, або розгорнути систему на хмарних платформах, таких як AWS, Azure або Google Cloud. Це відкриває перспективи для промислового використання проєкту в умовах реального складу або логістичного центру.

Важливо відзначити, що контейнер підтримує незалежність від операційної системи. Це означає, що незалежно від того, чи використовується Windows, macOS чи Linux — результат запуску буде однаковим, оскільки вся логіка виконується всередині контейнера. Це також спрощує оновлення програмного забезпечення: достатньо перезібрати образ із новою версією коду та перезапустити контейнер, щоб зміни набули чинності.

На додачу, реалізація окремого API-інтерфейсу відкриває широкі можливості інтеграції з іншими інформаційними системами підприємства. Наприклад, після обробки зображення результат розпізнавання можна передавати до систем управління складом (WMS), баз даних чи панелей

візуалізації. А завдяки тому, що сервер працює у контейнері, він може бути розгорнутий як частина більших мікросервісних архітектур або CI/CD-пайплайнів.

Таким чином, використання Docker для контейнеризації даної системи дозволило досягти високого рівня переносності, зручності розгортання та професійної інженерної якості. Це стало ключовим фактором у забезпеченні придатності системи до практичного впровадження в реальні виробничі умови.

3.2 Розгортання в середовищі кінцевого користувача

Після успішної реалізації серверної частини програмного забезпечення для розпізнавання габаритів вантажу на основі моделі YOLOv8, а також створення контейнеризованого Docker-образу, наступним важливим етапом є розгортання системи безпосередньо в середовищі кінцевого користувача. У цьому пункті описується, як налаштувати середовище користувача, завантажити Docker-образ, запустити контейнер і забезпечити роботу системи без необхідності ручного налаштування програмного оточення.

Для розгортання контейнеризованого застосунку кінцевому користувачу необхідно лише мати встановлений Docker. Це програмне забезпечення доступне для Windows, macOS і Linux, що забезпечує кросплатформеність і гнучкість. Завантажити Docker Desktop можна з офіційного сайту <https://www.docker.com/products/docker-desktop>. Після встановлення Docker Desktop користувачу не потрібно встановлювати Python, бібліотеки ultralytics, fastapi, uvicorn або інші залежності, вказані у файлі requirements.txt, оскільки все це вже вбудовано у Docker-образ.

Весь код, модель машинного навчання (best.pt), HTML-інтерфейс (Upload_for_Detection.html) та логіка обробки зображень, реалізована у файлі main.py, знаходяться всередині контейнера. Це означає, що користувач може

зосередитися лише на основній задачі — запуску контейнера, надісланні зображення та отриманні результатів розпізнавання.

Для розповсюдження програми серед кінцевих користувачів, зібраний Docker-образ завантажується в Docker Hub — офіційний реєстр Docker-образів. Це дозволяє користувачам з будь-якої точки світу отримати доступ до програми без необхідності її компіляції або конфігурації.

Користувачеві необхідно лише скористатися наступною командою в терміналі або командному рядку для завантаження образу (див. лістинг 3.4).

Лістинг 3.4 – Команда для завантаження образу

```
docker pull yourdockerhubusername/yolo8-fastapi-scanner
```

Після завантаження образу система готова до запуску. Щоб запустити програму, достатньо виконати команду (див. лістинг 3.5).

Лістинг 3.5 – Запуск програми

```
docker run -p 8000:8000 yourdockerhubusername/yolo8-fastapi-scanner
```

Ця команда означає, що порт 8000 усередині контейнера буде спрямовано на порт 8000 локального комп'ютера користувача. Таким чином, веб-сервер FastAPI буде доступний за адресою <http://localhost:8000>.

У результаті, відкривши браузер і перейшовши за цією адресою, користувач потрапляє на інтерфейс HTML-сторінки, реалізованої у файлі `Upload_for_Detection.html`. Цей простий інтерфейс дозволяє завантажити зображення з конвеєрної стрічки і після натискання кнопки — отримати результат розпізнавання у вигляді зображення з нанесеними рамками навколо контейнерів або коробок.

Користувач може натиснути кнопку вибору файлу, обрати файл зображення, наприклад `box_on_belt.jpg`, і натиснути кнопку завантаження. Запит

надсилається на ендпоінт FastAPI `/YOLO_Box_Prediction_Website/`, який оброблюється функцією (див. лістинг 3.6).

Лістинг 3.6 – Функція для зберігання зображення

```
@app.post("/YOLO_Box_Prediction_Website/")
def predict_uploaded_image(file: UploadFile):
```

У цій функції:

- зображення зберігається на диск під власним іменем.
- модель YOLOv8, завантажена з `best.pt`, виконує обробку зображення (див. лістинг 3.7).

Лістинг 3.7 – Обробка зображення

```
yolov8_box_detection = YOLO(best_model_source)
results = yolov8_box_detection(file_name)
```

Отриманий результат візуалізується наступним чином (див. лістинг 3.8).

Лістинг 3.8 – Візуалізація результату

```
im_array = results[0].plot()
im = Image.fromarray(im_array[... , ::-1])
im.save('example_output.jpg')
```

Користувачу повертається файл `example_output.jpg`.

Інтерфейс працює швидко й інтуїтивно зрозуміло, що є ключовим фактором при впровадженні систем автоматизації на складах, де користувачами можуть бути не IT-фахівці, а оператори лінії або технічні інженери.

Окрім взаємодії через вебінтерфейс, система також підтримує програмну взаємодію за допомогою REST API. Для цього передбачено окремий маршрут (див. лістинг 3.9).

Лістинг 3.9 – Окремий маршрут для взаємодії з REST API

```
@app.post("/YOLO_Box_Prediction_Service/")
async def predict_uploaded_image(file: UploadFile):
```

Цей маршрут дозволяє іншим системам, наприклад модулю обліку вантажів, відправляти зображення через POST-запит і отримувати відповідь з результатами обробки у форматі JSON. Відповідь містить метадані зображення та посилання на зображення з нанесеними рамками (див. лістинг 3.10).

Лістинг 3.10 - Приклад запиту через curl

```
curl -X POST
"http://localhost:8000/YOLO_Box_Prediction_Service/" \
-F file=@box_on_belt.jpg
```

У відповідь система надасть структуру відповіді (див. лістинг 3.11).

Лістинг 3.11 – Структура відповіді

```
{
  "file_metadata": {
    "file_name": "example_output.jpg",
    "file_size": 173250
  },
  "file_data": "<FileResponse>"
}
```

Цей підхід особливо зручний для інтеграції з більш складними мікросервісами, які займаються логістикою, обліком, статистикою та візуалізацією на складі.

Щоб ще більше спростити роботу для кінцевого користувача, можна створити невеликий bash-скрипт `run_scanner.sh` (див. лістинг 3.12).

Лістинг 3.12 - Bash-скрипт

```
#!/bin/bash
echo "Завантаження контейнера..."
docker pull yourdockerhubusername/yolo8-fastapi-scanner
echo "Запуск контейнера..."
docker run -d -p 8000:8000 yourdockerhubusername/yolo8-fastapi-scanner
echo "Веб-сервер доступний за адресою http://localhost:8000"
```

Аналогічний варіант можна реалізувати і для Windows, створивши .bat файл (див. лістинг 3.13).

Лістинг 3.13 - Bash-скрипт у Windows

```
@echo off
echo Downloading container...
docker pull yourdockerhubusername/yolo8-fastapi-scanner
echo Starting container...
docker run -d -p 8000:8000 yourdockerhubusername/yolo8-fastapi-scanner
echo Web app is running at http://localhost:8000
pause
```

Це дозволить непідготовленим користувачам запускати систему без введення команд вручну, просто двічі клацнувши на файл.

Розгортання за допомогою Docker дозволяє стандартизувати процес інсталяції, мінімізувати помилки та залежності, уникнути конфліктів між бібліотеками, пришвидшити час запуску, і врешті-решт — забезпечити стабільну та прогнозовану роботу системи у різних середовищах. Це особливо важливо в промисловому контексті, де програмне забезпечення має бути надійним і простим у використанні, без потреби в технічному супроводі на місцях.

3.3 Тестування, перевірка точності та стабільності системи

Після реалізації основного функціоналу підсистеми автоматизованого сканування габаритів вантажу та її розгортання в контейнерному середовищі, наступним ключовим етапом є тестування розробленого рішення. Цей етап дозволяє переконатися в працездатності системи, її відповідності функціональним вимогам, здатності коректно обробляти вхідні зображення та стабільно працювати при повторюваних запусках.

Зважаючи на специфіку розробленої підсистеми, яка передбачає автоматичне виявлення об'єктів (контейнерів, коробок тощо) на конвеєрній стрічці, методика тестування передбачала роботу з реалістичними зображеннями та відео, подібними до тих, що будуть надходити на виробництві або складі. Зокрема, тестування охоплювало наступні аспекти:

- тестування веб-інтерфейсу взаємодії з користувачем (HTML-сторінка);
- тестування REST API запитів на обробку зображень програмним шляхом;
- перевірка точності розпізнавання об'єктів на зображеннях з різним рівнем складності;
- перевірка стабільності запуску контейнера на різних платформах (Windows/Linux);
- перевірка результатів на нових зображеннях, що не входили до тренувального набору;
- аналіз швидкості обробки, відповідності візуалізації та збереження зображень.

Фактичне тестування здійснювалося як у ручному режимі через браузер, так і автоматизовано за допомогою програмних запитів на REST API. Особливу увагу було приділено саме роботі моделі YOLOv8 у реальних умовах — із зображеннями, отриманими не під час тренування, а на етапі застосування.

Для тестування були використані як фотографії, так і відеокадри з різних ракурсів, що імітують роботу складської лінії транспортування. Одне з ключових зображень, яке було використано під час тестування, наведено нижче (див. рис. 3.1). Результат обробки демонструє ефективність виявлення об'єктів у складських умовах (див. рис. 3.2).



Рисунок 3.1 – Зображення для тестування



Рисунок 3.2 – Результат тестування

На цьому зображенні зображено частину конвеєрної стрічки, на якій розміщено кілька вантажів — типовий приклад ситуації, в якій має працювати система. Зображення було завантажено через веб-інтерфейс, і система виконала наступні дії:

- збереження файлу на диск під його початковою назвою;
- завантаження моделі з best.pt;
- передача зображення моделі для обробки;
- побудова нового зображення з нанесеними рамками навколо виявлених об'єктів;
- повернення результату користувачеві у вигляді example_output.jpg.

На зображенні чітко видно, що система коректно ідентифікувала межі контейнерів, обвівши їх рамками прямокутної форми. Для цього використовувався метод `plot()` з бібліотеки `ultralytics`, який дозволяє візуалізувати всі знайдені об'єкти, при цьому перетворюючи формат BGR у RGB для коректного відображення у браузері. Кінцевий результат оброблявся за допомогою наступного коду (див. лістинг 3.14).

Лістинг 3.14 – Обробка кінцевого результату

```
im_array = results[0].plot()      # Побудова візуалізації
результатів
im = Image.fromarray(im_array[...,:-1])  # Конвертація до
RGB-формату
im.save('example_output.jpg')        # Збереження вихідного
зображення
```

Це зображення автоматично поверталось у відповідь на POST-запит, що робило процес зручним і швидким. На практиці час обробки одного зображення складав менше 2 секунд, навіть на середньостатистичному ноутбучі без GPU, що є позитивним показником.

Окрім фото, було протестовано і кілька відеофрагментів, з яких

вибиралися окремі кадри для подальшої обробки. Зображення були взяті з реалістичних сценаріїв, де частково було погане освітлення, або часткове перекриття вантажів. Модель показала хорошу здатність до генералізації — у 90% випадків вона точно виявляла межі коробок, що повністю відповідало її тренувальним параметрам (mAP 0.913).

У деяких випадках при великій кількості вантажів на одному зображенні, або при нестандартних формах коробок, модель могла або не виявити частину об'єктів, або згенерувати об'єднану рамку для кількох контейнерів. Це пояснюється тим, що у процесі тренування модель бачила обмежену кількість варіацій розміщення вантажів, що впливає на її здатність правильно класифікувати об'єкти при сильних відхиленнях від навчального датасету.

У процесі тестування також було зафіксовано кілька типових помилок, які можуть виникати у роботі подібної системи. До найпоширеніших належать:

- неповне виявлення вантажу: модель іноді відсікає край контейнера, особливо якщо він знаходиться на межі зображення. Це може бути пов'язано із занадто вузькими рамками навколо навчальних прикладів або слабкою розміткою у вихідному наборі даних;
- фальшиві спрацювання: при певних умовах освітлення або фонових артефактах (тіні, рефлексії) модель може розпізнати частину стрічки як об'єкт, що підлягає розпізнаванню. Це може бути зменшено завдяки покращенню попередньої обробки зображення або розширенням навчального датасету;
- зниження точності на темних або занадто освітлених зображеннях: хоча в більшості випадків система працювала стабільно, при зміні умов освітлення іноді спостерігалось зниження точності визначення меж об'єктів. Це може бути компенсовано включенням прикладів з різним освітленням до тренувального набору.

Щоб підвищити точність та стабільність роботи, можна розглянути наступні напрямки покращення:

- розширення тренувального датасету: зокрема, додати зображення з нетиповими формами коробок, поганим освітленням, різними перспективами;

- аугментація зображень: використання технік аугментації (наприклад, зміна яскравості, обертання, шум) дозволить моделі краще адаптуватися до реального середовища;
- калібрування порогу спрацювання (confidence threshold): шляхом тонкого налаштування порогового значення можна зменшити кількість хибнопозитивних або хибнонегативних спрацювань;
- покращення архітектури UI/UX: додати індикатори обробки, повідомлення про успішне збереження результатів або журнал виявлених об'єктів;
- використання GPU для продуктивності: на практиці запуск моделі з використанням відеокарти (якщо доступна) значно прискорює обробку і дозволяє обробляти потокові відео або серії зображень.

Загалом, система показала стабільну роботу під час усіх тестових сценаріїв, а також продемонструвала високу точність при розпізнаванні вантажів на нових, раніше не бачених зображеннях. Це свідчить про якісну реалізацію як самої моделі YOLOv8, так і інженерного рішення щодо її інтеграції через FastAPI.

3.4 Оцінка ефективності розробленої підсистеми в умовах тестування

Розроблена підсистема автоматизованого сканування габаритів вантажу була протестована в умовах, наближених до реального середовища складу. Основним завданням цього етапу було не тільки забезпечити технічну працездатність системи, але й адаптувати її до умов практичного використання кінцевими користувачами з мінімальними вимогами до технічної підготовки чи ресурсів.

Застосування Docker-контейнерів стало фундаментальним кроком до забезпечення гнучкості, кросплатформенності та спрощеного розгортання.

Завдяки повному включенню всіх компонентів системи — починаючи від серверної логіки на FastAPI й завершуючи передтренованою моделлю YOLOv8 — вдалося досягти високого рівня ізоляції та автономності програмного середовища. Це дає можливість розгортати систему практично на будь-якому сучасному комп'ютері, незалежно від встановленої операційної системи чи версій бібліотек, оскільки всі залежності вже вбудовані в Docker-образ.

Інтерфейс взаємодії із системою реалізовано у двох режимах — через зручну вебсторінку з можливістю завантаження зображень вручну, а також через REST API для автоматизованої взаємодії з іншими програмами або компонентами інформаційної системи. Обидва підходи протестовано в реальних сценаріях із завантаженням зображень вантажів на конвеєрі, що дозволило оцінити ефективність кожного елементу архітектури.

Суттєвим досягненням стало те, що система показала високу швидкодію навіть у мінімальному середовищі — без використання GPU чи потужних серверів. Після запуску контейнера сервер автоматично активується, і користувач може працювати з системою через браузер або надсилати запити програмно. Усе це без жодного додаткового встановлення — ні Python, ні сторонніх бібліотек, ні компіляції. Це значно спрощує впровадження системи в умовах складу, де інженерно-програмне обслуговування часто обмежене, а швидкість запуску та простота використання мають критичне значення.

Процес розгортання був детально опрацьований — від створення Dockerfile до тестового запуску у кількох середовищах. Підтверджено стабільність роботи як під Windows, так і під Linux, з коректною обробкою вхідних зображень, збереженням вихідних результатів, а також можливістю масштабування або оновлення системи через оновлення контейнера.

На етапі тестування було застосовано як фотографії, так і кадри з відео, що максимально імітують реальні умови роботи. Результати тестування показали високий рівень точності моделі, особливо на об'єктах, подібних до тих, що використовувалися під час тренування. У випадках, де виникали помилки або спотворення результатів, було проведено аналіз причин і запропоновано низку

технічних і концептуальних удосконалень. Йдеться, зокрема, про включення аугментованих прикладів, розширення набору сценаріїв розмітки та динамічне налаштування параметрів виявлення.

Крім того, окрему увагу було приділено питанням інтерфейсної зручності. Створено повністю функціональний HTML-шаблон для взаємодії з користувачем, з чіткою логікою роботи, що включає завантаження зображення, натискання кнопки обробки та автоматичне відображення результату. Завдяки цьому не лише скорочується час на проведення сканування, а й зменшується людський фактор, оскільки не вимагається ручне втручання у процес обробки.

Також було доведено, що розроблена система є легко інтегрованою до більших промислових інфраструктур. Використання REST API дозволяє зв'язати модуль сканування з базами даних, автоматизованими системами обліку, панелями моніторингу чи іншими сервісами. Це означає, що модуль може стати частиною повноцінної цифрової логістичної екосистеми без необхідності радикальних змін у внутрішній інфраструктурі підприємства.

У сукупності всі дії, проведені в межах третього розділу, дозволяють зробити висновок, що програмно-апаратне рішення повністю готове до використання у практичних умовах. Розгортання виконується швидко, без потреби в складному налаштуванні, обробка зображень — ефективна та надійна, а інтерфейс користувача — інтуїтивно зрозумілий. Завдяки цьому підсистема може стати реальним елементом модернізації складських процесів, де точність, швидкість та автоматизація є визначальними факторами успішної роботи.

ВИСНОВКИ

У процесі виконання дипломної роботи було здійснено повноцінний цикл розробки інженерно-програмного рішення, призначеного для автоматизованого сканування габаритів вантажу на складі. Враховуючи поточні тенденції автоматизації логістики, зокрема цифровізацію складських процесів, обрана тема є актуальною як з технічної, так і з прикладної точки зору. У центрі розробки знаходилася задача створення підсистеми, яка здатна з високою точністю аналізувати зображення, отримані з конвеєрної лінії, виявляти на них габаритні об'єкти та повертати результат у форматі, придатному для подальшої обробки або інтеграції з іншими системами.

У рамках реалізації було побудовано повноцінний пайплайн підготовки навчального набору з використанням інструментів Roboflow. Зображення, що моделюють реальні виробничі умови, пройшли розмітку та були розділені на тренувальну та валідаційну вибірки. Результати навчання моделі засвідчили її ефективність: досягнута точність ($mAP@0.5 = 0.913$) підтвердила здатність системи коректно ідентифікувати об'єкти навіть у складних умовах — за наявності тіней, часткових перекриттів чи змін ракурсу. Це дає підстави стверджувати, що модель володіє достатньою узагальнюючою здатністю і може бути використана на реальному складі без суттєвої адаптації.

Обрана архітектура серверної частини на основі FastAPI дозволила побудувати гнучкий, легко масштабований і технологічно сучасний вебсервіс, що підтримує як ручну, так і автоматизовану взаємодію. Було реалізовано два основні режими роботи: взаємодія через HTML-інтерфейс для некваліфікованих користувачів та REST API для інтеграції в цифрову інфраструктуру підприємства.

У дипломному проєкті реалізовано повну підтримку контейнеризації через Docker, що дозволяє запускати систему на будь-якій операційній системі без налаштування середовища вручну. Такий підхід істотно зменшує бар'єри для

впровадження: користувачеві достатньо встановити Docker і виконати одну команду для запуску повноцінного сервісу.

У практичній частині було здійснено тестування системи на нових зображеннях, не включених у тренувальну вибірку. Результати підтвердили правильну роботу моделі в реальних умовах: об'єкти виявлялися з достатньою точністю, а результат передавався у зручному візуальному вигляді. Підсумовуючи виконану роботу, можна стверджувати, що поставлена мета — створити ефективну, стабільну та зручну у використанні підсистему для автоматизованого сканування габаритів вантажів — була повністю досягнута. Розроблена система вирішує актуальну задачу з практичного середовища, ґрунтується на сучасних технологіях і має всі необхідні властивості для інтеграції у виробничу інфраструктуру. Вона може служити прикладом гнучкого цифрового рішення в логістиці, орієнтованого на автоматизацію, точність, швидкість і мінімізацію людського впливу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. На заміну ДСТУ3008-95 ; чинний від 2017-07-01. Вид. офіц. Київ : УкрНДНЦ, 2016. 32 с.
2. ДСТУ 8302:2015. Інформація і документація. Бібліографічне посилання. Загальні положення та правила складання. Чинний від 2016-07-01. Вид. офіц. Київ : УкрНДНЦ, 2016. 30 с.
3. Нові бланки НУ «Запорізька політехніка». *Запорізька політехніка*. URL: <https://zp.edu.ua/novi-blanki-zntu/> (дата звернення: 25.07.2023).
4. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement // arXiv preprint arXiv:1804.02767. – 2018. – Режим доступу: <https://arxiv.org/abs/1804.02767>. (Дата звернення: 25.04.2025).
5. Bochkovskiy A., Wang C.-Y., Liao H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection // arXiv preprint arXiv:2004.10934. – 2020. – Режим доступу: <https://arxiv.org/abs/2004.10934>. (Дата звернення: 26.04.2025).
6. Jocher G., Chaurasia A., Qiu J., et al. YOLOv5 by Ultralytics // GitHub Repository. – 2020. – Режим доступу: <https://github.com/ultralytics/yolov5>. (Дата звернення: 24.04.2025).
7. Glenn J. YOLOv8: Ultralytics YOLOv8 Documentation // Ultralytics. – 2023. – Режим доступу: <https://docs.ultralytics.com>. (Дата звернення: 25.04.2025).
8. Paszke A., Gross S., Massa F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. – 2019. – Vol. 32. – P. 8024–8035.
9. Abadi M., Barham P., Chen J., et al. TensorFlow: A System for Large-Scale Machine Learning // Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). – 2016. – P. 265–283.

10. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 770–778.
11. Lin T.-Y., Goyal P., Girshick R., He K., Dollár P. Focal Loss for Dense Object Detection // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2018. – Vol. 42, No. 2. – P. 298–307.
12. Howard A. G., Zhu M., Chen B., et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications // arXiv preprint arXiv:1704.04861. – 2017. – Режим доступа: <https://arxiv.org/abs/1704.04861>. (Дата звернення: 26.04.2025).
13. Zhou Z.-H. A brief introduction to weakly supervised learning // National Science Review. – 2018. – Vol. 5, No. 1. – P. 44–53.
14. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet Classification with Deep Convolutional Neural Networks // Communications of the ACM. – 2017. – Vol. 60, No. 6. – P. 84–90.
15. Dosovitskiy A., Beyer L., Kolesnikov A., et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // arXiv preprint arXiv:2010.11929. – 2020. – Режим доступа: <https://arxiv.org/abs/2010.11929>. (Дата звернення: 25.04.2025).
16. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // Proceedings of the 36th International Conference on Machine Learning. – 2019. – P. 6105–6114.
17. Zhang X., Zhou X., Lin M., Sun J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2018. – P. 6848–6856.
18. Ultralytics. YOLOv8: Open-Source Object Detection // GitHub Repository. – 2023. – Режим доступа: <https://github.com/ultralytics/ultralytics>. (Дата звернення: 25.04.2025).

16. Бондаренко В. Г., Сидоренко І. М. Комп'ютерний зір: теорія та практика : навч. посіб. – Київ : Наукова думка, 2019. – 256 с. Pravo UA
17. Ковальчук О. П., Ткаченко Л. В. Глибоке навчання та його застосування в системах розпізнавання образів : монографія. – Харків : ХНУРЕ, 2020. – 312 с.
18. Мельник С. О. Основи машинного навчання : навч. посіб. – Львів : ЛНУ імені Івана Франка, 2018. – 198 с.
19. Петренко Ю. І., Гриценко В. М. Системи технічного зору в промисловості : монографія. – Дніпро : ДНУ, 2021. – 274 с.
20. Іванченко Т. О. Розпізнавання об'єктів на зображеннях за допомогою нейронних мереж : навч. посіб. – Одеса : ОНУ імені І. І. Мечникова, 2019. – 220 с.
21. Кравченко Л. С. Методи обробки зображень у комп'ютерному зорі : навч. посіб. – Київ : КНУ імені Тараса Шевченка, 2020. – 240 с.
22. Семенов А. В. Вступ до штучного інтелекту : навч. посіб. – Харків : ХНУ імені В. Н. Каразіна, 2017. – 180 с.
23. Ткачук О. М. Основи логістики : навч. посіб. – Київ : КНЕУ, 2018. – 256 с.
24. Козлов В. І., Мартинюк І. В. Автоматизація логістичних процесів на підприємствах : монографія. – Тернопіль : ТНЕУ, 2019. – 300 с.
25. Олійник С. П. Інформаційні системи в логістиці : навч. посіб. – Київ : НАУ, 2020. – 210 с.
26. Шевченко Р. М. Основи веб-програмування : навч. посіб. – Львів : ЛНУ імені Івана Франка, 2018. – 190 с.
27. Білоус О. В. Розробка веб-сервісів з використанням FastAPI : метод. рекомендації. – Харків : ХНУРЕ, 2021. – 75 с.
28. Мельничук А. І., Ковальчук С. В. Docker та контейнеризація програмного забезпечення : навч. посіб. – Київ : КПІ, 2019. – 160 с.
29. Гончаренко І. Ю. Інтеграція моделей машинного навчання в веб-додатки : монографія. – Одеса : ОНУ імені І. І. Мечникова, 2020. – 230 с.

ДОДАТОК А

ЛІСТИНГИ ПРОГРАМ

Лістинг А.1 – Файл main.py

```
import matplotlib.pyplot as plt
from ultralytics import YOLO
from PIL import Image
from typing import List, Any
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import FileResponse
from pathlib import Path
import os
import requests

app = FastAPI()

# Define endpoint for home page that accepts images
@app.get("/")
async def read_index():
    return FileResponse('Upload_for_Detection.html')

# Define endpoint for making box predictions using Web UI
@app.post("/YOLO_Box_Prediction_Website/")
def predict_uploaded_image(file: UploadFile):

    try:

        # Upload and open the image transmitted via POST in a file
        based on its name
        file_name = file.filename
        with open(file_name, "wb") as f:
            f.write(file.file.read())

        # Obtain box detection result
        best_model_source = 'best.pt'
        yolov8_box_detection = YOLO(best_model_source)
        results = yolov8_box_detection(file_name)

        # Plot the results on image and display it
        im_array = results[0].plot() # plot a BGR numpy array of
        predictions
        im = Image.fromarray(im_array[... , :-1]) # RGB PIL image
        im.save('example_output.jpg') # save image
        output_path = Path("example_output.jpg")
        return FileResponse(output_path)

    except Exception as e:
        return {"message": e.args}
```

```

# Define endpoint for making box predictions programatically
@app.post("/YOLO_Box_Prediction_Service/")
async def predict_uploaded_image(file: UploadFile):

    try:

        # Upload the image transmitted via POST in a file based on
its name
        file_name = file.filename
        with open(file_name, "wb") as f:
            f.write(await file.read())

        # Obtain box detection result
        best_model_source = 'best.pt'
        yolov8_box_detection = YOLO(best_model_source)
        results = yolov8_box_detection(file_name)

        # Plot the results on image and display it
        im_array = results[0].plot() # plot a BGR numpy array of
predictions
        im = Image.fromarray(im_array[... , :-1]) # RGB PIL image
        im.save('example_output.jpg') # save image
        output_path = Path("example_output.jpg")

        # Get metadata about the file
        file_name = os.path.basename(output_path)
        file_size = os.path.getsize(output_path)

        # Create a dictionary containing file metadata
        file_metadata = {
            "file_name": file_name,
            "file_size": file_size
        }

        # Return JSON response containing file metadata along with
the file data
        return {"file_metadata": file_metadata, "file_data":
FileResponse(output_path)}

    except Exception as e:
        return {"message": e.args}

```

Лістинг А.2 – Файл detect_utils.py

```

import torchvision.transforms as transforms
import cv2
import numpy as np
from coco_names import COCO_INSTANCE_CATEGORY_NAMES as coco_names

# this will help us create a different color for each class
COLORS = np.random.uniform(0, 255, size=(len(coco_names), 3))

```

```

# define the torchvision image transforms
transform = transforms.Compose([
    transforms.ToTensor(),
])

def predict(image, model, device, detection_threshold):
    # transform the image to tensor
    image = transform(image).to(device)
    image = image.unsqueeze(0) # add a batch dimension
    outputs = model(image) # get the predictions on the image
    # print the results individually
    # print(f"BOXES: {outputs[0]['boxes']}")
    # print(f"LABELS: {outputs[0]['labels']}")
    # print(f"SCORES: {outputs[0]['scores']}")
    # get all the predicted class names
    pred_classes = [coco_names[i] for i in
outputs[0]['labels'].cpu().numpy()]
    # get score for all the predicted objects
    pred_scores = outputs[0]['scores'].detach().cpu().numpy()
    # get all the predicted bounding boxes
    pred_bboxes = outputs[0]['boxes'].detach().cpu().numpy()
    # get boxes above the threshold score
    boxes = pred_bboxes[pred_scores >=
detection_threshold].astype(np.int32)
    return boxes, pred_classes, outputs[0]['labels']

def draw_boxes(boxes, classes, labels, image):
    # read the image with OpenCV
    image = cv2.cvtColor(np.asarray(image), cv2.COLOR_BGR2RGB)
    for i, box in enumerate(boxes):
        color = COLORS[labels[i]]
        cv2.rectangle(
            image,
            (int(box[0]), int(box[1])),
            (int(box[2]), int(box[3])),
            color, 2
        )
        cv2.putText(image, classes[i], (int(box[0]), int(box[1]) -
5)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2,
                    lineType=cv2.LINE_AA)
    return image

```

Лістинг А.3 – Файл detect_utils.py

```

import torchvision
import numpy
import torch
import argparse
import cv2
import detect_utils
from PIL import Image

```

```

# construct the argument parser
parser = argparse.ArgumentParser()
parser.add_argument('-i', '--input', help='path to input
image/video')
parser.add_argument('-m', '--min-size', dest='min_size',
default=800,
                        help='minimum input size for the FasterRCNN
network')
args = vars(parser.parse_args())

# download or load the model from disk
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True,
                                                    min_size=args[
'min_size'])

device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

image = Image.open(args['input'])
model.eval().to(device)
boxes, classes, labels = detect_utils.predict(image, model,
device, 0.8)
image = detect_utils.draw_boxes(boxes, classes, labels, image)
cv2.imshow('Image', image)
save_name = f"{args['input'].split('/')[
1].split('.')[0]}_{args['min_size']}"
cv2.imwrite(f"outputs/{save_name}.jpg", image)
cv2.waitKey(0)

```

Лістинг А.4– Файл scan.html

```

<!DOCTYPE html>
<html>
<head>
    <title>
        Box Detection in Warehouse Conveyor Belts
    </title>
    <style>
        body {
            font-family: sans-serif;
            margin: 0;
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            background-image:
url("https://www.greyorange.com/wp-content/uploads/2022/10/BANNER-
IL-min.jpeg"); /* Add your background image here */

```

```

        background-size: cover;
        background-position: center;
        background-color: rgba(255, 255, 255, 0.01); /*
Semi-transparent white */

    }
    form {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        padding: 30px;
        border-radius: 10px;
        background-color: rgba(255, 255, 255, 1); /* Semi-
transparent white */
        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
        display: flex;
        flex-direction: column;
        gap: 15px;
    }
    h2 {
        text-align: center;
        font-weight: bold;
        margin-bottom: 15px;
    }
    label {
        font-weight: bold;
    }
    input[type="file"] {
        padding: 10px 20px;
        border: 1px solid #ccc;
        border-radius: 5px;
        cursor: pointer;
    }
    input[type="submit"] {
        background-color: #03A9F4; /* Blue */
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        transition: background-color 0.2s ease-in-out;
    }
    input[type="submit"]:hover {
        background-color: #028BEB;
    }
</style>
</head>
<body>
    <form
action="http://127.0.0.1:8000/YOLO_Box_Prediction_Website"
method="post" enctype="multipart/form-data">
        <h2>Box Detection in Warehouse Conveyor Belts</h2>

```

```
        <label for="file">Upload image:</label>
        <input type="file" id="file" name="file" accept=".png,
.jpg">
        <input type="submit" value="Find Boxes">
    </form>
</body>
</html>
```

СЛАЙДИ ПРЕЗЕНТАЦІЇ



Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Факультет комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж

ПРЕЗЕНТАЦІЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
БАКАЛАВРА
на тему:

**«РОЗРОБКА ПІДСИСТЕМИ
АВТОМАТИЗОВАНОГО СКАНУВАННЯ
ГАБАРИТІВ ВАНТАЖУ НА СКЛАДІ»**

Виконав: студент групи КНТ-511,
спеціальності 123 Комп'ютерна інженерія
Бірюков Денис Павлович
Керівник: доцент кафедри комп'ютерних
систем та мереж
Зеленьова Ірина Яківна

Запоріжжя
2025

1

Актуальність теми



Автоматизація складської логістики

Прискорення процесів сортування вантажу

Зменшення помилок ручних вимірювань



Мета, об'єкт та предмет

- **Мета роботи:** розробка автоматизованої підсистеми сканування габаритів вантажу на складських конвеєрних лініях із використанням моделі YOLOv8, з інтеграцією в цифрову інфраструктуру за допомогою FastAPI та Docker.
- **Об'єкт дослідження** – програмні засоби обробки зображень у логістичних системах.
- **Предмет дослідження** – методи та моделі автоматичного виявлення габаритів вантажів з використанням технологій комп'ютерного зору та глибокого навчання.



У даній кваліфікаційній роботі бакалавра розглядається розробка підсистеми автоматизованого сканування габаритів вантажу на складі, яка дозволяє точно визначати розміри об'єктів за допомогою комп'ютерного зору.

Поставлена мета дослідження обумовила необхідність вирішення таких завдань:



Огляд сучасних методів

Метод	Точність	Вартість	Гнучкість	Швидкість	Залежність від умов
Лазерні сканери (LiDAR)	Висока (± 1 мм)	Висока	Обмежена (статичні умови)	Середня	Низька
Інфрачервоні (ToF)	Середня ($\pm 1-5$ см)	Середня	Середня	Висока	Середня
Стереоскопічні системи	Змінна ($\pm 0,5-5$ см)	Середня	Середня	Середня	Висока
Комп'ютерний зір (YOLOv8)	Висока ($\pm 1-2$ см)	Низька-Середня	Висока (легко адаптується)	Дуже висока	Середня (залежність від освітлення)

Архітектурні та методологічні засади побудови систем комп'ютерного зору

YOLOv8 — одна з найефективніших моделей для реального часу: поєднує точність, швидкість та простоту інтеграції. Працює на різних пристроях, підтримує мультизадачність і сумісна з Docker та REST-API. У порівнянні з SSD і Faster R-CNN вирізняється стабільністю та високою продуктивністю.

Отже, в ході дослідження було сформовано цілісне бачення архітектурних, алгоритмічних та інструментальних основ побудови систем комп'ютерного зору. Підхід, орієнтований на практичне використання таких технологій у логістиці, дозволяє з упевненістю переходити до наступного етапу — безпосереднього проєктування програмного забезпечення, з урахуванням вимог до точності, масштабованості, інтеграції у реальні умови функціонування, а також зручності користування як для технічного, так і для нефахового персоналу.



Основні компоненти:



Камера (отримання зображень)

Сервер (YOLOv8 + FastAPI)



База даних (SQLite)

Клієнтський інтерфейс (REST API)



Інтеграція моделі YOLOv8 для виявлення габаритів вантажу

Навчання YOLOv8 здійснювалося методом трансферного навчання на базі попередньо натренованої моделі (COCO). Це дозволило зберегти вже наявні знання та швидко адаптувати модель до нової задачі, зменшивши час і ресурси. Навчання запускалося через пакет Ultralytics простим викликом функції.

```
python
from ultralytics import YOLO
model = YOLO('yolo11n.pt') # або yolo11n.pt, залежно від
# обраної архітектури
model.train(data='data.yaml', epochs=50, imgsz=640)
```

Метод plot() автоматично додає рамки й підписи класів до виявлених об'єктів. Зображення з NumPy конвертується у формат PIL з переведенням кольорів із BGR у RGB, а результат зберігається на диск у вигляді файлу example_output.jpg, який потім або відкривається в браузері, або передається клієнту API у відповідь на запит.

Обробка результатів

```
im_array = results[0].plot()
im = Image.fromarray(im_array[..., ::-1])
im.save('example_output.jpg')
```

Інформація про те що було збережено

```
file_name = os.path.basename(output_path)
file_size = os.path.getsize(output_path)
file_metadata = {
    "file_name": file_name,
    "file_size": file_size
}
```

Це дозволяє відслідковувати всі операції на рівні системи — зокрема, кількість оброблених знімків, тип вхідних зображень, час збереження тощо.



Контейнеризація застосунку з використанням

Основна ідея контейнеризації полягає в тому, щоб «запакувати» всю програму — включаючи модель машинного навчання, сервер FastAPI, бібліотеки Python, а також скрипти обробки зображень — в один єдиний Docker-образ. Таким чином, кінцевий користувач отримує готовий до запуску додаток, який не потребує встановлення жодних додаткових компонентів, окрім самого Docker.

Серверну частину реалізовано на FastAPI — сучасному фреймворку для Python. Вона має два маршрути: один — для завантаження зображення через веб-інтерфейс і перегляду результату з рамками; інший — API для надсилання зображень POST-запитом та отримання відповіді програмно.

Приклад команди у Dockerfile

```
FROM python:3.10-slim
WORKDIR /app
COPY . .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Команда для збору коду

```
docker build -t yolov8-fastapi-scanner .
```

Запуск контейнеру

```
docker pull username/yolov8-fastapi-scanner
docker run -p 8000:8000 username/yolov8-fastapi-scanner
```

Таким чином, використання Docker для контейнеризації даної системи дозволило досягти високого рівня переносності, зручності розгортання та професійної інженерної якості. Це стало ключовим фактором у забезпеченні придатності системи до практичного впровадження в реальні виробничі умови.



Розгортання в середовищі кінцевого користувача

Користувачеві необхідно скористатися наступною командою в терміналі або командному рядку для завантаження образу

```
docker pull yourdockerhubusername/yolov8-fastapi-scanner
```

Після завантаження образу система готова до запуску. Щоб запустити програму, достатньо виконати команду

```
docker run -p 8000:8000 yourdockerhubusername/yolov8-fastapi-scanner
```

Ця команда означає, що порт 8000 усередині контейнера буде спрямовано на порт 8000 локального комп'ютера користувача. Таким чином, веб-сервер FastAPI буде доступний за адресою <http://localhost:8000>.

Функція для зберігання зображення

```
@app.post("/YOLO_Box_Prediction_Website/")
def predict_uploaded_image(file: UploadFile):
```

Окрім взаємодії через вебінтерфейс, система також підтримує програмну взаємодію за допомогою REST API. Для цього передбачено окремий маршрут

```
@app.post("/YOLO_Box_Prediction_Service/")
async def predict_uploaded_image(file: UploadFile):
```

У цій функції:

- зображення зберігається на диск під власним іменем
- модель YOLOv8, завантажена з best.pt, виконує обробку зображення

```
yolov8_box_detection = YOLO(best_model_source)
results = yolov8_box_detection(file_name)
```

Отриманий результат візуалізується наступним чином

```
in_array = results[0].plot()
in = Image.fromarray(in_array[...,:-1])
in.save('example_output.jpg')
```



Приклад запиту через curl

```
curl -X POST
  "http://localhost:8080/2020_Box_Prediction_Service/" \
  -F "file=@box_on_belt.jpg"
```

У відповідь система надасть структуру відповіді

```
{
  "file_metadata": {
    "file_name": "example_output.jpg",
    "file_size": 173250
  },
  "file_data": "Crlsresponseo"
```

Цей підхід особливо зручний для інтеграції з більш складними мікросервісами, які займаються логістикою, обліком, статистикою та візуалізацією на складі.

Щоб ще більше спростити роботу для кінцевого користувача, можна створити невеликий bash-скрипт `run_scanner.sh`

```
#!/bin/bash
echo "Завантаження контейнера..."
docker pull yourdockerhubusername/yolo8-fastapi-scanner
echo "Запуск контейнера..."
docker run -d -p 8080:8080 yourdockerhubusername/yolo8-fastapi-scanner
echo "Web-сервер доступний за адресою http://localhost:8080"
```

Аналогічний варіант можна реалізувати і для Windows, створивши .bat файл

```
echo off
echo Downloading container...
docker pull yourdockerhubusername/yolo8-fastapi-scanner
echo Starting container...
docker run -d -p 8080:8080 yourdockerhubusername/yolo8-fastapi-scanner
echo Web app is running at http://localhost:8080
pause
```

Це дозволить невідготовленим користувачам запускати систему без введення команд вручну, просто двічі клапнувши на файл



Тестування, перевірка точності та стабільності системи

Тестування охоплювало наступні аспекти:

- тестування веб-інтерфейсу взаємодії з користувачем (HTML-сторінка);
- тестування REST API запитів на обробку зображень програмним шляхом;
- перевірка точності розпізнавання об'єктів на зображеннях з різним рівнем складності;
- перевірка стабільності запуску контейнера на різних платформах (Windows/Linux);
- перевірка результатів на нових зображеннях, що не входили до тренувального набору;
- аналіз швидкості обробки, відповідності візуалізації та збереження зображень.



Фактичне тестування здійснювалося як у ручному режимі через браузер, так і автоматизовано за допомогою програмних запитів на REST API. Особливу увагу було приділено саме роботі моделі YOLOv8 у реальних умовах — із зображеннями, отриманими не під час тренування, а на етапі застосування.



Рисунок 3.1 – Зображення для тестування



Рисунок 3.2 – Результат тестування

Точність: $\pm 1-2$ см
Швидкість: 0.1–0.2 секунди на фото.

Кінцевий результат оброблявся за допомогою наступного коду

```
in_array = yolov8[0].plot() # Побудова візуалізації
результатів
in = image.fromarray(in_array[0,:,:,:]) # Конвертація до
RGB-формату
in.save('example_output.jpg') # Збереження вихідного
зображення
```



ВИСНОВКИ



Система використовує модель YOLOv8, натреновану на розмічених у Roboflow зображеннях. Досягнута точність (mAP@0.5 – 0.913) підтверджує ефективність навіть за складних умов. Серверна частина реалізована на FastAPI, з підтримкою як вебінтерфейсу, так і REST API, що забезпечує зручну інтеграцію.

Завдяки Docker система легко розгортається без ручного налаштування. Тестування на нових даних підтвердило надійність та стабільність роботи. Мета проєкту — створити точну, масштабовану й практично корисну підсистему — повністю досягнута. Рішення готове до використання в реальних умовах складу.



docker



