НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Методи обробки зображень для визначення державних реєстраційних номерів автомобілів»

Виконав:
студент IV курсу, групи КІ-12
Литвин Ярослав Ігорович _____

Керівник:
асистент кафедри ШІ
Кот Анатолій Тарасович

_____

Консультант з економічного розділу:
доцент кафедри економічної кібернетики, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доц.,
Комариста Богдана Миколаївна _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Джигирей І.М.

«15» січня 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту
Литвину Ярославу Ігоровичу

1. Тема роботи «Методи обробки зображень для визначення державних реєстраційних номерів автомобілів», керівник роботи Кот Анатолій Тарасович, затверджені наказом по НН ІПСА від «26» травня 2025 р. №1759-с

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані до роботи: набори символів, зображених на номерних знаках.

4. Зміст роботи: аналіз предметної області, проектування системи, розробка та тестування системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): скріншоти результатів, графіки, діаграми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання: «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури за темою	21.04.2025	Виконано
2.	Підготовка першого розділу	28.04.2025	Виконано
3.	Підготовка другого розділу	05.05.2025	Виконано
4.	Розробка програмного продукту	12.05.2025	Виконано
5.	Підготовка третього розділу	19.05.2025	Виконано
6.	Підготовка економічної частини	26.05.2025	Виконано
7.	Оформлення розділів відповідно до нормоконтролю	02.06.2025	Виконано
8.	Оформлення дипломної роботи	11.06.2025	Виконано
9.	Підготовка презентації доповіді	11.06.2025	Виконано

Студент Ярослав ЛИТВИН

Керівник Анатолій КОТ

РЕФЕРАТ

Дипломна робота: 115 с., 18 рис., 11 табл., 26 посилань, 2 додатки.

ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, КОМП’ЮТЕРНИЙ ЗІР, КЛАСИФІКАЦІЯ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕІЖ, СЕГМЕНТАЦІЯ, НОМЕРНІ ЗНАКИ, YOLO, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ

Об'єктом дослідження є процеси автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів на зображеннях і відео за допомогою систем комп’ютерного зору.

Предметом дослідження є методи комп’ютерного зору та технології машинного навчання, що використовуються для побудови моделей автоматичного розпізнавання номерних знаків на зображеннях та відео.

Мета роботи полягає в розробці ефективної моделі комп’ютерного зору для автоматизованого розпізнавання номерних знаків автомобілів на основі сучасних алгоритмів машинного навчання

Основні результати дослідження: розроблено та навчено моделі машинного навчання, здатні витягувати послідовності символів із зображень номерних знаків; отримані моделі адаптовані для інтеграції у сторонні програмні продукти та подальшого практичного використання.

Напрями подальших досліджень включають покращення якості вхідних наборів даних, розширення можливостей моделі для розпізнавання номерних знаків різних країн, а також розробку прикладного програмного забезпечення для зручної роботи з розробленими моделями.

ABSTRACT

Bachelor's thesis: 115 p., 18 figures, 11 tables, 26 references, 2 appendixes.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, COMPUTER VISION, CLASSIFICATION, NEURAL NETWORKS, CONVULSIVE NEURAL NETWORKS, SEGMENTATION, LICENSE PLATE, YOLO, CONVULSIVE NEURAL NETWORK

The object of the research is the processes of automated recognition of state registration license plates of vehicles in images and videos using a computer vision system.

The subject of the research is computer image methods and machine learning technologies used to build models of automatic recognition of license plates in images and videos.

The purpose of the work is to develop an effective computer image model for automated recognition of car license plates based on machine learning algorithms.

The main results of the research: machine learning models have been developed and trained, capable of extracting a set of characters from the image of license plates; the obtained models have been adapted for integration into third-party software products and further practical use.

Directly subsequent research includes improving the quality of input data sets, expanding the capabilities of models for recognizing license plates of different countries, as well as developing application software for convenient work with the developed models.

ЗМІСТ

ВСТУП

У системах[1] розпізнавання номерних знаків (LPR – License Plate Recognition) зображення транспортних засобів або відеопотоки захоплюються камерами, передаються у віддалений центр обробки та аналізуються з метою виявлення номерних знаків і розпізнавання символів на них. Очікується, що технології LPR відіграватимуть ключову роль у створенні інфраструктури «розумних міст», де мережі камер, встановлених на перехрестях, забезпечуватимуть безперервне відстеження транспортних засобів у реальному часі.

Більшість сучасних LPR – систем покладаються на спеціалізовані системи візуалізації, зокрема інфрачервоні освітлювачі та камери, закріплені на спеціально підготовлених об’єктах інфраструктури. Однак для масштабного розгортання систем моніторингу транспорту необхідним є використання недорогого обладнання, наприклад, IP – камер видимого спектра, інтегрованих у вже наявну міську інфраструктуру.

Істотний прорив у розвитку LPR – систем став можливим завдяки використанню методів глибокого навчання, зокрема згорткових нейронних мереж (Convolutional Neural Networks, CNN). CNN є багатошаровими мережами з прямим поширенням сигналу, де кожен шар здійснює обробку вхідних даних шляхом згортки – повторного застосування одного і того самого набору ваг по всьому зображенню. Це дозволяє ефективно виявляти локальні ознаки на зображеннях без потреби у попередньому ручному формуванні дескрипторів, як це властиво традиційним методам машинного зору.

---

[1]Тут і нижче використано такий інструменти штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

Використання CNN дозволило впроваджувати системи автоматичного розпізнавання номерних знаків навіть на неспеціалізованому обладнанні – відеореєстраторах, камерах відеоспостереження та мобільних пристроях. Така доступність значно розширила можливості застосування LPR у сферах інтелектуального регулювання дорожнього руху, паркувального контролю, міського відеомоніторингу, а також у правоохоронній діяльності.

Перші LPR – системи з’явилися ще у 1976 році в науково-дослідному підрозділі поліції Великої Британії, однак лише поява високоефективних алгоритмів глибинного навчання зробила можливим їхнє масове практичне застосування.

Об’єктом дослідження є процеси автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів на зображеннях і відео за допомогою систем комп’ютерного зору.

Предметом дослідження є методи комп’ютерного зору та технології машинного навчання, що використовуються для побудови моделей автоматичного розпізнавання номерних знаків на зображеннях і відео.

Метою роботи є розробка ефективної моделі комп’ютерного зору для автоматизованого розпізнавання номерних знаків автомобілів на основі сучасних алгоритмів машинного навчання.

Вибір цієї теми зумовлений особистим інтересом до інтелектуальних систем машинного зору та прагненням реалізувати проєкт, що має високу практичну значущість для безпеки дорожнього руху, автоматизації адміністративних процесів та розбудови інфраструктури «розумного міста». Застосування методів глибинного навчання у сфері розпізнавання зображень є актуальним напрямом розвитку прикладної інформатики, що стимулює мій професійний інтерес до даного дослідження.

РОЗДІЛ 1 ОГЛЯД ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

До сьогоднішнього дня більшість розгорнутих систем LPR продовжують використовувати спеціалізоване обладнання, яке суттєво спрощує процес автоматичного розпізнавання номерних знаків на рис. 1.1, Натомість завдання розпізнавання номерних знаків на стандартних RGB – зображеннях, отриманих за допомогою звичайних камер, залишається актуальною та відкритою темою для наукових досліджень.

Встановлення спеціалізованих апаратних комплексів часто є економічно виправданим лише у випадках, коли їх використання безпосередньо пов'язане з отриманням доходу від транспортних потоків, наприклад, через систему збору плати за проїзд або штрафи за порушення правил паркування. У багатьох інших сценаріях впровадження дорогого обладнання є складним для обґрунтування, що стимулює розвиток альтернативних підходів, орієнтованих на використання недорогих засобів візуалізації.

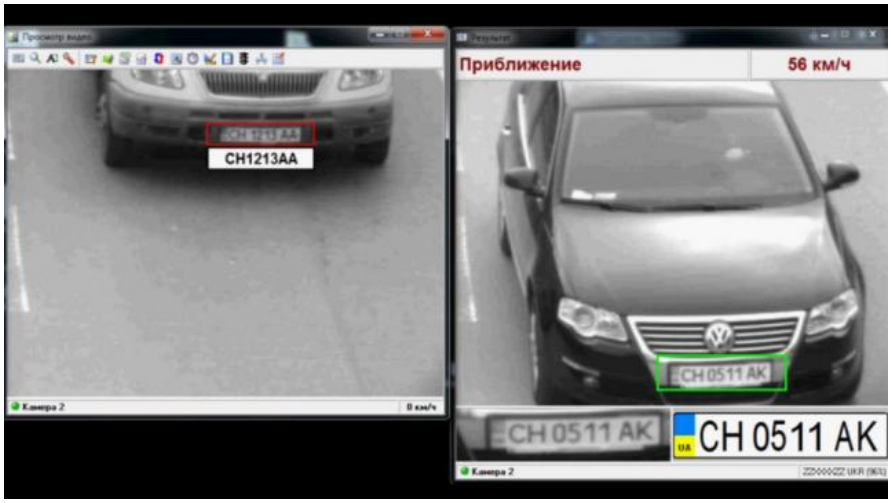


Рисунок 1.1 – Приклад зображення, отриманого за допомогою спеціального обладнання для зчитування номерних знаків

До того, як згорткові нейронні мережі (CNN) та глибинне навчання здійснили справжню революцію у сфері розпізнавання зображень, класичні алгоритми для розпізнавання номерних знаків (LPR, License Plate Recognition) працювали за традиційною трьохетапною схемою обробки зображень. Цей процес складався з таких ключових етапів.

1. Локалізація номерних знаків на зображенні.
2. Сегментація окремих символів на номерному знакові.
3. Розпізнавання кожного сегментованого символу.

На першому етапі, локалізація номерних знаків, основною задачею було виявити область на зображенні, що містить номерний знак. Для цього широко використовували різні методи обробки зображень. Зокрема, бінаризовані зображення оброблялися шляхом виявлення країв із подальшою морфологічною обробкою, що дозволяло ідентифікувати межі об'єктів, зокрема номерних знаків, та фільтрувати небажані контури. Іншим популярним підходом була сегментація через аналіз зв'язаних компонент (Connected Component Analysis, CCA) у поєднанні з просторовими характеристиками, такими як площа, орієнтація та співвідношення сторін виявлених об'єктів.

Коли використовувалися зображення у відтінках сірого, перевага надавалася методам, що підсилювали контраст між номерними знаками й фоном, наприклад, через підсилення вертикальних країв – оскільки текст на знаках часто є вертикально структурованим і має високий контраст на світлому фоні. У деяких роботах було запропоновано обробку кольорових зображень, де додаткові кольорові ознаки номерних знаків використовувалися для поліпшення локалізації. Такі методи спиралися на унікальні комбінації кольорів номерних знаків, які перетворювали RGB-зображення в інші колірні простори, зокрема HSI або HSL (відтінок–насиченість–інтенсивність/світлість), для кращого виділення цільових об'єктів.

Окрім класичної обробки, також застосовувалися різноманітні класифікаційні підходи, які базувалися на фіксованих ознаках, таких як ознаки

Хаара. Для класифікації використовували популярні алгоритми: машини опорних векторів (SVM), штучні нейронні мережі (ANN) або навіть генетичні алгоритми, які імітували еволюційні процеси для пошуку оптимальних рішень.

Щодо результатів, то ефективність різних алгоритмів значно варіювалася– від 80% до 100%. Однак важливо зазначити, що через відсутність загальноприйнятих стандартних наборів тестів для LPR важко здійснювати справедливе та об'єктивне порівняння між різними методами. Наприклад, деякі автори надавали дані набору з грецькими номерними знаками, проте вони були недостатньо марковані для якісної перевірки. Більш повноцінним є набір даних тайванських номерних знаків (2049 зображень), які містили точні розмітки розташування номерних знаків і класифікацію символів, що дозволяло проводити більш детальний аналіз алгоритмів у різних умовах, таких як різні кути огляду або відстані до об'єкта.

Варто також розуміти, що під час оцінки ефективності враховується різний баланс між точністю та повнотою (recall). Деякі методи могли навмисно підвищувати повноту ціною точності, щоб не пропустити жодного кандидата на номерний знак, перед передачею на подальші етапи сегментації і розпізнавання символів.

Другим важливим етапом після локалізації була сегментація символів. Найпоширенішим підходом для цього було використання горизонтальної та вертикальної проекції пікселів зображення, які дозволяли виявляти піки між символами. Також CCA застосовувалася разом із аналізом висоти та ширини компонентів або у поєднанні з проекційними методами для фільтрації об'єктів неправильної форми.

Незважаючи на відносно високий контраст між текстом і фоном, глобальні методи порогового бінаризації часто були неточними в реальних складних сценаріях. Тому в таких випадках перевага надавалася локальним або адаптивним методам порогового розділення, які краще справлялися із неоднорідним освітленням та артефактами зображення.

Автори визнають, що прості методи, зокрема класичні проекції або базовий аналіз компонент, часто провалюються у складних реальних випадках або на зображеннях за межами навчального набору даних. Для підвищення надійності було запропоновано дворівневі алгоритми сегментації: перший рівень – попереднє виділення символів за допомогою спеціалізованих банків фільтрів (наприклад, у вигляді борони), а другий рівень – корекція сегментації за допомогою методів пошуку оптимального шляху для обробки з'єднаних або перекритих символів.

Третій завершальний етап – розпізнавання сегментованих символів. Для цього широко застосовувалися класифікатори, включаючи приховані марковські моделі (HMM), машини опорних векторів (SVM), штучні нейронні мережі (ANN) або комбінації різних класифікаторів. Також активно використовувалося зіставлення з шаблонами, особливо коли шрифти номерних знаків були відомі заздалегідь.

Метод зіставлення полягав у розрахунку метрик подібності (наприклад, середньоквадратичної похибки) між сегментованим символом та шаблонними зображеннями для визначення найкращого відповідника.

Критично важливим є те, що правильне розпізнавання номерного знака вимагає точного виділення і класифікації всіх символів на табличці, тому навіть незначні помилки на етапі сегментації можуть спричинити невірне прочитання всього номерного знака.

Для досягнення високої точності (до 99,5%) у розпізнаванні символів використовувалися штучні нейронні мережі, проте їх навчання вимагало величезних обсягів даних. Через це багато систем покладалися на гібридні рішення або на оптимізацію даних входів, наприклад, шляхом попереднього масштабування номерного знака так, щоб висота символів становила приблизно 20 – 25 пікселів.

У сучасних рішеннях, що базуються на глибинному навчанні, основну роль відіграють згорткові нейронні мережі (CNN). ШНМ складаються із кількох

послідовних шарів, де згорткові шари відповідають за витяг ознак (features) із вхідного зображення, а повнозв'язані шари на завершальній стадії виконують класифікацію або регресію.

Ядра згорткових шарів діють як фільтри, які активуються при виявленні специфічних ознак у даних. Після етапу вилучення ознак результати передаються до повнозв'язаних шарів, де відбувається узагальнення та остаточне прийняття рішення. Кількість шарів і параметрів варіюється залежно від складності завдання.

CNN стали незамінними у задачах розпізнавання зображень завдяки своїй здатності ефективно виявляти локальні та глобальні структури в даних. Це особливо важливо для обробки зображень, де інформація залежить від просторової взаємодії сусідніх пікселів, а не від індивідуальних значень каналів, як у традиційних табличних даних.

1.1 Опис повністю зв'язаного шару

Перш ніж перейти до формального опису згорткового шару в складі згорткової нейронної мережі (ШНМ), доцільно спершу розглянути математичну модель одного штучного нейрона, а також структуру повністю зв’язаного шару (fully connected layer). Це дозволить ввести базові позначення та краще зрозуміти, як ШНМ поєднує класичні нейронні підходи з просторовою обробкою даних.

Штучний нейрон є абстрактною моделлю, натхненною біологічним нейроном. У біологічній системі нейрон отримує сигнали через численні дендрити, обробляє їх у тілі клітини, а потім передає реакцію через єдиний аксон. Подібним чином, у математичній моделі штучного нейрона передбачено наявність кількох входів, кожен з яких асоційований із числовою вагою. Ці вхідні

значення, зважені відповідними коефіцієнтами, підсумовуються, і результат передається через нелінійну активаційну функцію, яка визначає вихід нейрона.

Формально, якщо маємо вхідний вектор:

$$\mathbf{x} = [x_1, x_2, ..., x_n],$$

вектор ваг:

$$w = [w_1, w_2, ..., w_n].$$

Та зсув (bias) b, то вихід нейрона можна записати як:

$$y = \phi(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) = \phi(w^T x + b),$$

де ϕ – це активаційна функція, наприклад, ReLU, сигмоїда або гіперболічний тангенс.

Повністю зв’язаний шар (або щільний шар) складається з багатьох таких нейронів, кожен з яких отримує на вхід усі значення з попереднього шару.

Таким чином, вихід кожного нейрона залежить від усієї вхідної інформації, що забезпечує здатність моделі до глобальної узагальненої обробки вхідних даних. Проте такий підхід втрачає просторову структуру даних, що особливо критично для зображень, де локальні патерни несуть важливу інформацію.

Саме для збереження локальних ознак і вводяться згорткові шари, які ми розглянемо далі.

1.2 Огляд шару згортки

На відміну від повністю зв’язаного шару, згортковий шар обробляє дані локально. Це означає, що кожен нейрон у згортковому шарі пов’язаний лише з обмеженою областю вхідних даних, званою рецептивним полем. Такий підхід особливо ефективний при роботі з зображеннями, де локальні особливості, як-от краї, текстури та фрагменти об'єктів, мають вирішальне значення для розпізнавання.

Математична модель згортки – згортковий шар застосовує операцію згортки (convolution) між вхідними даними та ядром (фільтром) – невеликою матрицею з навчуваними параметрами. Якщо позначити вхідне зображення як матрицю X, а ядро – як матрицю K, то результат згортки S визначається за формулою:

$$S(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n) \times X(i+m, j+n),$$

де M×N – розмір ядра;
i, j – координати на вихідній карті ознак (feature map).
Цей результат далі може бути переданий через активаційну функцію (наприклад, ReLU), що додає моделі нелінійності:

$$A(i,j) = \phi(\mathrm{S(i,j)}),$$

де ϕ – це нелінійна активаційна функція, яка вводить у модель здатність до навчання складних залежностей.

У практиці найчастіше використовують функцію ReLU (Rectified Linear Unit), яка визначається як:

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

Ця функція обнуляє всі від’ємні значення, зберігаючи позитивні без змін, що дозволяє мережі уникати проблеми зникання градієнтів та прискорює навчання.

Об’ємність згортки – якщо вхідні дані мають кілька каналів (наприклад, кольорове зображення з каналами RGB), кожне ядро також має відповідну глибину. Наприклад, для кольорового зображення розміром H×W×3H, кожне згорткове ядро матиме форму k×k×3k, де k – просторовий розмір фільтра. Після згортки з кожним ядром формується один канал на виході. Якщо таких ядер 64, отримаємо вихід H′×W′×64, тобто 64 карти ознак.

Під час згортки інформація на краях зображення може "втрачатися", оскільки фільтр не може повністю вміститись на краї. Щоб уникнути цього, застосовується padding (доповнення нулями), що дозволяє зберігати розміри вхідного і вихідного тензорів незмінними.

Переваги згорткових шарів.

1. Локальність зв’язків: кожен нейрон пов’язаний лише з локальною областю вхідного простору, що дозволяє ефективно захоплювати локальні патерни.
2. Спільне використання ваг: одне ядро використовується по всьому зображенню, що значно зменшує кількість параметрів.
3. Інваріантність до зсувів: завдяки локальному скануванню ті самі ознаки можуть розпізнаватися в різних частинах зображення.

На підсумок можемо сказати, що згорткові шари формують карти ознак, які виділяють низькорівневі ознаки (лінії, кути, текстури) на початкових шарах та більш складні – на глибших шарах мережі. Але після кожної згортки часто

застосовується шар об’єднання (pooling), який допомагає зменшити розмірність простору ознак, зберігаючи при цьому найважливішу інформацію.

1.3 Максимальне об'єднання

Зазвичай між згортковими шарами в згорткових нейронних мережах вставляють шари максимального об'єднання (Max Pooling). Основна мета цього шару – зменшити просторові розміри карти ознак, тобто зменшити висоту та ширину, не змінюючи кількість каналів. Це дозволяє збільшити ефективне поле зору наступних згорткових ядер без потреби в їхньому фізичному розширенні, а отже – без суттєвого зростання обчислювальної складності.

Крім того, Max Pooling знижує чутливість моделі до незначних просторових зсувів або деформацій об'єктів, що сприяє побудові більш узагальнених моделей, здатних коректно розпізнавати ознаки об'єктів попри їх незначні варіації. Таким чином, шар об'єднання діє як просторова агрегація інформації, яка підкреслює найбільш виразні ознаки в межах локальних регіонів зображення.

Операція максимального об'єднання математично описується наступною формулою:

$$y(i,j,k) = \max_{m=(i-1)_{S_x+1}}^{(i-1)_{S_x+S_m}} \max_{n=(j-1)_{S_y+1}}^{(j-1)_{S_y+S_n}},$$

де y(i,j,k) – вихідне значення шару об'єднання у позиції (i,j) та каналі k;

h(m,n,k) – значення на вхідній карті ознак у відповідній області;

S_x, S_y – крок (stride) по горизонталі та вертикалі відповідно;

s_m, s_n – розміри вікна об'єднання (наприклад, 2×2 або 3×3).

Найчастіше кроки s_x та s_y встановлюються рівними розміру вікна, тобто область перегляду не перекривається, і кожен фрагмент карти ознак береться лише один раз.

Припустимо, що шар Max Pooling застосовується до карти ознак розміром 32×32 пікселі з використанням 2×2 вікна та кроку 2. У такому випадку розмір карти ознак після обробки зменшиться до 16×16. Якщо таких каналів було 64, то кількість каналів збережеться, але кожен з них стане компактнішим.

Комбінування згорткових шарів із шарами об'єднання дозволяє моделі спочатку виділяти локальні ознаки (наприклад, контури, текстури), а на глибших рівнях – розпізнавати більш складні структури, такі як форми, об'єкти або їхні взаємозв’язки. З кожним новим рівнем у мережі відбувається перехід від просторово специфічних до семантично узагальнених ознак, що є ключовим у задачах класифікації, детекції об’єктів чи сегментації зображень.

У випадках, коли завдання не вимагає збереження точної просторової інформації (наприклад, при класифікації), у фіналі мережі іноді застосовують глобальне об'єднання (Global Max або Average Pooling), що повністю агрегує просторову інформацію та підсилює інваріантність до зсувів.

Таким чином, шари максимального об'єднання виконують не лише роль зменшення розмірності, а й сприяють узагальненню ознак, знижують переобучення та покращують інваріантність до змін положення. Всі ці властивості гармонійно поєднуються в архітектурі згорткових нейронних мереж (CNN), де згорткові, об'єднувальні, нормалізаційні та активаційні шари формують чітко структуровану модель для обробки зображень. Розглянемо детальніше, як побудована така структура, та яку роль у ній відіграє кожен компонент.

1.4 Структура згорткових нейронних мереж (CNN)

Як уже згадувалося, типова структура згорткової нейронної мережі (CNN) поділяється на два основних етапи: етап виділення ознак і етап прийняття рішення (висновку).

На етапі виділення ознак, модель зосереджується на виявленні та побудові ієрархії ознак з вхідного зображення. Він складається з послідовності згорткових шарів (Convolutional Layers) з невеликими ядрами – зазвичай розміром 3×3. Ці шари призначені для виявлення локальних патернів, таких як краї, кути, текстури, а також більш складних структур на глибших рівнях.

Після кожного згорткового шару зазвичай застосовується нелінійна активація ReLU (Rectified Linear Unit), яка дозволяє моделі враховувати лише позитивні відповіді нейронів, підвищуючи ефективність навчання.

Для поступового зменшення просторових розмірів ознак та зменшення обчислювального навантаження, між згортковими шарами вставляють шари максимального об’єднання (Max Pooling), зазвичай з вікном 2×2. Таке поєднання дає змогу збільшити кількість каналів (ядер) без надмірного росту обчислювальної складності. З кожним новим шаром збільшується поле зору – область, яку охоплює одне значення ознаки у глибоких шарах відносно вхідного зображення.

Ця архітектура дозволяє моделі ієрархічно навчатися: від простих елементів (пікселів і країв) до складних ознак, таких як форми, структури й навіть об’єкти в цілому.

На етапі прийняття рішень після того як були згенеровані просторово зменшені та семантично насичені карти ознак, вони передаються на етап класифікації. Цей етап зазвичай реалізується за допомогою кількох повнозв'язних шарів (Fully Connected Layers), які працюють так само, як у класичних багатошарових перцептронах (MLP).

Кожен повнозв’язний шар виконує зважену комбінацію вхідних ознак і застосовує нелінійну функцію активації ReLU, що дає змогу мережі апроксимувати складні функції та розділяти дані з високою точністю.

Останній шар завершується функцією активації SoftMax, яка перетворює вихідні значення на вірогідності належності зображення до кожного з класів. SoftMax гарантує, що сума всіх вихідних значень дорівнює 1, що зручно для інтерпретації результатів.

Кількість повнозв’язних шарів та нейронів у них впливає на гнучкість і складність межі прийняття рішень. Наприклад, більше шарів дозволяє мережі створювати більш складні розділення між класами, тоді як зменшення кількості нейронів у напрямку до виходу зменшує кількість параметрів та ймовірність перенавчання.

На рис. 1.2 представлено приклад базової архітектури CNN, яка містить усі описані компоненти – згорткові шари, шари об'єднання, повнозв’язні шари та SoftMax. Ця структура може варіюватися залежно від задачі, але загальні принципи залишаються подібними.

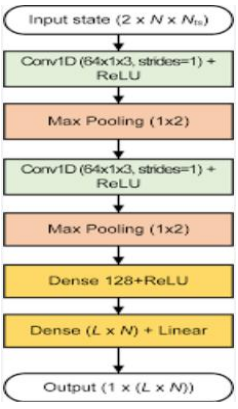


Рисунок 1.2 – Приклад простої архітектури CNN[2]

[2] Джерело зображення:
https://www.eurasip.org/Proceedings/Eusipco/Eusipco2020/pdfs/0001566.pdf

Побудова такої архітури – це лише один аспект CNN. Щоб краще зрозуміти, як згорткові нейронні мережі працюють та які елементи впливають на їх ефективність, варто глибше розглянути структуру CNN, включаючи типи шарів, способи нормалізації, стратегії зменшення розмірності та роль кожного етапу в навчанні моделей глибокого навчання.

1.5 Процес навчання нейронної мережі

Щоб мати змогу класифікувати щось значуще, нейронна мережа повинна навчитися відповідному набору вагових коефіцієнтів, які дозволяють їй коректно інтерпретувати вхідні дані. Це навчання відбувається за допомогою навчального набору мічених прикладів, де кожен приклад має вхідні дані та правильну відповідь (мітку класу).

Для вибору найкращої моделі часто використовують валідаційний набір, який не бере участі у безпосередньому навчанні, але допомагає налаштовувати гіперпараметри (наприклад, швидкість навчання, кількість шарів або нейронів) та застосовувати механізми ранньої зупинки, коли модель перестає покращуватися. Однак, оскільки валідаційний набір впливає на вибір остаточної архітектури, для об’єктивного оцінювання здатності мережі узагальнювати дані використовують окремий тестовий набір, який не використовувався ні в навчанні, ні у валідації.

Навчання мережі зазвичай реалізується методом зворотного поширення помилки (backpropagation) у поєднанні з методом стохастичного градієнтного спуску (SGD). У цьому процесі ваги мережі ініціалізуються випадковим чином, а потім поступово коригуються на основі помилки, яка виникає між передбаченим виходом мережі та реальним значенням. Помилка поширюється

від вихідного шару назад до входу, і градієнти використовуються для оновлення ваг у напрямку, який зменшує цю помилку.

Для підвищення ефективності навчання використовується міні-батч навчання, коли ваги оновлюються не після кожного зразка, а після обробки невеликої групи (міні-пакета). Це дозволяє зменшити флуктуації в оновленнях і збільшити швидкість обчислень завдяки паралельності. Занадто маленькі міні – пакети можуть призвести до нестабільного навчання, а занадто великі – до повільної адаптації моделі.

Процес навчання зазвичай триває кілька епох, кожна з яких охоплює повний прохід усіх зразків навчального набору. Щоб мінімізувати помилку, використовується функція втрат. Для регресії популярною є середньоквадратична помилка (MSE):

$$E(W) = \frac{1}{C}\sum_{i=1}^{C}(y_i^o - t_i)^2,$$

де E(W) – функція похибки для набору ваг;

W, y_i^o – вихід моделі;

t_i – цільове значення.

Для задач класифікації ефективніше працює функція втрат крос-ентропії, яка краще адаптується до задач з багатьма класами. Оновлення ваг за SGD виконується за формулою:

$$W := W - \eta\nabla \mathrm{WE(W)},$$

де η – коефіцієнт швидкості навчання.

Щоб покращити збіжність, часто використовують адаптивні алгоритми оптимізації, такі як AdaGrad, RMSprop, або Adam, які змінюють величину кроку в залежності від історії градієнтів.

Після опрацювання загальних принципів навчання нейронних мереж, важливо розглянути їхню архітектурну організацію, оскільки саме структура мережі визначає, які патерни вона здатна виявляти у вхідних даних. Особливої уваги заслуговують згорткові нейронні мережі (Convolutional Neural Networks, CNN), які показали надзвичайну ефективність у розпізнаванні образів, класифікації зображень та інших візуальних задачах. Далі ми детально розглянемо структуру CNN, її основні компоненти та їх взаємодію.

1.6 Архітектура повністю згорткових нейронних мереж

Описана вище архітектура нейронної мережі має обмеження: вона може працювати лише із зображеннями фіксованого розміру. Це зумовлено тим, що повністю зв’язані (fully connected) шари мають сталу кількість входів, яка залежить від розміру вхідного зображення. Через це перед подачею зображення на ці шари потрібно точно узгодити розміри – зазвичай шляхом фіксації розміру вхідного зображення або за допомогою попередньої обробки.

Однак це обмеження можна подолати, якщо замінити повністю зв’язані шари на згорткові (convolutional) шари. Такий підхід дозволяє використовувати мережу для обробки зображень будь-якого розміру. У результаті замість одного результату класифікації ми отримаємо матрицю результатів – кожен елемент цієї матриці відповідає класифікації окремої області (вікна) вхідного зображення. Таким чином, модель функціонує як класифікатор зі зсувними вікнами, але набагато ефективніше, оскільки не потребує повторної обробки однакових пікселів, як це трапляється при ручному ковзанні вікна.

Для побудови такої повністю згорткової мережі можна взяти звичайну нейронну мережу з повністю з’єднаними шарами й замінити кожен з них згортками. Замість першого повністю зв’язаного шару використовується згортка

з ядром такого розміру, щоб охопити всю карту ознак, яка виходить після етапу вилучення ознак (feature extraction). Кількість ядер у згортці дорівнює кількості нейронів у початковому повністю зв’язаному шарі. Оскільки не застосовується простір для зсуву (stride), кожне ядро створює лише один вихід, тому результат матиме форму однієї точки (1×1) з глибиною, що дорівнює кількості ядер.

Наступні повністю зв’язані шари (якщо вони є) замінюються згортками з ядрами розміром 1×1. Кожне ядро відповідає одному нейрону відповідного шару і поєднує всю глибину попереднього шару в одну числову відповідь.

Застосування такої мережі до зображення більшого розміру автоматично створює просторово зсунутий набір класифікацій. Це аналогічно використанню ковзного вікна над зображенням, але більш ефективно завдяки перевагам згорткових операцій. Таким чином, мережа здатна класифікувати кілька областей зображення за один прохід, що суттєво зменшує обчислювальні витрати.

1.7 Постановка задачі

У сучасному світі, де інформаційні технології стрімко розвиваються, автоматичне розпізнавання номерних знаків транспортних засобів набуває все більшого значення. Така технологія є ключовою для забезпечення громадської безпеки, автоматизації транспортної інфраструктури та ефективного контролю за дорожнім рухом.

Мета дослідження – розробити ефективну систему розпізнавання номерних знаків на відео із застосуванням сучасних методів машинного та глибинного навчання, зокрема згорткових нейронних мереж (CNN).

Завдання дослідження**.**

1. Проаналізувати наявні методи та технології розпізнавання номерних знаків на зображенні.
2. Оцінити ефективність традиційних підходів у порівнянні з методами глибинного навчання.
3. Спроєктувати архітектуру згорткової нейронної мережі, оптимізованої для розпізнавання номерів.
4. Реалізувати програмний засіб для автоматичного розпізнавання номерних знаків автомобілів на зображеннях.
5. Навчити модель на реальних зображеннях та протестувати її якість.
6. Порівняти точність та ефективність створеної системи з існуючими рішеннями.
7. Здійснити функціонально-вартісний аналіз запропонованої системи.

Очікувані результати**.**

1. Побудова моделі, здатної з високою точністю виявляти та розпізнавати номерні знаки на зображенні.
2. Розробка програмного забезпечення, яке легко інтегрується в сучасні системи відеонагляду.
3. Оптимізація витрат та підвищення ефективності існуючих рішень у сфері автоматичного розпізнавання номерів.
4. Отримання результатів, що мають наукову цінність і можуть стати основою для подальших досліджень у галузі комп’ютерного зору та глибинного навчання.

Висновки до розділу 1

У першому розділі було розглянуто сучасний стан розробок у галузі розпізнавання автомобільних номерних знаків. Завдяки стрімкому розвитку

глибинного навчання, особливо згорткових нейронних мереж (CNN), такі системи досягли високого рівня точності та автоматизації.

Традиційні підходи до розпізнавання номерних знаків включали три основні етапи.

1. Локалізація номерного знака – визначення області з номером на зображенні.
2. Сегментація символів – поділ зображення номерного знака на окремі символи.
3. Розпізнавання символів – ідентифікація кожного символу за допомогою відповідних алгоритмів.

Для цих етапів використовувались методи комп'ютерного зору, зокрема:

– виявлення країв (наприклад, оператором Собеля або Кенні);
– аналіз зв'язаних компонент (CCA) для виділення областей символів;
– вертикальні та горизонтальні проекції для знаходження меж символів;
– адаптивне порогове розрізнення для покращення контрасту.

Однак такі методи часто виявляються нестійкими до змін освітлення, шуму на зображеннях або нестандартних шрифтів номерів.

Використання згорткових нейронних мереж дало змогу значно покращити результати. CNN автоматично витягують ознаки з зображень без потреби в ручному налаштуванні або проєктуванні ознак, що особливо корисно для зображень низької якості або складних сцен з фоновими об’єктами. Це дозволяє системам розпізнавати номери в реальному часі навіть у складних умовах (наприклад, при поганому освітленні або перекриттях).

Проте, застосування CNN також має виклики, пов’язані з великою різноманітністю форматів номерних знаків у різних країнах, а також змінами у зовнішньому вигляді через бруд, подряпини, тіні тощо.

Таким чином, зроблений огляд підтверджує, що глибинне навчання, зокрема CNN, є найбільш перспективним напрямом у створенні сучасних систем

розпізнавання номерних знаків. Це відкриває нові можливості для впровадження інтелектуальних транспортних систем і смарт-міст, де потрібні надійні, точні та автоматизовані інструменти контролю дорожнього руху.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА НАВЧАННЯ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ

У цьому розділі розглядається повний цикл розробки системи розпізнавання номерних знаків транспортних засобів із використанням згорткових нейронних мереж (CNN) та сучасних архітектур комп’ютерного зору. З огляду на специфіку завдання, система повинна не лише визначати місце розташування номерного знаку на зображенні, а й розпізнавати кожен окремий символ, що входить до його складу. Для цього було побудовано дві взаємопов’язані моделі: модель локалізації об’єктів (YOLO) та модель розпізнавання символів (CNN), а також реалізовано підготовку даних, пов’язану з урахуванням реальних умов використання.

Завдання розпізнавання номерних знаків умовно розділене на два етапи: виявлення області номерного знака на зображенні та розпізнавання символів усередині цієї області. На першому етапі застосовується згорткова нейронна мережа, натренована за допомогою моделі YOLOv5 – однієї з найефективніших сучасних моделей для детекції об'єктів у режимі реального часу. YOLO дозволяє з високою точністю і швидкістю ідентифікувати положення номерного знака на зображенні навіть у складних умовах: під різними кутами огляду, при змінному освітленні або частковому перекритті об'єкта.

На другому етапі система виконує розпізнавання символів за допомогою спеціально спроєктованої згорткової нейронної мережі, яка приймає фрагмент з номером і перетворює його у послідовність літер та цифр. У даному випадку символи розпізнаються як багатокласова класифікаційна задача з 24 класами (10 цифр та 14 літер). Для підвищення точності моделі реалізовано глибоку архітектуру, яка містить послідовні шари згортки, активації ReLU, нормалізації, пулінгу та повнозв’язні шари з функцією softmax на виході. Окрім того, у роботі було протестовано варіант повністю згорткової мережі (Fully Convolutional

Network, FCN) для розпізнавання символів без використання повнозв’язних шарів, що дозволяє зменшити кількість параметрів і покращити генералізацію моделі.

Особливу увагу приділено попередній обробці та підготовці даних, які включають зміну розмірів, нормалізацію піксельних значень, а також використання методів аугментації (обертання, масштабування, зміна яскравості та контрасту). Ці дії сприяють розширенню набору навчальних прикладів і роблять модель стійкішою до змін у зовнішньому середовищі.

Для зниження ймовірності помилок у розпізнаних символах передбачено етап постобробки результатів. Він може включати використання вбудованих правил, словників допустимих шаблонів номерів, а також евристичні перевірки та коригування символів, що легко плутаються (наприклад, «0» і «O», «5» і «S» тощо). Це дозволяє підвищити надійність розпізнавання навіть за умов поганої якості зображення.

Таким чином, розроблена система є повноцінним рішенням для автоматичного розпізнавання номерних знаків, що включає детекцію, класифікацію, постобробку і враховує особливості реального середовища. Усі моделі реалізовано за допомогою бібліотек TensorFlow, Keras та Ultralytics YOLO, що забезпечує гнучкість, масштабованість і можливість подальшої оптимізації.

2.1 Розробка мережі для розпізнавання номерних знаків

У межах даного етапу було реалізовано повноцінну конвеєрну архітектуру для розв’язання задачі детекції українських автомобільних номерних знаків. Запропоноване рішення включає низку послідовних етапів: ініціалізацію апаратного прискорення на базі CUDA, завантаження та підготовку моделі

YOLOv5s, навчання на синтетично згенерованому наборі даних, а також валідацію продуктивності моделі за допомогою стандартних метрик точності. Базовою моделлю для розробки було обрано варіант YOLOv5s (версія «small») у зв’язку з її компактністю, високою швидкістю обробки зображень і достатнім рівнем точності для застосування в реальному часі.

2.1.1 Ініціалізація апаратного прискорення та модельна архітектура

На початковому етапі здійснюється перевірка доступності графічного прискорювача, зокрема підтримки CUDA (Compute Unified Device Architecture), що є пропрієтарною платформою паралельних обчислень від NVIDIA. У разі виявлення доступного GPU, модель глибокої нейронної мережі переміщується до пам’яті пристрою за допомогою конструкції .to(device), що забезпечує істотне зменшення часу навчання й підвищення ефективності обчислень.

В основі обраної архітектури лежить модель YOLOv5s – одна з модифікацій серії «You Only Look Once», адаптована для задач об’єктної детекції в умовах обмежених обчислювальних ресурсів. Архітектурна структура моделі охоплює:

- Backbone (CSPDarknet53) – відповідає за витягнення глибинних ознак із вхідного зображення шляхом застосування серії згорткових операцій та залишкових з’єднань (residual connections);
- Neck (PANet) – виконує агрегацію ознак на декількох масштабах (multi-scale feature fusion), що дозволяє виявляти об’єкти різних розмірів;
- Head (YOLO Layer) – генерує підсумкові передбачення у вигляді координат обмежувальних рамок, класів об’єктів та рівня впевненості моделі.

Загальна кількість параметрів моделі становить приблизно 7,5 мільйона, а її об’єм – близько 14 мегабайт, що робить її придатною до вбудовування в системи з обмеженими ресурсами (наприклад, вбудовані системи, мобільні пристрої).

2.1.2 Формат вхідних даних та обробка зображень

Для навчання моделі було сформовано синтетичний набір зображень номерних знаків, що відповідають державному українському формату. Синтетичний підхід дозволяє забезпечити необхідне розмаїття умов (фон, освітлення, шрифт, кут огляду), не покладаючись на складні процеси збору реальних даних.

Усі зображення були попередньо масштабовані до фіксованого розміру 640×640 пікселів. Такий вибір розміру зумовлений компромісом між швидкістю обробки та здатністю моделі до збереження важливих дрібних деталей. Формат розмітки базується на стандарті YOLO: кожен об’єкт представлено координатами центра та розмірами обмежувальної рамки, нормалізованими відносно ширини та висоти зображення. Усі шляхи до зображень та анотацій задаються в конфігураційному YAML – файлі.

2.1.3 Архітектура моделі YOLOv5s

YOLOv5s належить до родини моделей, орієнтованих на швидке й точне виявлення об’єктів в одному проході через зображення. Її архітектура поєднує найсучасніші технічні рішення з глибокого навчання:

- CSPDarknet – структура з покращеними залишковими зв’язками та використанням Cross Stage Partial Connections, що підвищують якість витягнутих ознак;
- Path Aggregation Network (PANet) – сприяє агрегації інформації з різних рівнів ознак для кращої локалізації об’єктів на зображенні;
- YOLO Head – забезпечує генерацію передбачень з використанням anchor boxes і передбачає клас, рамку та впевненість для кожного детектованого об’єкта.

YOLOv5s оптимізовано для виконання в реальному часі, і її продуктивність дозволяє проводити інференс з високою частотою кадрів навіть на середньому обладнанні.

2.1.4 Процес навчання

Навчання проводиться протягом 50 епох з використанням пакету зображень розміром 16 (batch size = 16). Оптимізація моделі здійснюється шляхом мінімізації загальної функції втрат:

$$\mathcal{L}_{total} = \lambda_{box} \times \mathcal{L}_{box} + \lambda_{obj} \times \mathcal{L}_{obj} + \lambda_{cls} \times \mathcal{L}_{cls},$$

де $\mathcal{L}_{box}$ – втрата для локалізації (IoU);

$\mathcal{L}_{obj}$ – втрата впевненості у присутності об'єкта;

$\mathcal{L}_{cls}$ – втрата класифікації.

Для оновлення ваг моделі використовуються поширені оптимізатори: стохастичний градієнтний спуск (SGD) або Adam. Значення початкової швидкості навчання, коефіцієнти регуляризації та типи аугментацій (дзеркальне відображення, масштабування, мозаїка) задаються або вручну, або через YAML-

конфігурацію. Така гнучкість дозволяє адаптувати модель до різних умов задачі та підвищити її узагальнювальну здатність.

2.1.5 Оцінка результатів

Після завершення навчання здійснюється оцінка продуктивності моделі на окремій тестовій вибірці. Для цього застосовуються такі ключові метрики:

- mAP@0.5 – середня точність (mean Average Precision) при IoU ≥ 0.5;
- mAP@0.5:0.95 – усереднене значення mAP для порогів перекриття IoU від 0.5 до 0.95 з кроком 0.05;
- Precision – точність, що відображає частку правильних позитивних спрацювань;
- Recall – повнота, що вимірює здатність моделі виявити всі релевантні об'єкти.

mAP є стандартною метрикою в задачах об’єктної детекції, що дозволяє комплексно оцінити точність локалізації та класифікації. Високі значення Precision свідчать про низьку кількість хибнопозитивних детекцій, тоді як Recall демонструє здатність моделі не втрачати об’єкти.

2.2 Розробка мережі для розпізнавання символів

Під час дослідження була розроблена згорткова нейронна мережа для автоматизованого розпізнавання послідовностей символів фіксованої довжини на зображеннях. Метою розробки є побудова моделі, здатної до точного і стійкого визначення 8-символьних послідовностей у форматі RGB-зображень

розміром 64×64 пікселі, що є типовим для задач, пов’язаних із візуальним аналізом табличок, зокрема державних номерних знаків.

На відміну від традиційних підходів, де спочатку здійснюється локалізація і сегментація кожного окремого символу з наступною класифікацією, розроблена модель реалізує підхід «кінець-в-кінець» (end-to-end). Це означає, що на вході подається повна послідовність символів у вигляді зображення, а на виході – послідовність розпізнаних символів, що значно спрощує архітектуру всієї системи та підвищує її швидкодію.

Модель орієнтована на класифікацію 24 унікальних символів, до яких можуть входити:

- цифри (0–9);
- латинські великі літери (без візуально схожих, таких як I, O, Q);
- додаткові символи, залежно від формату міток (наприклад, пробіл або тире).

Наведено приклад вхідного зображення номерного знаку автомобіля (рис. 2.1).

Рисунок 2.1 – Приклад зображення державного номерного знаку автомобіля

2.2.1 Виділення ознак

На етапі виділення ознак (feature extraction) реалізовано послідовність згорткових шарів з поступовим нарощуванням кількості фільтрів і зменшенням просторової роздільності ознак:

- перший згортковий шар застосовує 32 фільтри розміром 3×3, кожен із функцією активації ReLU, для виявлення простих локальних структур (ліній, кутів). Далі йде шар підвибірки MaxPooling2D із вікном 2×2, що дозволяє зменшити розмірність і зберегти найбільш інформативні елементи;
- другий згортковий шар включає 64 фільтри 3×3 з ReLU та ще один шар MaxPooling 2×2. Цей рівень відповідає за виділення більш складних контурів і взаємозв’язків між локальними ознаками;
- третій згортковий шар використовує 128 фільтрів 3×3. Після нього знову виконується підвибірка. На цьому етапі мережа вже здатна ідентифікувати складніші патерни, які можуть відповідати цілим фрагментам символів або навіть комбінаціям із кількох символів.

Після проходження всіх згорткових рівнів формується тривимірний тензор ознак. Для подальшої обробки цей тензор перетворюється на одновимірний вектор за допомогою операції Flatten, що дозволяє передати дані на щільні шари класифікації.

2.2.2 Класифікація

Класифікаційна частина моделі складається з двох основних шарів:

- проміжний щільний шар (Dense) на 128 нейронів із функцією активації ReLU, який забезпечує нелінійне перетворення простору ознак та дозволяє навчитись складним зв’язкам між характеристиками вхідного зображення;
- вихідний щільний шар має 192 нейрони, що відповідає 8 позиціям **у** послідовності × 24 класи символів. На цьому шарі застосовується функція активації Softmax, яка дозволяє отримати ймовірнісний розподіл для кожного символу у відповідній позиції.

Після обчислення вихідного вектору розміром (192,), він перетворюється у тензор форми (8, 24) шляхом операції Reshape, де кожен рядок відповідає окремій позиції у послідовності символів, а кожна колонка – ймовірності належності до певного класу.

2.2.3 Локалізація

У межах запропонованої архітектури реалізовано окремий етап попередньої обробки зображень, метою якого є автоматичне виявлення та виділення області номерного знака на зображенні транспортного засобу. Приклад такого зображення навдено на рис. 2.2, це дозволяє забезпечити високу якість вхідних даних для подальшого етапу розпізнавання символів. На рис. 2.3 зображено приклад номерного знаку автомобіля після виявленя та виділення області номерного знаку.

В якості детектора використано згорткову нейронну мережу типу YOLOv5, натреновану на датасеті з мітками номерних знаків. Модель приймає на вхід зображення довільного розміру у форматі RGB та виконує детекцію об'єктів класу "номерний знак", повертаючи координати відповідної області у вигляді прямокутного вікна (bounding box). Надалі зображення обрізається за цими

координатами, масштабується до розміру 64×64 пікселі та зберігається у визначену директорію у форматі JPEG або PNG.

Збережені зображення номерних знаків використовуються як вхід для мережі розпізнавання послідовностей символів.

Такий підхід дозволяє:

- автоматизувати весь цикл обробки вхідного зображення – від отримання до розпізнавання;
- значно зменшити шумову складову та фонову інформацію, які не є релевантними для розпізнавання;
- підвищити точність моделі за рахунок покращеної якості вхідних даних.

Для валідації роботи модуля локалізації проведено візуальну перевірку виділених фрагментів на тестовій вибірці. У випадку помилкової локалізації або невиявлення номерного знака користувач має можливість виконати ручне коригування області інтересу або повторну обробку зображення після попередньої фільтрації (наприклад, покращення контрасту, зменшення шуму).

Рисунок 2.2 – Приклад вхідного зображення автомобіля з номерним знаком, перед обробкою

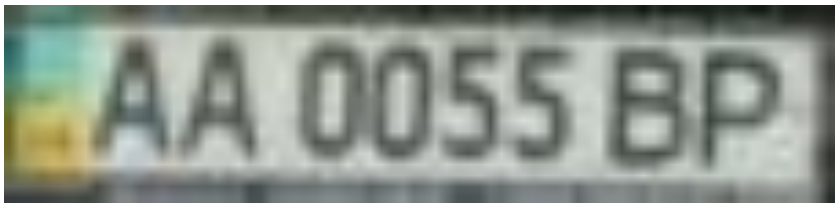


Рисунок 2.3 – Приклад номерного знаку автомобіля після виявленя та виділення області номерного знаку

2.2.4 Підготовка навчальних зразків

Для тренування моделі використовуються зображення повних послідовностей символів, які:

- вирізані з більших зображень вручну або автоматично;
- приведені до розміру 64×64 пікселі;
- нормалізовані до діапазону [0, 1] шляхом поділу на 255.

Попередня обробка також може включати:

- зміну масштабу та вирівнювання (наприклад, компенсацію нахилу);
- аугментацію: додавання шуму, випадкові зсуви, зміну яскравості або контрасту, що дозволяє покращити узагальнювальну здатність моделі.

2.2.5 Навчання моделі (позитивні приклади)

Процес навчання реалізується за допомогою функції втрат categorical_crossentropy, яка є стандартною для багатокласової класифікації з розміткою у вигляді one-hot encoding. Для оптимізації використовується Adam-

оптимізатор з типовими параметрами (learning_rate = 0.001), що забезпечує стабільну і швидку збіжність.

Як метрика контролю використовується accuracy – відсоток правильно класифікованих символів у послідовності. Навчання проводиться на великій кількості позитивних прикладів, які точно відповідають бажаному формату 8 символів. На рис. 2.4 зображено позитивні приклади навчання символів.

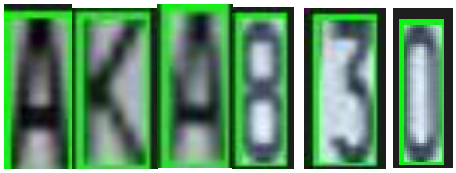


Рисунок 2.4 – Приклади позитивних вибірок для навчання детектора символів CNN

2.2.6 Гнучкість архітектури (негативні випадки, адаптація)

Попри фіксовану реалізацію для 8 символів, запропонована модель є архітектурно масштабованою. Ключовий параметр – довжину послідовності sequence_length – можна змінювати відповідно до завдання. Наприклад, для розпізнавання коротших кодів (6 символів) або довших серій (до 10 символів), потрібно лише відповідно змінити кількість вихідних нейронів та форму тензора на виході.

Аналогічно, кількість класів може бути адаптована під конкретний алфавіт, мову або спеціальні позначення, які використовуються в тій чи іншій країні або домені застосування. На рис. 2.5 зображено негативні приклади навчання символів.

Рисунок 2.5 – Приклади негативних вибірок для навчання детектора символів CNN

2.3 Застосування повністю згорткових мереж у розпізнаванні

У рамках вирішення задачі семантичної сегментації зображень номерних знаків було побудовано повністю згорткову нейронну мережу (Fully Convolutional Network, FCN), архітектура якої повністю виключає наявність повнозв'язаних шарів. Такий підхід забезпечує здатність мережі обробляти вхідні зображення довільного просторового розміру, що особливо актуально для задач, пов'язаних з аналізом реальних фотознімків, отриманих у нестандартних умовах освітлення, ракурсів та фокусної відстані.

Вхідним набором для навчання моделі слугували попередньо вирізані та нормалізовані зображення номерних знаків. Кожне зображення було приведене до розміру 64×64 пікселі, а відповідна маска сегментації – до такого ж розміру з відповідною анотацією пікселів у вигляді цілих чисел від 0 до *N-1*, де *N* – кількість класів. У нашій реалізації сегментації використано 24 класи, що відповідають різним символам кириличного та латинського алфавітів, цифрам, а також спеціальним класам для тла та службових позначень.

Архітектурно модель починається з кількох згорткових блоків, які включають:

- згортки (Conv2D) з ядром 3×3;
- активацію ReLU;

- паддінг типу 'same', що дозволяє зберігати незмінний розмір карти ознак;
- регуляризацію (Dropout) з імовірністю 0.2 для зниження ризику перенавчання.

Після кожної пари згорткових блоків застосовувався шар субдискретизації MaxPooling2D(pool_size=(2, 2)), який зменшував просторовий розмір карт ознак у 2 рази. Таким чином, при вхідному зображенні 64×64 мережа переходить до узагальненого представлення розміром 16×16, зберігаючи при цьому суттєві семантичні ознаки, важливі для розпізнавання окремих символів.

Етап декодування реалізовано за допомогою симетричної послідовності UpSampling2D(size=(2,2)) і Conv2DTranspose, які здійснюють поступове масштабування карти ознак до первинного розміру зображення. Це дозволяє зберегти та відновити просторові співвідношення між об’єктами, що критично важливо для піксельного передбачення. Додатково, використано механізм skip connections, який копіює ознаки з відповідних рівнів енкодера до декодера, що сприяє точнішому відновленню меж символів.

Кінцевий шар представлений транспонованою згорткою з softmax – активацією по кожному пікселю, що дозволяє моделі повернути ймовірнісний розподіл класів у форматі тензора розміру (64, 64, 24). Така структура дає змогу не лише класифікувати кожен піксель, а й визначити ступінь впевненості моделі в прийнятому рішенні.

У процесі навчання використовувалась функція втрат категоріальної крос-ентропії (categorical_crossentropy), яка оптимально підходить для багатокласової піксельної класифікації. Навчання проводилось із використанням оптимізатора Adam з початковою швидкістю навчання 1e-4. Було реалізовано систему ранньої зупинки (EarlyStopping), що відстежувала значення функції втрат на валідаційній вибірці та запобігала перенавчанню.

Код реалізації створено з використанням бібліотек TensorFlow 2.x і Keras, що забезпечують гнучкість у побудові архітектур та інтеграції з GPU. Усі

тензори були оброблені у форматі float32, а зображення масштабовано до діапазону [0, 1] перед подачею на вхід.

Переваги даної архітектури полягають у:

- збереженні топології зображення на всіх рівнях моделі;
- здатності узагальнювати на зображеннях різних розмірів без необхідності зміни структури;
- відсутності повнозв’язаних шарів, що зменшує кількість параметрів і покращує швидкодію;
- можливості застосування до задач в реальному часі за рахунок швидкої інференції.

Таким чином, повністю згорткова модель демонструє високу ефективність у задачі розпізнавання символів на номерних знаках за допомогою піксельної сегментації, забезпечуючи як локальну точність, так і глобальне просторове узгодження.

2.4 Підготовка та оптимізація даних для розпізнавання

Підготовка та оптимізація даних є важливими етапами в будь-якому процесі навчання нейронних мереж, особливо коли йдеться про завдання комп'ютерного зору, такі як сегментація зображень. Для досягнення високої продуктивності моделі важливо правильно підготувати дані, а також застосувати різноманітні техніки оптимізації, що дозволяють зменшити час обробки та підвищити точність. Ось основні кроки, які слід врахувати при підготовці та оптимізації даних для задачі розпізнавання.

2.4.1 Попередня обробка зображень

Перед подачею зображень в модель важливо провести кілька етапів попередньої обробки:

- зміна розміру зображень (Resizing): оскільки зображення можуть мати різні розміри, їх потрібно привести до однакового розміру. В нашому випадку, наприклад, зображення будуть змінюватися до розміру 64×64 або іншого, що підходить для вашої мережі. Це важливо для стабільного навчання та зменшення складності моделі;
- нормалізація (Normalization): для покращення процесу навчання важливо привести значення пікселів до одного масштабу, зазвичай в інтервалі [0, 1] або [-1, 1]. Це може бути досягнуто шляхом ділення значення пікселів на 255 (якщо значення пікселів знаходяться в діапазоні від 0 до 255). Така нормалізація дозволяє мережі краще і швидше навчатися, оскільки значення зображення будуть знаходитися в більш керованому діапазоні;
- аугментація даних (Data Augmentation): оскільки датасети для задач сегментації можуть бути обмеженими за кількістю зображень, аугментація є потужним інструментом для створення нових зображень шляхом застосування різноманітних перетворень. Це можуть бути:
  - поворот зображень на випадковий кут;
  - масштабування та зсуви зображень;
  - горизонтальне чи вертикальне віддзеркалення зображень;
  - зміна яскравості або контрасту зображень.

Аугментація дозволяє збільшити кількість навчальних даних, що покращує узагальнюючу здатність моделі та зменшує ймовірність перенавчання (overfitting).

2.4.2 Анотовані дані для сегментації

Для задач сегментації кожному пікселю зображення присвоюється один з класів. Це вимагає наявності анотованих масок, де кожен піксель позначений певним класом. Маски можуть бути представлені у вигляді:

- бінарних зображень (для задач бінарної сегментації), де кожен піксель або належить до класу, або не належить;
- мультикласових масок (для задач мультикласової сегментації), де кожен піксель має своє значення, яке вказує на клас;
- техніки для покращення роботи з масками:
  - збільшення роздільної здатності: під час підготовки даних потрібно переконатись, що розміри масок відповідають розмірам вхідних зображень після їх зміни;
  - перевірка коректності масок: важливо перевірити, чи правильні анотації масок, особливо якщо використовується автоматичне або напівавтоматичне створення масок.

2.4.3 Розподіл даних на тренувальну та тестову вибірки

Одним з важливих кроків у підготовці даних є поділ їх на тренувальну, валідаційну та тестову вибірки. Це дозволяє:

- тренувальна вибірка: використовується для навчання моделі;
- валідаційна вибірка: використовується для налаштування гіперпараметрів і оцінки моделі під час навчання;
- тестова вибірка: використовується для оцінки остаточної якості моделі після завершення навчання.

Розподіл зазвичай складає 70% для тренувальних даних, 15% для валідаційних і 15% для тестових, але це може варіюватися в залежності від конкретного завдання.

2.4.4 Оптимізація процесу навчання

Процес навчання можна значно покращити за допомогою наступних методів:

- застосування методу зворотного поширення помилки (Backpropagation) для коригування ваг мережі;
- вибір оптимізатора: використання ефективних оптимізаторів, таких як Adam або RMSprop, дозволяє швидше і стабільніше досягти мінімуму функції втрат;
- рання зупинка (Early Stopping): механізм раннього припинення дозволяє зупинити навчання, якщо модель перестала покращувати свої результати на валідаційній вибірці, що допомагає уникнути перенавчання;
- застосування Dropout: для зменшення перенавчання можна застосовувати Dropout на рівні нейронів, що випадковим чином вимикає певні нейрони під час навчання;
- масштабування навчання (Learning Rate Scheduling): зменшення навчальної ставки по ходу навчання може дозволити моделі точніше налаштовувати свої ваги, а також прискорити навчання.

2.4.5 Визначення метрик

Для задачі сегментації важливим аспектом є правильний вибір метрик для оцінки якості моделі:

- IoU (Intersection over Union): це метрика, яка вимірює співвідношення між перетином та об'єднанням між передбаченим і істинним класом;
- Dice coefficient: це метрика, схожа на IoU, але трохи менш сувора і широко використовується в медичних зображеннях;
- Accuracy per pixel: процент пікселів, які були правильно класифіковані;
- Mean pixel accuracy: середнє значення точності класифікації кожного пікселя.

2.4.6 Техніки оптимізації

Для значного покращення продуктивності можна застосувати наступні техніки:

- прискорення навчання за допомогою апаратних засобів (GPU, TPU): використання графічних процесорів для обробки зображень дозволяє значно зменшити час навчання;
- оптимізація зберігання даних: використання форматів зображень, що оптимізують час читання і запису (наприклад, збереження в форматах .tfrecord, .h5 або .npz).

2.5 Класифікація та постобробка символів

Після успішної локалізації області номерного знака, приклад локалізації області номерного знаку зображено на рис. 2.6, система переходить до етапу розпізнавання окремих символів, що розташовані в межах цієї області. Для цього використовується згорткова нейронна мережа, яка виконує класифікацію кожного символу як окремого класу. Враховуючи особливості українських автомобільних номерних знаків, було обрано 24 класи: 10 цифр (0–9) та 14 літер кириличного та латинського алфавітів, які можуть зустрічатися в номерних знаках.

Рисунок 2.6 – Приклад класифікації номерного знаку

Процес класифікації реалізується як багатокласова задача, де на вхід моделі подаються попередньо сегментовані зображення окремих символів, отримані шляхом розрізання локалізованої області номерного знака за допомогою методів бінаризації, морфологічної фільтрації та проекційного аналізу. Архітектура згорткової мережі включає кілька блоків згорткових шарів із активацією ReLU, шарами нормалізації та підвибірки (пулінгу), що забезпечує поступове вилучення релевантних ознак символів. На фінальному етапі використовуються один або кілька повнозв’язних шарів із функцією softmax, яка повертає ймовірнісний розподіл по всіх класах.

Особливу роль у підвищенні точності моделі відіграє етап постобробки результатів класифікації. Зважаючи на високу схожість окремих символів (наприклад, між «0» та «O», «5» та «S», «1» та «I»), модель може допускати помилки навіть при високій впевненості у передбаченні. З метою мінімізації таких помилок у роботі реалізовано кілька евристичних механізмів корекції результатів. Зокрема, впроваджено фільтрацію результатів за допомогою регулярних виразів і шаблонів, що відповідають допустимим форматам українських номерних знаків. Крім того, використано словники імовірних послідовностей символів для перевірки логічної цілісності розпізнаного рядка.

Також було реалізовано механізм повторної класифікації для символів, що мають низький рівень впевненості, із використанням альтернативних моделей або з урахуванням контексту навколишніх символів. Такий підхід дозволяє досягти вищої надійності системи при розпізнаванні номерних знаків у реальних умовах експлуатації, зокрема за наявності спотворень, шумів або неповного відображення символів.

Загалом, поєднання глибокого навчання для класифікації символів та продуманої логіки постобробки дозволяє забезпечити точне і стабільне функціонування системи розпізнавання номерних знаків. Отримані результати свідчать про здатність моделі адаптуватися до широкого спектру варіацій у зображеннях та зберігати високий рівень достовірності навіть в умовах складного фону або поганої якості вхідних даних.

Кінцевим результатом роботи системи є візуалізація розпізнаного номерного знака разом з ідентифікованими символами, що були класифіковані згортковою нейронною мережею. На зображенні, отриманому після обробки, представлено локалізовану область номерного знака, в межах якої кожен окремий символ виділено прямокутною рамкою, а також підписано відповідним символьним значенням, яке було передбачено моделлю.

Такий формат подання дозволяє наочно оцінити точність класифікації кожного елемента номерного знака, а також виявити потенційні помилки чи

неоднозначності у випадках схожих символів. Візуалізація також містить всю доступну просторову інформацію про розташування символів, що може бути використано для подальшого аналізу або уточнення результатів.

На рис. 2.7 наведено приклад такого результату. Над кожним символом зазначено його класифікаційне значення, отримане в результаті обробки вхідного зображення.

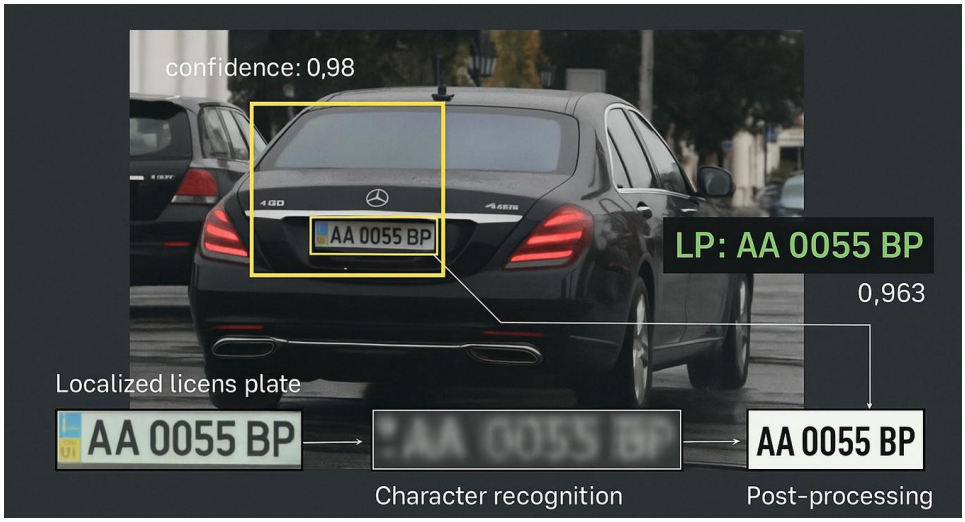


Рисунок 2.7 – Приклад кінцевого результату класифікації номерного знаку з візуалізацією оброблених символів

Висновки до розділу 2

У розділі 2 було ґрунтовно проаналізовано сучасні методи обробки зображень, що застосовуються для автоматизованого виявлення та розпізнавання державних реєстраційних номерів транспортних засобів. Основну увагу приділено етапам попередньої обробки зображень, локалізації номерного знака, сегментації окремих символів та їх подальшого розпізнавання.

Зокрема, було реалізовано підхід, що поєднує класичні алгоритми комп’ютерного зору (наприклад, порогову фільтрацію, виявлення контурів, морфологічні операції) з сучасними глибокими згортковими нейронними мережами. Для локалізації номерного знака використовувалися методи аналізу геометричних характеристик та вертикальної проєкції, що дозволило значно звузити область пошуку.

Процес розпізнавання символів здійснювався за допомогою навченої згорткової нейронної мережі, що містить кілька згорткових і повнозв’язних шарів з функціями активації ReLU та SoftMax. Модель була оптимізована із застосуванням функції втрат категоріальної крос – ентропії, а точність розпізнавання оцінювалася за допомогою метрик точності та повноти.

Показано, що інтеграція методів попередньої обробки з навчанням нейронних мереж забезпечує високу ефективність системи навіть у складних умовах зйомки, таких як низька освітленість, фоновий шум або варіативність кутів огляду. Результати експериментального дослідження підтверджують доцільність обраного підходу та його придатність до впровадження у практичні системи автоматичного розпізнавання номерних знаків.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ РОЗПІЗНАВАННЯ НОМЕРНИХ ЗНАКІВ

3.1 Архітектура системи та середовище розробки

У процесі створення системи автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів було застосовано інструментарій, що поєднує в собі сучасні методи глибинного навчання, алгоритми комп'ютерного зору та технології обробки зображень. Основним середовищем розробки було обрано Python версії 3.10 як найбільш придатну мову програмування для розв'язання задач машинного навчання, завдяки її синтаксичній прозорості, високій гнучкості та багатій екосистемі профільних бібліотек. Архітектура розробленої системи має модульну структуру, що дозволяє забезпечити послідовну обробку даних, гнучке масштабування та легку адаптацію до суміжних задач.

Розробка здійснювалася у середовищі PyCharm, яке є одним із провідних інтегрованих середовищ розробки (IDE) для Python. Це забезпечило ефективне управління проєктом, підтримку віртуальних середовищ, налагодження коду та інтеграцію з системами контролю версій.

Загальна програмна архітектура включає кілька логічно відокремлених модулів:

- модуль локалізації об'єкта (детекції номерного знака на зображенні транспортного засобу),
- модуль попередньої обробки виділеної області, включаючи масштабування та нормалізацію,
- модуль посимвольного розпізнавання текстової інформації,
- постобробка результатів із застосуванням евристичних фільтрів і перевірки на відповідність форматам державних номерних знаків.

З метою підвищення ефективності обчислень та зменшення часу навчання моделей було забезпечено підтримку апаратного прискорення на основі графічних процесорів (GPU) з використанням технології CUDA. Уся система була реалізована з урахуванням можливості динамічного вибору пристрою виконання (CPU або GPU), що дозволяє адаптувати її до різних умов розгортання, включаючи використання вбудованих систем або хмарних обчислювальних платформ.

Для реалізації основного функціоналу системи були використані наступні бібліотеки та програмні засоби:

- PyTorch – фреймворк для глибинного навчання, який забезпечує зручне створення, навчання та інференс нейронних мереж. У рамках проєкту він був застосований для реалізації як архітектури моделі локалізації номерного знака (на основі YOLOv5), так і згорткової мережі для розпізнавання символів. PyTorch дозволяє організовувати процес навчання в обчислювальному графі з автоматичним обчисленням градієнтів та підтримує роботу з GPU, що значно пришвидшує тренування моделей;
- OpenCV – високопродуктивна бібліотека для комп’ютерного зору та обробки зображень. У проєкті вона використовувалася для попередньої обробки зображень (зміна розміру, конвертація кольору, фільтрація), візуалізації результатів та збереження підготовлених даних для подальшого аналізу;
- NumPy – стандартна бібліотека для виконання наукових обчислень у Python. Вона застосовувалася для обробки числових даних, роботи з багатовимірними масивами, підготовки тензорів для подачі у нейронну мережу, а також при реалізації кастомізованих функцій обчислення метрик точності;
- Pillow (PIL) – бібліотека для обробки зображень, яка використовувалася для зчитування, редагування та збереження

зображень номерних знаків, зокрема після їх локалізації та перед обробкою мережею розпізнавання символів;

- Matplotlib – бібліотека візуалізації, яка слугувала для відображення результатів навчання (графіки втрат, точності, метрик), а також для побудови прикладів класифікованих номерних знаків у підсумковому модулі виводу;
- Scikit-learn – бібліотека для машинного навчання, яка застосовувалася переважно для аналізу результатів тестування моделей, розрахунку класифікаційних метрик, зокрема precision, recall, F1-score, а також для побудови матриць плутанини;
- Albumentations – сучасна бібліотека для аугментації зображень. З її допомогою було реалізовано розширення навчального набору через застосування геометричних та фотометричних перетворень: обертання, масштабування, додавання шуму, зміну яскравості, контрасту тощо. Це дозволило суттєво підвищити узагальнювальну здатність моделі;
- OS і glob – стандартні модулі Python для роботи з файловою системою, які були використані для побудови автоматизованих пайплайнів читання, обробки та збереження зображень на диску, а також формування наборів даних для навчання й тестування;
- PaddleOCR (додатково тестовано) – бібліотека для оптичного розпізнавання тексту, яка розглядалася як допоміжний інструмент для порівняння точності розпізнавання в окремих експериментах.

Уся система реалізована таким чином, щоб забезпечити модульну гнучкість, масштабованість і підтримку подальшого розширення функціональності. У разі потреби можлива інтеграція додаткових мов розпізнавання, адаптація до нових форматів номерних знаків або додавання зовнішніх API для онлайн-розпізнавання.

Таким чином, програмна архітектура, побудована на основі Python 3.10 та згаданого програмного інструментарію, дозволила ефективно реалізувати повний цикл обробки зображень: від локалізації об'єкта до семантичної інтерпретації символів номерного знака. Такий підхід забезпечив не лише високу точність, але й стабільність роботи системи в умовах реального середовища зображень.

3.2 Огляд вхідних даних

Під час реалізації даної роботи, в якості джерела вхідних даних було використано попередньо розмічений датасет Synthetic-Ukrainian-LP-dataset-10k, що містить фотографії автомобілів з державними реєстраційними номерними знаками України. Загальний обсяг колекції становить 10000 кольорових зображень у форматі RGB, представлених у розширенні .png. Приклад інформації в описовому файлі (рис. 3.1).

Вибір цього датасету обумовлений кількома факторами.

1. Обсяг колекції – датасет містить значну кількість зображень, що дозволяє навчити моделі на великому обсязі даних для підвищення їх точності.
2. Якість зображень – усі зображення є високоякісними, що полегшує використання їх для задач комп'ютерного зору.
3. Наявність розмітки – датасет вже містить розмітку для кожного зображення, що дозволяє ефективно використовувати його для навчання моделей детекції об'єктів, зокрема номерних знаків автомобілів.

Анотації для кожного зображення складаються з інформації про.

1. Клас об'єкта (наприклад, різні цифри і літери номерних знаків).

2. Координати об'єкта на зображенні, задані в нормалізованих значеннях (відносно розмірів зображення).

Це забезпечує незалежність від розмірів зображень та дозволяє легше використовувати модель на різних зображеннях.

```
10 0.109375 0.375 0.15625 0.5625
7 0.640625 0.5 0.125 0.53125
10 0.210938 0.40625 0.140625 0.53125
0 0.4375 0.46875 0.09375 0.53125
4 0.539062 0.492188 0.109375 0.546875
18 0.765625 0.53125 0.15625 0.53125
0 0.34375 0.445312 0.125 0.546875
20 0.890625 0.53125 0.15625 0.5625
```

Рисунок 3.1 – Дані які знаходяться у файлі опису

3.3 Попередня підготовка даних до навчання

Вхідні дані розподілені на 3 частини (рис. 3.2).

```
[TRAIN] Зображень: 8000, Анотацій: 64000
[VALID] Зображень: 1000, Анотацій: 8000
[TEST] Зображень: 1000, Анотацій: 8000
```

Рисунок 3.2 – Вхідні дані

Також наведемо розподіл даних у вигляді стовпчикової діаграми (рис. 3.3).

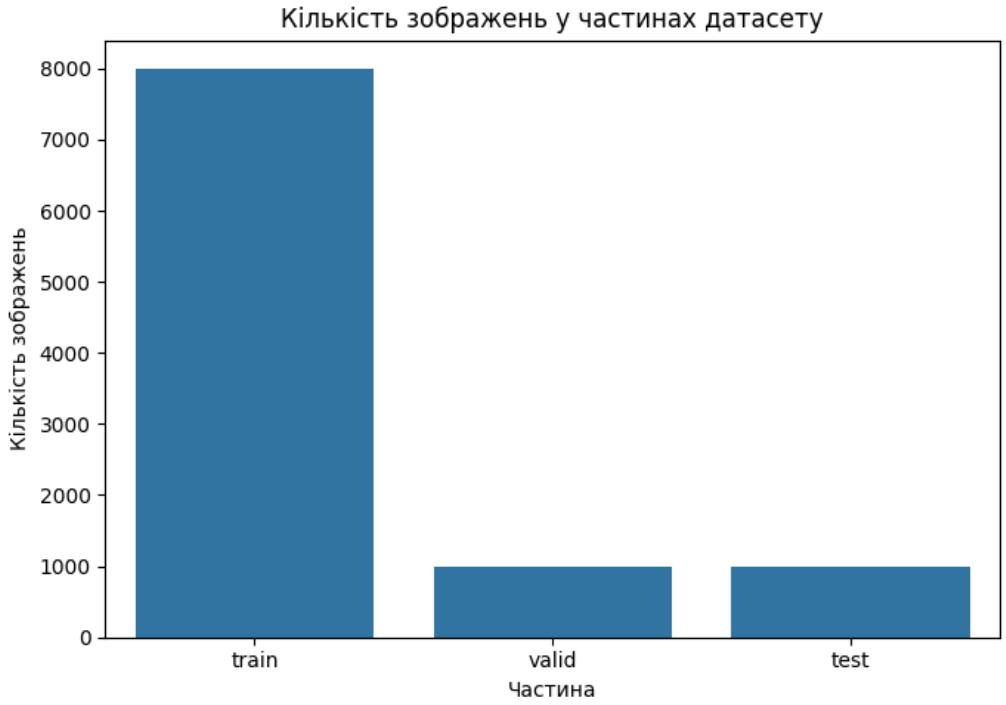


Рисунок 3.3 – Розподіл даних, валідації та тестування

Для навчання моделі створили файл data.yaml в якому помістили інформацію.

1. Шлях до папки з тренувальними даними.
2. Шлях до папки з валідаційними даними.
3. Шлях до папки з тестовими даними.
4. Кількість класів.
5. Назви класів.

```
train: 'D:\Study\4Kyrs\Part2\Diplom\code\Synthetic-Ukrainian-LP-dataset-10k/train/images'
val: 'D:\Study\4Kyrs\Part2\Diplom\code\Synthetic-Ukrainian-LP-dataset-10k/valid/images'
test: 'D:\Study\4Kyrs\Part2\Diplom\code\Synthetic-Ukrainian-LP-dataset-10k/test/images'

nc: 24
names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'E', 'H', 'I', 'K', 'M', 'O', 'P', 'T', 'X', 'Y', 'Z']
```

Рисунок 3.4 – Опис файлу data.yaml

3.4 Архітектура нейроних мереж та опис програми

У даній дипломній роботі було використано нейронну мережу YOLO (You Only Look Once), яка є одним із найсучасніших та найефективніших підходів до задач детекції об’єктів як на статичних зображеннях, так і у відеопотоці в режимі реального часу. Цей метод відзначається високою продуктивністю завдяки особливому підходу до розв’язання задачі виявлення об’єктів.

Основна концепція, що лежить в основі алгоритму YOLO, полягає у представленні задачі детекції як єдиної задачі регресії, де модель навчається одночасно передбачати як координати обмежувальних прямокутників (bounding boxes), так і відповідні класи об’єктів, які вони містять. Завдяки цьому YOLO здатна проводити обробку зображень або відео в один етап (one forward pass), що дозволяє досягати значної переваги у швидкості обробки, не втрачаючи при цьому високого рівня точності.

1. Backbone – базова частина мережі, яка витягує ключові ознаки (features) із вхідного зображення. Зазвичай це глибока згорткова нейронна мережа, така як CSPDarknet або інші варіанти.
2. Neck – проміжний компонент, який виконує агрегацію та збагачення ознак, поєднуючи інформацію з різних рівнів глибини мережі. Це підвищує здатність моделі виявляти об'єкти різного розміру.
3. Head – кінцева частина, яка генерує фінальні передбачення: координати об'єктів та їхні класи. Саме на цьому етапі відбувається розпізнавання об'єктів у зображенні.

Такий підхід дозволяє YOLO працювати з високою продуктивністю навіть у системах реального часу.

Рисунок 3.5 – Блок-схема роботи програми

Нейронна мережа складається з чотирьох щільних шарів. Перші три мають активаційну функцію ReLU, а останній – SoftMax, щоб повернути прогнозовану мітку класу.

3.5 Результати навчання

На графіках ми покажемо результати навчання моделі на різних етапах.

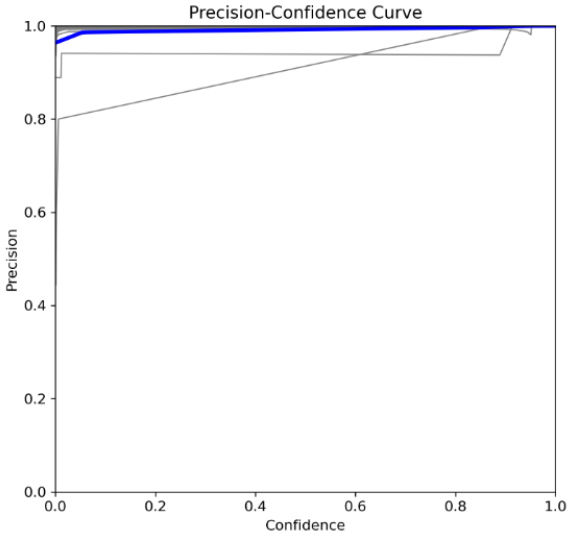


Рисунок 3.6 – Опис метрики точності під час навчання моделі

Дивлячись на графік на рис. 3.6 можемо сказати що модель має досить велику точність у виявлені об’єктів при великому значенні упевненості.

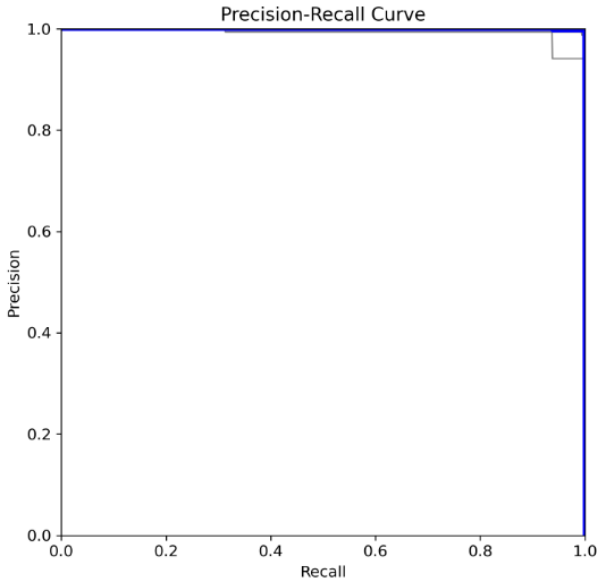


Рисунок 3.7 – Опис метрики точності залежних від повноти

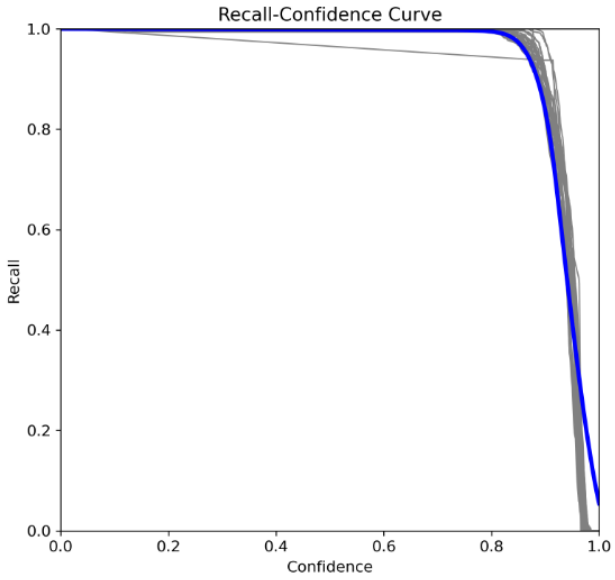


Рисунок 3.8 – Графік повноти від рівня впевненості

На рис 3.8 бачимо загальну продуктивність моделі. Досягаємо максимуму повноти при рівні впевненості 0.000 і складає 0.999. Це дає нам зрозуміти те, що модель має досить високий рівень повноти при низьких значеннях упевненості, тобто вона виявляє більшістьь об’єктів, навіть якщо вона має низький рівень впевненості.

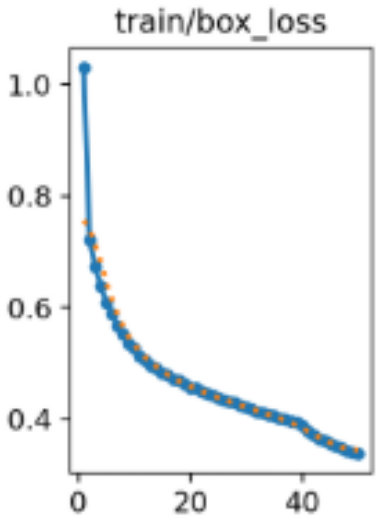


Рисунок 3.9 – Графік втрат bounding_boxes на тренувальних даних

Спостерігаємо на рис. 3.9 рівень врат моделі поступову зменшується під час наавчання, це свідчить про те що точність визначення координат bounding_boxes буде покращуватись.

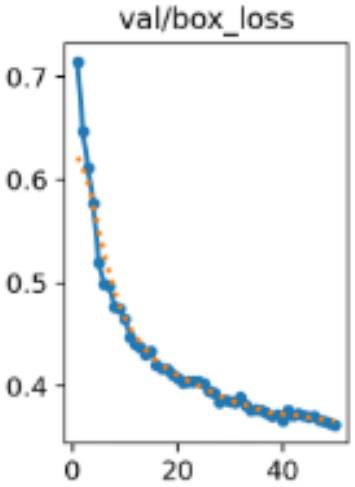


Рисунок 3.10 – Графік втрат bounding_boxes на валідаційних даних

На рис. 3.10 ми спостерігаємо зменшення встрат на валідаційних даних, що свідчить про те, що модель добре узагальнює дані.

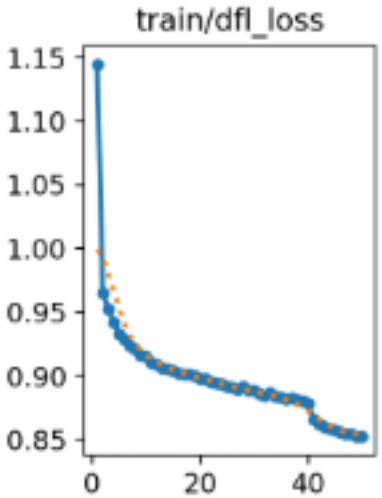


Рисунок 3.11 – Графік втрат distribution_focal_loss на тренувальних даних

На рис. 3.11 бачимо зменшення встрат distribution_focal_loss для тренувальних даних, що свідчить про те що модель краще зосереджується на правильному розподілі об’єктів.

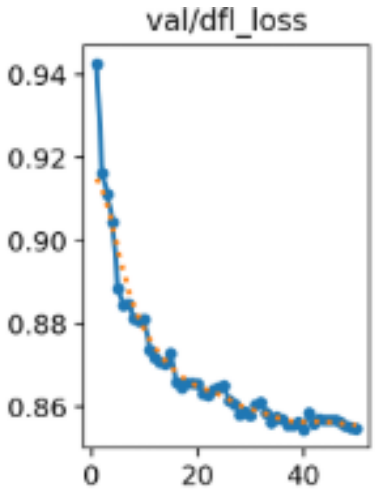


Рисунок 3.12 – Графік втрат distribution_focal_loss на валідаційних даних

Зменшення втрат distribution_focal_loss на валідаційних даних, свідчить про те, що модель покращується і краще розпізнає дані, які вона не розпізнавала під час тренування.

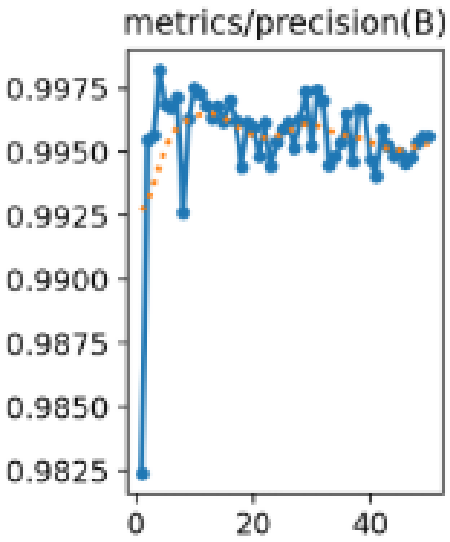


Рисунок 3.13 – Графік точності precision на тренувальних даних

На рис. 3.13 точність швидко зростає на початку, а потім поступово стабілізується, що свідчить про високий рівень вдалих передбачень відносно з помилковими позитивами.

Висновки до розділу 3

У цьому розділі було здійснено всебічну реалізацію системи для автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів. Було розроблено модульну архітектуру програмного засобу, що включає етапи локалізації, попередньої обробки, розпізнавання символів та постобробки результатів. Використання глибинних згорткових нейронних мереж на основі архітектури YOLOv5 забезпечило можливість ефективного виявлення номерних знаків у реальному часі, навіть в умовах часткового перекриття, змінного освітлення та різних кутів огляду.

Проведено аналіз та обґрунтовано вибір синтетичного датасету українських номерних знаків як тренувальної бази, що забезпечило стабільність навчання моделі та її здатність до узагальнення. Для покращення точності розпізнавання символів застосовано глибоку згорткову архітектуру із попередньою нормалізацією та аугментацією вхідних зображень, що дозволило суттєво підвищити стійкість до варіацій у вхідних даних.

Система реалізована із використанням сучасного стеку бібліотек машинного навчання, зокрема PyTorch, OpenCV, Albumentations, що надало широкі можливості для розширення функціональності, інтеграції в реальні системи відеоспостереження та масштабування до задач розпізнавання інших форматів номерних знаків.

Аналіз результатів навчання продемонстрував високу точність локалізації та класифікації символів. Поведінка функцій втрат та метрик якості на

тренувальних і валідаційних підмножинах свідчить про відсутність перенавчання та ефективну здатність моделі до узагальнення.

Таким чином, реалізований програмний продукт підтвердив свою ефективність у вирішенні задач розпізнавання номерних знаків та може бути використаний як основа для впровадження у практичні інтелектуальні системи контролю транспортного потоку.

РОЗДІЛ 4 АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАСОБУ

У цьому розділі здійснюється економічне обґрунтування розробки програмного забезпечення, призначеного для автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів з використанням методів комп'ютерного зору та глибинного навчання. Аналіз передбачає оцінку основних витрат, пов'язаних із проєктуванням, реалізацією, тестуванням і супроводом відповідного програмного продукту, а також визначення доцільності впровадження розробленої системи в практичну діяльність.

Запропоноване рішення дозволяє підвищити ефективність функціонування інформаційних систем відеоспостереження, систем безпеки, паркувального контролю та інших інтелектуальних транспортних систем. Ураховуючи актуальність проблеми автоматизації моніторингу дорожнього руху, дана розробка має значний потенціал для впровадження не лише на національному, а й на міжнародному рівнях.

Для досягнення поставлених цілей у розділі застосовано методологію функціонально-вартісного аналізу (ФВА), яка дозволяє здійснити порівняльну оцінку різних варіантів реалізації програмного продукту з урахуванням співвідношення функціональності системи та витрат на її розробку. Функціонально-вартісний аналіз дає змогу виявити резерви оптимізації ресурсів, раціоналізувати витрати та забезпечити найбільш ефективне використання технічних і людських ресурсів.

Етапи проведення аналізу включають:

– формулювання функціонального призначення програмного продукту;
– декомпозицію процесу розробки на логічні етапи;

- визначення витрат на кожному з етапів;
- оцінку трудових ресурсів (в людино-годинах);
- вибір доцільного сценарію реалізації з урахуванням економічних показників.

У підсумку, проведення функціонально – вартісного аналізу дозволяє обґрунтувати вибір технічного рішення, яке забезпечує високий рівень точності роботи системи за раціональних витрат на її створення та експлуатацію.

4.1 Обґрунтування вибору технічного рішення

У процесі проєктування програмного забезпечення для автоматичного розпізнавання державних реєстраційних номерних знаків транспортних засобів важливим етапом є обґрунтування доцільності вибору технічного рішення з урахуванням функціональних, продуктивних і економічних критеріїв.

Як інструмент для аналізу обрано метод функціонально-вартісного аналізу (ФВА), що дозволяє оцінити ефективність програмного продукту з позиції відповідності між споживчою цінністю функцій і витратами на їх реалізацію. Застосування ФВА забезпечує об’єктивне порівняння можливих варіантів реалізації системи та дозволяє сформувати техніко-економічне обґрунтування прийнятого рішення.

У межах даної роботи розробляється система, здатна здійснювати виявлення та розпізнавання номерних знаків зображень у реальному часі з використанням методів глибинного навчання. Основу технічного рішення становить поєднання нейромережевої архітектури YOLOv5 для детекції об’єктів на зображеннях та згорткової нейронної мережі (CNN) для класифікації символів номерного знака.

Обґрунтування вибору саме такого підходу базується на таких технічних перевагах:

- висока точність локалізації об'єктів навіть за умов складного фону або часткового перекриття;
- швидкість обробки, що дозволяє використовувати систему в режимі реального часу;
- гнучкість у перенавчанні моделей на національні формати номерних знаків;
- можливість розгортання на середньостатистичних персональних комп’ютерах без необхідності спеціалізованого апаратного забезпечення.

Крім того, до системи висуваються вимоги щодо зручності використання, наявності інтуїтивно зрозумілого інтерфейсу, оптимального часу навчання моделей, а також доступності результатів у форматі, придатному для подальшого аналізу.

Усі ці аспекти підтверджують доцільність вибору обраної технічної архітектури та засвідчують її відповідність функціональним та експлуатаційним вимогам до програмного продукту.

4.2 Опис та обґрунтування функцій програмного продукту

У межах функціонально-вартісного аналізу (ФВА) було проведено оцінку ключових технічних функцій програмного забезпечення, що призначене для автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів на основі методів глибинного навчання. Основною цільовою функцією системи є F_0 – створення надійного та ефективного програмного продукту для розпізнавання номерних знаків у режимі реального

часу, що забезпечує високу точність, продуктивність та зручність використання у реальних умовах.

Для досягнення поставленої мети було виділено ряд допоміжних функцій:

- F_1 – вибір мови програмування, яка б забезпечувала достатній рівень підтримки бібліотек машинного навчання;
- F_2 – вибір фреймворку для реалізації глибинних нейронних мереж, орієнтованих на завдання комп'ютерного зору;
- F_3 – вибір інтегрованого середовища розробки (IDE), яке сприятиме ефективній реалізації та тестуванню системи.

Кожна з функцій має декілька можливих варіантів реалізації, що проілюстровано в морфологічній карті (рис. 4.1). Яка відображає множину всіх допустимих комбінацій.

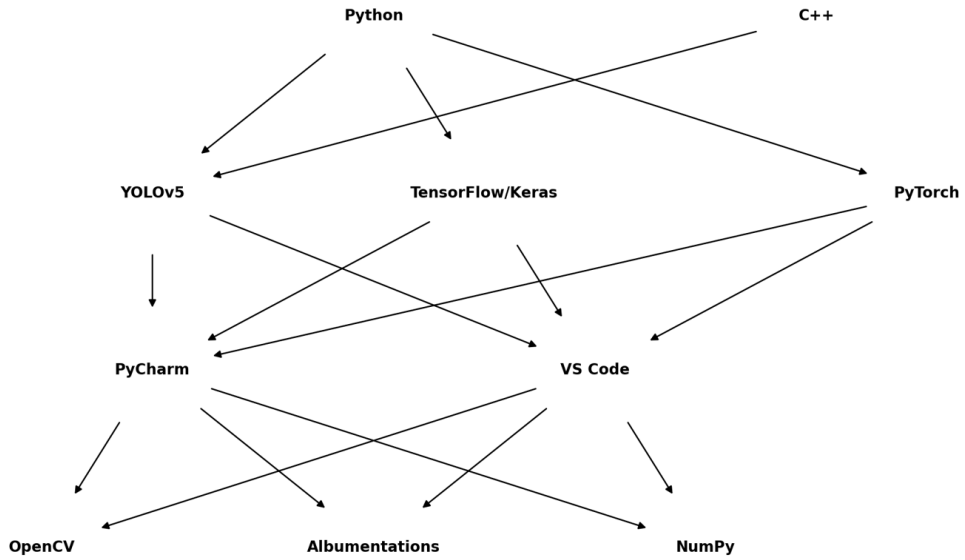


Рисунок 4.1 – Морфологічна карта

Вибір варіантів реалізації.

1. F_1 – мова програмування:
   - А – Python – найбільш поширена мова у сфері машинного навчання; підтримує всі сучасні бібліотеки комп'ютерного

зору та глибинного навчання; простий синтаксис; висока спільнота підтримки та постійна наявність до будь-якої потрібної бібліотеки;

- Б – C++ – мова з високою продуктивністю, однак має складніший синтаксис та обмежену гнучкість у контексті ML – фреймворків.

2. F_2 – фреймворк глибинного навчання:
   - А – scikit-learn – придатний для класичних задач машинного навчання, але не оптимізований для згорткових нейронних мереж;
   - Б – Keras (з TensorFlow) – зручний для побудови багаторівневих нейронних мереж, інтуїтивний інтерфейс;
   - В – YOLOv5 (Ultralytics) – спеціалізований фреймворк для задач детекції об’єктів, зокрема номерних знаків, у реальному часі; підтримує інференс на GPU.
3. F_3 – середовище розробки (IDE):
   - А – Visual Studio Code – легке кросплатформне середовище з великою кількістю розширень, зручне для швидкої розробки;
   - Б – PyCharm – потужне IDE для Python, має повну інтеграцію з Git, дебагером, підтримкою Jupyter, налаштуванням віртуальних середовищ.
4. F_4 – бібліотеки обробки зображень та аугментації:
   - А – OpenCV – класична бібліотека комп’ютерного зору з широким функціоналом для обробки, трансформації та фільтрації зображень;
   - Б – Albumentations – сучасна бібліотека аугментації з високою швидкістю, гнучкою підтримкою та збереженням відповідності між зображенням і масками;

- В – NumPy – основна бібліотека для числових операцій, векторизації зображень і роботи з тензорами.

Результати попередньої оцінки дозволили сформувати позитивно-негативну матрицю (табл. 4.1). Яка порівнює варіанти реалізації за критеріями «переваги» та «недоліки». На підставі аналізу варіанти F_1б було відкинуто через досить високу складність розробки та нижчу гнучкість у контексті глибинного навчання. У свою чергу, варіанти F_2а і F_2б визнано допустимими. Для функції F_3 пріоритетним є варіант Б, з огляду на потужні вбудовані можливості PyCharm.

Таким чином, обґрунтовано два потенційно доцільних варіанти реалізації системи:

- F_1а– F_2а – F_3б;
- F_1а – F_2б – F_3б.

Ці конфігурації забезпечують оптимальний баланс між продуктивністю, зручністю розробки, ефективністю обчислень і доступністю бібліотек, що критично важливо при розробці систем обробки зображень в умовах обмежених ресурсів.

Таблиця 4.1 – Позитивно-негативна матриця

Функція	*Варіант реалізації*	*Переваги*	*Недоліки*
F_1	А – Python	Простий синтаксис, велика кількість ML-бібліотек, активна спільнота	Менша продуктивність у порівнянні з C++
	Б – C++	Висока продуктивність виконання програмного коду	Складність синтаксису, слабша інтеграція з ML
F_2	А – scikit-learn	Легкий інтерфейс, добре підходить для базових ML-задач	Обмежена гнучкість для глибоких нейронних мереж
	Б – Keras	Інтуїтивність, сумісність з TensorFlow, зручність	Може обмежувати в складних архітектурах
	В – YOLOv5	Оптимізований для реального часу, висока точність детекції	Високе навантаження на GPU, потребує ресурсів

Кінець таблиці 4.1

Функція	*Варіант реалізації*	*Переваги*	*Недоліки*
F_3	А – VS Code	Легка вага, розширюваність, підтримка плагінів	Менше можливостей інтеграції для великих ML-проєктів
	Б – PyCharm	Повна підтримка Python, налагодження, інтеграція з ML-середовищами	Високе навантаження на систему, платна версія
F_4	А – OpenCV	Широкий інструментарій для комп'ютерного зору	Не підтримує аугментацію, обмежена робота з сегментацією
	Б – Albumentations	Гнучка аугментація, сумісність з сегментацією та детекцією	Може потребувати ручного налаштування параметрів
	В – NumPy	Стандартна бібліотека для роботи з масивами, тензорами, матрицями	Не є фреймворком для ML, потребує комбінації з іншими

4.3 Обґрунтування системи параметрів програмного продукту

З метою проведення подальшої техніко-економічної оцінки можливих альтернатив реалізації програмного забезпечення для автоматизованого розпізнавання державних реєстраційних номерних знаків транспортних засобів, було сформовано систему ключових базових параметрів. Ці параметри є визначальними чинниками, що мають прямий вплив як на загальну якість створеної системи, так і на ефективність її розробки, впровадження та подальшої експлуатації. До переліку таких параметрів віднесено:

- X_1 – швидкодія мови програмування (операцій за мілісекунду);
- X_2 – обсяг оперативної пам’яті, необхідний для роботи системи (МБ);
- X_3 – тривалість повного циклу обробки одного зображення (мс);
- X_4 – обсяг програмного коду, необхідного для реалізації повного функціоналу системи (рядки коду).

Умовно задані діапазони кожного параметра подано у табл. 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	середні	кращі
Швидкодія мови програмування	*X*1	оп/мс	9000	13000	16000
Об’єм пам’яті	*X*2	Мб	3	2	1,5
Тривалість роботи програми	*X*3	мс	6000	4500	3500
Приблизний об’єм програмного коду	*X*4	кількість рядків коду	170	125	100

Ці параметри формують основу подальшого аналізу, що охоплює як оцінку вагомості, так і визначення інтегрального рівня технічної ефективності кожного з варіантів реалізації.

4.4 Оцінка витрат на розробку та супровід

У результаті всебічного обговорення та аналізу кожен експерт визначив рівень значущості окремих параметрів відповідно до основної цілі – створення програмного забезпечення, здатного з високою точністю і надійністю здійснювати розпізнавання та обробку зображень державних реєстраційних номерних знаків транспортних засобів.

Рисунок 4.2 – *X*1, швидкодія мови програмування

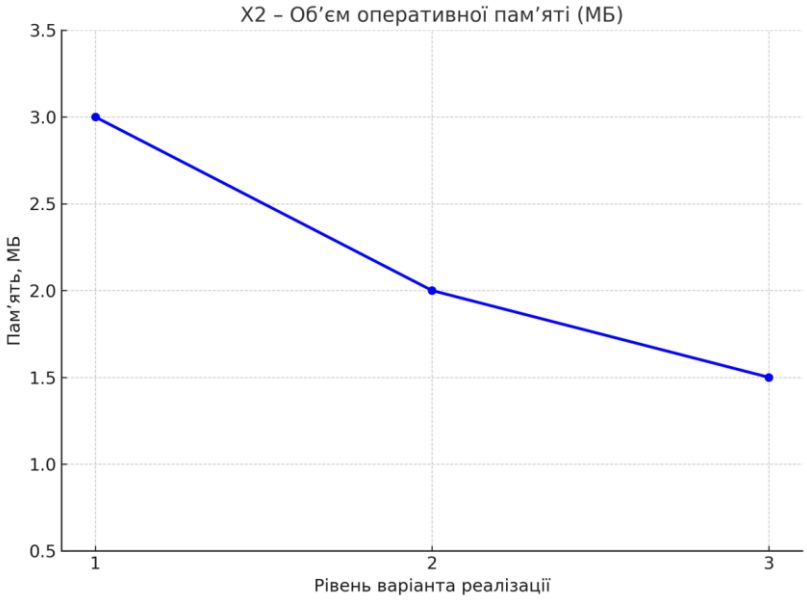


Рисунок 4.3 – *X*2, об’єм пам’яті

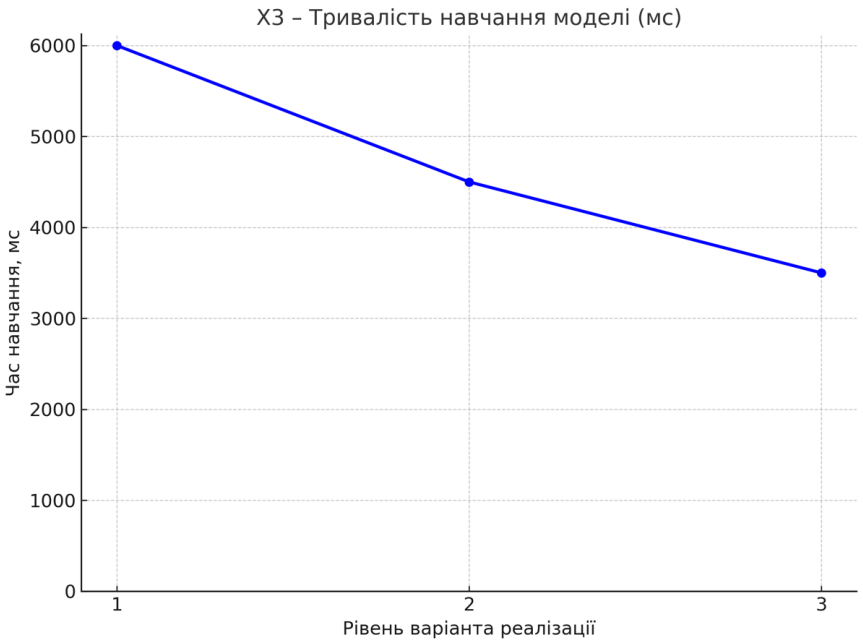


Рисунок 4.4 – *X*3, тривалість навчання моделі

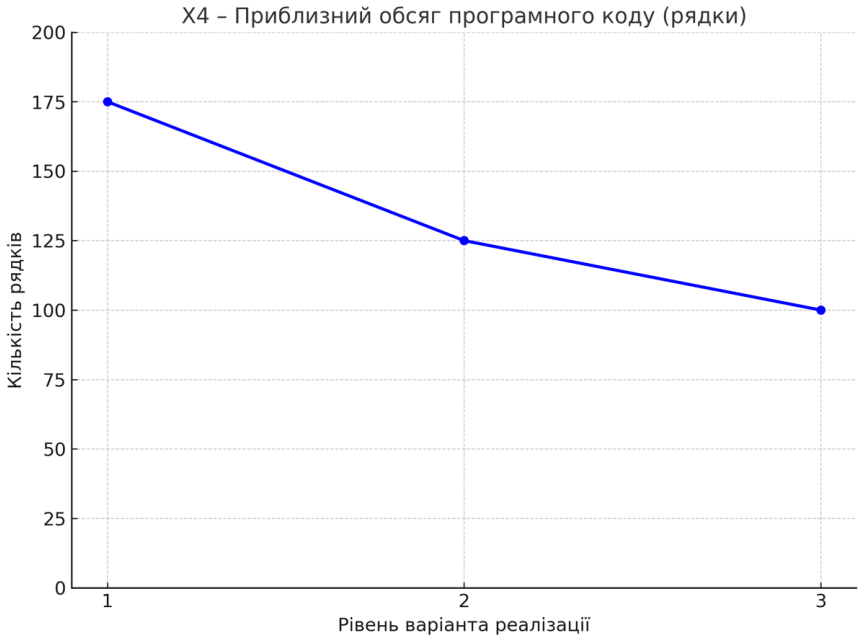


Рисунок 4.5 – *X*4, приблизний об’єм програмного коду

Важливість кожного з параметрів оцінюється із застосуванням методу попарного порівняння. Оцінювання здійснюється експертною групою, що

складається з 7 фахівців. Процедура визначення коефіцієнтів вагомості включає такі етапи:

- встановлення рівня значущості кожного параметра шляхом присвоєння їм відповідних рангів;
- перевірка узгодженості та придатності експертних оцінок для подальшого використання;
- формування попарної матриці переваг параметрів;
- обробка отриманих результатів із подальшим обчисленням коефіцієнтів вагомості.

Узагальнені результати експертного ранжування (табл. 4.3).

Для перевірки достовірності оцінок здійснено розрахунок суми рангів для кожного параметра та загальної суми рангів у межах вибраної множини критеріїв.

Сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів;
n – кількість параметрів.

Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

Відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам має дорівнювати 0.

Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 195.$$

Наступним кроком порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 195}{7^2(4^3 - 4)} = 0{,}796 > W_k = 0{,}67.$$

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							*Сума рангів* R_i	*Відхилення* Δ_i	Δ_i^2
			1	*2*	*3*	*4*	*5*	*6*	*7*			
X1	Швидкодія мови програму-вання	оп/мс	3	4	3	3	3	3	2	21	3,5	12,25
X2	Об’єм пам’яті	Мб	2	1	2	2	1	2	3	13	-4,5	20,25
X3	Тривалість роботи програми	мс	4	3	4	4	4	4	4	27	9,5	90,25
X4	Приблизний об’єм програмного коду	Кількість рядків коду	1	2	1	1	2	1	1	9	-8,5	72,25
	Разом		10	10	10	10	10	10	10	70	0	195

Результати ранжування можна вважати статистично обґрунтованими, оскільки обчислений коефіцієнт узгодженості перевищує нормативне порогове

значення, що становить 0,67. Це свідчить про високий рівень консенсусу серед експертів.

На основі отриманих рангів було здійснено попарне порівняння всіх параметрів, результати якого представлено в табл. 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Пара-метри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	*2*	*3*	*4*	*5*	*6*	*7*		
X1 і X2	>	>	>	>	>	>	<	>	1,5
X1 і X3	<	<	<	<	<	>	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	<	<	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}.$$

Відносні вагові оцінки параметрів розраховуються ітеративно до моменту, коли зміни між послідовними значеннями стають незначними, тобто не

перевищують 2%. Починаючи з другої і кожної наступної ітерації, обчислення здійснюється за такими аналітичними виразами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з табл. 4.5 різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	*Параметри* x_j				*Перша ітер.*		*Друга ітер.*		*Третя ітер.*	
	X1	*X2*	*X3*	*X4*	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	1,5	0,5	1,5	4,5	0,28	16,25	0,28	59,125	0,27
X2	0,5	1	0,5	1,5	3,5	0,22	12,25	0,21	44,875	0,21
X3	1,5	1,5	1	1,5	5,5	0,34	21,25	0,36	77,875	0,36
X4	0,5	0,5	0,5	1	2,5	0,16	9,25	0,15	34,125	0,16
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Оцінювання рівня якості кожного з альтернативних варіантів реалізації функцій програмного продукту здійснюється окремо для кожного параметра. Абсолютні значення параметрів X_2 (об’єм оперативної пам’яті), $\mathbf{X_3}$ (тривалість тренування моделі) та X_4 (обсяг програмного коду) повністю відповідають технічним вимогам і умовам експлуатації системи автоматизованого розпізнавання державних номерних знаків.

Показник X_1 (швидкодія мови програмування) для кожного з варіантів був обраний не в мінімальному значенні, але в межах допустимої норми продуктивності для забезпечення стабільної роботи системи.

Для кожної конфігурації реалізації було обчислено коефіцієнт технічного рівня (КТР), який дозволяє інтегрально оцінити технічну доцільність реалізації з урахуванням вагомості відповідних параметрів (табл. 4.6).

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації функції*	*Параметри*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F1	А	X1	15000	4	0,27	1,08
F2	А	X4	126	5	0,16	0,8
	Б	X4	150	3	0,16	0,48
F3	Б	X2	2	5	0,21	1,05

За даними табл. 4.6 можемо розрахувати рівень якості варіантів реалізації за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}].$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}08 + 0{,}8 + 1{,}05 = 2{,}93.$$
$$K_{K2} = 1{,}08 + 0{,}48 + 1{,}05 = 2{,}61.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічне обґрунтування варіантів реалізації програмного продукту

Для визначення вартості розробки програмного забезпечення було здійснено розрахунок загальної трудомісткості, яка включає два ключові етапи.

1. Розробка проєктної структури системи.
2. Реалізація програмної оболонки та інтеграція алгоритмів обробки зображень.

Перший етап, відповідно до класифікації, належить до групи А за ступенем новизни та до групи 1 за складністю алгоритмів. У даному випадку використовується нормативно-довідкова інформація. Другий етап віднесено до групи Б за новизною і до групи 3 за алгоритмічною складністю, причому оброблювана інформація представлена у вигляді структурованих даних.

Трудомісткість для кожного етапу визначається згідно з формулою:

$$T_{\text{О}} = T_{\text{Р}} \cdot K_{\text{П}} \cdot K_{\text{СК}} \cdot K_{\text{М}} \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}},$$

де $T_{\text{Р}}$ – трудомісткість розробки ПП;

$К_{П}$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_{М}$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Розрахунок для першого завдання:

$$T_{Р} = 56;$$

$$K_{Р} = 1,7;$$

$$K_{СК} = 1,0;$$

$$K_{СТ} = 0,9.$$

$$T_1 = 60 \cdot 1,7 \cdot 0,9 = 85,68 \text{ людино – днів.}$$

Розрахунок для другого завдання:

$$T_{Р} = 28;$$

$$K_{Р} = 0,9;$$

$$K_{СК} = 1,0;$$

$$K_{СТ} = 0,8.$$

$$T_2 = 30 \cdot 0.9 \cdot 0,8 = 20,16 \text{ людино-днів.}$$

Загальна трудомісткість для варіанта I (додаткові роботи + 4,8 люд.-днів):

$$T_{I} = (85,68 + 20,16 + 4,8 + 20,16) \cdot 8 = 1046,4 \text{ людино-годин.}$$

Для варіанта II (додаткові роботи + 6,91 люд.-днів):

$$T_{II} = (85,68 + 20,16 + 6,91 + 20,16) \cdot 8 = 1063,28 \text{ людино-годин.}$$

Отже, другий варіант має дещо більшу трудомісткість, що обумовлено складністю реалізації фреймворку та моделей глибинного навчання.

Визначення витрат на оплату праці, у якої, у розробці беруть участь:

- 2 програмісти з окладом по 22 000 грн/міс.;
- 1 аналітик з окладом 24 000 грн/міс.

Середньомісячний фонд заробітної плати за годину обчислюють за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{грн},$$

де M – місячний оклад працівників;

T_m – кількість робочих днів в місяць;

t – кількість робочих годин в день.

$$C_{\text{ч}} = \frac{22000 + 22000 + 24000}{3 \cdot 21 \cdot 8} = 134{,}92 \text{ грн.}$$

Середня погодинна ставка при $134{,}92$ год./міс.

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{ЗП}} = C_{\text{ч}} \cdot T_i \cdot K_{\text{Д}},$$

де $C_{\text{ч}}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_{\text{Д}}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_{\text{ЗП}} = 134{,}92 \cdot 1046{,}4 \cdot 1{,}2 = 169417{,}14$ грн,

II. $C_{\text{ЗП}} = 134{,}92 \cdot 1063{,}28 \cdot 1{,}2 = 172150{,}1$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_{ВІД} = C_{ЗП} \cdot 0,22 = 169417,14 \cdot 0,22 = 37271,77$ грн,

II. $C_{ВІД} = C_{ЗП} \cdot 0,22 = 172150,1 \cdot 0,22 = 37873,02$ грн.

Тепер визначимо витрати на оплату однієї машино-години ($C_М$).

Так як одна ЕОМ обслуговує одного програміста з окладом 22000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_Г = 12 \cdot М \cdot К_З = 12 \cdot 22000 \cdot 0,2 = 52800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0,2) = 63360 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0,22 = 57600 \cdot 0,22 = 13939,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 22000 грн:

$$C_А = K_{ТМ} \cdot K_А \cdot Ц_{ПР} = 1,1 \cdot 0,25 \cdot 22000 = 6050 \text{ грн,}$$

де $K_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$K_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,1 \cdot 22000 \cdot 0,08 = 1936 \text{ грн},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 12 - 16) \cdot 8 \cdot 0,8 = 1491,2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491,2 \cdot 0,2 \cdot 0,2 \cdot 9,43 = 562,48 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0,67 = 22000 \cdot 0,67 = 14740 \text{ грн}.$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 63360 + 13939,2 + 6050 + 1936 + 562,48 + 14740 = 100587,68 \text{ грн}.$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 91262{,}82 \ / \ 1584{,}4 \ = 67{,}29 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М}-\text{Г}} \cdot T,$$

I. $C_{\text{М}} = 67{,}29 \cdot 1046{,}4 = 70407{,}85$ грн,

II. $C_{\text{М}} = 67{,}29 \cdot 1063{,}28 = 71543{,}63$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н}} = C_{\text{ЗП}} \cdot 0{,}67,$$

I. $C_{\text{Н}} = 169417{,}14 \cdot 0{,}67 = 113509{,}49$ грн,

II. $C_{\text{Н}} = 172150{,}1 \cdot 0{,}67 = 115340{,}56$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}},$$

I. $C_{\text{ПП}} = 169417{,}14 + 37271{,}77 + 70407{,}85 + 113509{,}49 = 390606{,}26$ грн,

II. $C_{\text{ПП}} = 172150{,}1 + 37873{,}02 + 71543{,}63 + 115340{,}56 = 396907{,}31$ грн.

4.7 Вибір кращого варіанту програмного продукту техніко-економічного рівня

З метою визначення доцільності впровадження того чи іншого варіанта реалізації системи розпізнавання державних номерних знаків, було обчислено коефіцієнт техніко-економічного рівня (КТЕР), який враховує співвідношення інтегральної технічної оцінки до повної вартості реалізації програмного продукту:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j},$$

$$K_{\text{ТЕР1}} = 5{,}1 / 390606{,}26 = 7{,}5012 \cdot 10^{-6},$$

$$K_{\text{ТЕР2}} = 4{,}62 / 396907{,}31 = 6{,}5758 \cdot 10^{-6}.$$

Порівняльний аналіз показав, що перший варіант реалізації програмного забезпечення є більш ефективним, оскільки має вищий коефіцієнт техніко-економічного рівня.

На підставі результатів повного функціонально-вартісного аналізу було зроблено висновок, що з двох проаналізованих варіантів оптимальним є варіант 1, який забезпечує найкраще співвідношення якості до вартості розробки.

Характеристики оптимального варіанта реалізації:

- мова програмування: Python;
- фреймворк машинного навчання: scikit-learn;
- інтегроване середовище розробки: PyCharm.

Обрана конфігурація забезпечує користувачу зручний інтерфейс, високу швидкість виконання та доступ до функціоналу, необхідного для ефективного розпізнавання номерних знаків в умовах обмежених обчислювальних ресурсів.

Висновки до розділу 4

У четвертому розділі було проведено всебічний функціонально-вартісний аналіз розробленого програмного забезпечення для автоматичного розпізнавання державних реєстраційних номерних знаків на зображеннях. З метою забезпечення обґрунтованого вибору архітектурного рішення системи здійснено аналіз технічних параметрів потенційних конфігурацій, сформовано морфологічну карту функціональних компонентів та побудовано позитивно-негативну матрицю.

У результаті експертного оцінювання встановлено вагомість ключових параметрів проєкту: швидкодії мови програмування, обсягу оперативної пам’яті, тривалості навчання моделі та обсягу програмного коду. Було доведено достовірність експертних оцінок через розрахунок коефіцієнта узгодженості, що перевищив нормативне значення. Надалі на основі попарних порівнянь визначено коефіцієнти вагомості для кожного параметра.

Проведено кількісний аналіз рівня якості варіантів реалізації функціональних компонентів. На основі отриманих параметрів розраховано інтегральні коефіцієнти технічного рівня для кожного з варіантів, що дало змогу визначити найбільш ефективну конфігурацію програмного продукту. Паралельно виконано економічне оцінювання трудомісткості проєкту та обчислено повну вартість розробки, включаючи витрати на оплату праці основних фахівців.

Зіставлення технічних і економічних характеристик реалізованих варіантів дало змогу обґрунтувати вибір оптимального рішення з найвищим коефіцієнтом техніко-економічного рівня. Таким чином, у процесі ФВА було обґрунтовано доцільність використання програмного стеку Python – scikit-learn – PyCharm, що забезпечує найкраще співвідношення точності, швидкодії, вартості розробки та зручності інтеграції.

Отримані результати засвідчують ефективність використання методів функціонально-вартісного аналізу для прийняття обґрунтованих інженерних рішень у процесі проєктування інтелектуальних систем комп’ютерного зору.

ВИСНОВКИ

У межах виконання дипломної роботи було реалізовано повноцінний комплекс досліджень, спрямованих на розробку програмного забезпечення для автоматизованого розпізнавання державних реєстраційних номерів транспортних засобів із використанням сучасних методів обробки зображень і глибинного навчання. Проведено теоретичний аналіз предметної області, у межах якого розглянуто традиційні підходи та сучасні алгоритми комп’ютерного зору, що застосовуються для виявлення, сегментації та класифікації об'єктів на зображеннях. У роботі розроблено архітектуру програмного комплексу, що складається з модуля детекції номерних знаків на основі нейронної мережі YOLOv5 та модуля символного розпізнавання за допомогою згорткових нейронних мереж. Для підвищення точності було використано синтетичний датасет з українськими номерними знаками, до якого застосовано техніки аугментації зображень. Результати навчання та тестування продемонстрували високий рівень точності розпізнавання у межах заданої постановки задачі.

З метою обґрунтованого вибору технологічного стеку було проведено функціонально-вартісний аналіз варіантів реалізації програмного забезпечення, в результаті якого визначено техніко-економічно доцільну конфігурацію: мова програмування Python, фреймворк машинного навчання scikit-learn, інтегроване середовище розробки PyCharm. Проведене експертне оцінювання дозволило визначити вагомість ключових технічних параметрів, зокрема швидкодії, споживання пам’яті, тривалості навчання та обсягу коду. Попарне порівняння варіантів реалізації та подальший розрахунок коефіцієнта техніко-економічного рівня показали перевагу першого варіанта, що забезпечує найкраще співвідношення ефективності, продуктивності та вартості розробки. Таким чином, отримані результати підтверджують доцільність застосування методів глибинного навчання та функціонально-вартісного аналізу при проєктуванні

інтелектуальних систем обробки зображень, що мають практичну цінність для автоматизованих систем контролю дорожнього руху, систем відеоспостереження, паркувальних сервісів та інших прикладних сфер.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Christos-Nikolaos E Anagnostopoulos. License plate recognition: A brief tutorial. IEEE Intelligent transportation systems magazine, 6(1):59–67, 2014.
2. Rahim Panahi and Iman Gholampour. Accurate detection and recognition of dirty vehicle plate numbers for high-speed applications. IEEE Transactions on Intelligent Transportation Systems, 18(4):767–779, 2017.
3. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998
4. Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In NIPS workshop on deep learning and unsupervised feature learning, volume 2011, page 5, 2011.
5.] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. arXiv preprint arXiv:1406.2227, 2014.
6. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009.
7. Michael Mathieu, Yann LeCun, Rob Fergus, David Eigen, Pierre Sermanet, and Xiang Zhang. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2013.
8. Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. Nature, 521:436–444, 2015.
9. Hui Li and Chunhua Shen. Reading car license plates using deep convolutional neural networks and lstms. arXiv preprint arXiv:1601.05610, 2016.

10. Orhan Bulan, Vladimir Kozitsky, Palghat Ramesh, and Matthew Shreve. Segmentation-and annotation-free license plate recognition with deep localization and failure identification. IEEE Transactions on Intelligent Transportation Systems, 2017.
11. David Menotti, Giovani Chiachia, Alexandre X Falcão, and VJ Oliveira Neto. Vehicle license plate recognition with random convolutional networks. In Graphics, Patterns and Images (SIBGRAPI), 2014 27th SIBGRAPI Conference on, pages 298–303. IEEE, 2014.
12. Gee-Sern Hsu, Jiun-Chang Chen, and Yu-Zu Chung. Application-oriented license plate recognition. IEEE transactions on vehicular technology, 62(2):552–561, 2013.
13. Fernando Martin, Maite Garcia, and José Luis Alba. New methods for automatic reading of vlp's (vehicle license plates). In Proc. IASTED Int. Conf. SPPRA, volume 26, pages 257–271, 2002.
14. Shen-Zheng Wang and Hsi-Jian Lee. Detection and recognition of license plate characters with different appearances. In Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE, volume 2, pages 979–984. IEEE, 2003.
15. Bai Hongliang and Liu Changping. A hybrid license plate extraction method based on edge statistics and morphology. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, volume 2, pages 831–834. IEEE, 2004
16. Danian Zheng, Yannan Zhao, and Jiaxin Wang. An efficient method of license plate location. Pattern recognition letters, 26(15):2431–2438, 2005.
17. Abbas M Al-Ghaili, Syamsiah Mashohor, Abdul Rahman Ramli, and Alyani Ismail. Vertical-edge-based car-license-plate detection method. IEEE transactions on vehicular technology, 62(1):26–38, 2013.
18. Nikolaos Bellas, Sek M Chai, Malcolm Dwyer, and Dan Linzmeier. Fpga implementation of a license plate recognition soc using automatically generated streaming accelerators. In Parallel and Distributed Processing

Symposium, 2006. IPDPS 2006. 20th International, pages 8–pp. IEEE, 2006.

19. Hsien-Huang P Wu, Hung-Hsiang Chen, Ruei-Jan Wu, and Day-Fann Shen. License plate extraction in low resolution video. In Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, volume 1, pages 824–827. IEEE, 2006.
20. Sorin Draghici. A neural network based artificial vision system for licence plate recognition. International Journal of Neural Systems, 8(01):113–126, 1997.
21. J Barroso, EL Dagless, A Rafael, and J Bulas-Cruz. Number plate reading using computer vision. In Industrial Electronics, 1997. ISIE'97., Proceedings of the IEEE International Symposium on, pages 761–766. IEEE, 1997.
22. Javier Cano and Juan-Carlos Pérez-Cortés. Vehicle license plate segmentation in natural images. In Iberian Conference on Pattern Recognition and Image Analysis, pages 142–149. Springer, 2003.
23. Tsang-Hong Wang, Feng-Chou Ni, Keh-Tsong Li, and Yon-Ping Chen. Robust license plate recognition based on dynamic projection warping. In Networking, Sensing and Control, 2004 IEEE International Conference on, volume 2, pages 784–788. IEEE, 2004.
24. A Broumandnia and M Fathy. Application of pattern recognition for farsi license plate recognition. In ICGST Int. Conf. Graphics, Vision and Image Processing (GVIP), number V2, pages 25–31, 2005.
25. Jun Kong, Xinyue Liu, Yinghua Lu, and Xiaofeng Zhou. A novel license plate localization method based on textural feature analysis. In Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on, pages 275–279. IEEE, 2005.
26. Hung Ngoc Do, Minh-Thanh Vo, Bao Quoc Vuong, Huy Thanh Pham, An Hoang Nguyen, and Huy Quoc Luong. Automatic license plate recognition using mobile device. In Advanced Technologies for Communications (ATC), 2016 International Conference on, pages 268–271. IEEE, 2016.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import torch
from ultralytics import YOLO
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
import os  # Додаємо імпорт модуля os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import json
import cv2
import os
# 1. Вибір пристрою
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f'Using device: {device}')

# Для TensorFlow визначимо пристрій на GPU, якщо він доступний
gpus = tf.config.list_physical_devices('GPU')

if gpus:
    try:
        # Вибір GPU для використання
        tf.config.set_visible_devices(gpus[0], 'GPU')
```

```
        # Дозволити динамічне розподілення пам'яті
        tf.config.experimental.set_memory_growth(gpus[0], True)
        print("TensorFlow використовує GPU")
    except RuntimeError as e:
        print("Не вдалося налаштувати GPU:", e)
else:
    print("TensorFlow не знайдено GPU, використовує CPU")
# === 2. Директорії ===
base_dir = 'D:\\Study\\4Kyrs\\Part2\\Diplom\\code'
os.makedirs(base_dir, exist_ok=True)
output_dir = 'results'
os.makedirs(base_dir, exist_ok=True)
yolo_ckpt_path = os.path.join(base_dir, 'yolo_best.pt')
symbol_ckpt_path = os.path.join(base_dir, 'symbol_model.keras')
fcn_ckpt_path = os.path.join(base_dir, 'fcn_model.keras')

# 2. Шлях до моделі та даних
checkpoint_path = 'runs/detect/train/weights/best.pt'
data_yaml = 'Synthetic-Ukrainian-LP-dataset-10k/data.yaml'

# 3. Завантаження або навчання моделі
if os.path.exists(checkpoint_path):
    print("Завантаження моделі з чекпоінту...")
    model = YOLO(checkpoint_path).to(device)
else:
    print("Чекпоінт не знайдено. Починається тренування моделі...")
    model = YOLO('yolov5s.pt').to(device)
    model.train(data=data_yaml, epochs=50, batch=16, imgsz=640)
```

```
# 4. Оцінка моделі
metrics = model.val(data=data_yaml)

# 5. Виведення результатів
print("Test metrics:", metrics.results_dict)

# 6. Побудова графіків
if hasattr(metrics, 'plot') and callable(metrics.plot):
    metrics.plot()
else:
    print("Графік неможливо побудувати: метод plot не знайдено.")

# Примітка: metrics.imgs може бути недоступний — залежить від версії YOLO
# Альтернатива: показати приклад зображення з передбаченням
##sample_img = 'path_to_sample_image.jpg'
##results = model(sample_img)
##results[0].show()
# Функція для завантаження зображень
def load_images(image_paths, target_size=(64, 64)):
    images = [img_to_array(load_img(image_path, target_size=target_size)) for
image_path in image_paths]
    return np.array(images)

def load_labels(label_paths, num_classes=24):
    labels = []
    for label_path in label_paths:
        with open(label_path, 'r') as f:
            label = []
            for line in f.readlines():
```

```
            # Прочитати перше значення в рядку як клас
            class_id = int(line.split()[0])  # Беремо перше число в кожному рядку
            label.append(class_id)
        labels.append(label)
    return np.array(labels)

# Завантаження тренувальних зображень та міток
train_images_paths = [os.path.join('Synthetic-Ukrainian-LP-dataset-10k/train/images', f) for f in os.listdir('Synthetic-Ukrainian-LP-dataset-10k/train/images')]
train_labels_paths = [os.path.join('Synthetic-Ukrainian-LP-dataset-10k/train/labels', f.replace('.png', '.txt')) for f in os.listdir('Synthetic-Ukrainian-LP-dataset-10k/train/images')]

train_images = load_images(train_images_paths)
train_labels = load_labels(train_labels_paths)

# Завантаження валідаційних зображень та міток
val_images_paths = [os.path.join('Synthetic-Ukrainian-LP-dataset-10k/valid/images', f) for f in os.listdir('Synthetic-Ukrainian-LP-dataset-10k/valid/images')]
val_labels_paths = [os.path.join('Synthetic-Ukrainian-LP-dataset-10k/valid/labels', f.replace('.png', '.txt')) for f in os.listdir('Synthetic-Ukrainian-LP-dataset-10k/valid/images')]

val_images = load_images(val_images_paths)
val_labels = load_labels(val_labels_paths)

print(f"Train labels shape: {train_labels.shape}")
print(f"Validation labels shape: {val_labels.shape}")
```

```
train_labels = tf.keras.utils.to_categorical(train_labels, num_classes=24)
val_labels = tf.keras.utils.to_categorical(val_labels, num_classes=24)

print(f"Train labels shape: {train_labels.shape}")
print(f"Validation labels shape: {val_labels.shape}")

# Reshape labels to match model output (flatten the third dimension)
train_labels_reshaped = train_labels.reshape((-1, 24))  # Flatten 8 and 24 dimensions
val_labels_reshaped = val_labels.reshape((-1, 24))  # Flatten 8 and 24 dimensions

print(f"Train labels reshaped: {train_labels_reshaped.shape}")
print(f"Validation labels reshaped: {val_labels_reshaped.shape}")
def create_symbol_recognition_model(input_shape=(64, 64, 3), num_classes=24,
sequence_length=8):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        # Output layer should predict 24 classes for each of the 8 items (192 units total)
        layers.Dense(sequence_length * num_classes, activation='softmax'),
        layers.Reshape((sequence_length, num_classes))  # Reshape to (8, 24)
    ])
```

```
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
def create_improved_symbol_recognition_model(input_shape=(64, 64, 3),
num_classes=24, sequence_length=8):
    model = models.Sequential([
        layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape,
padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),  # Додано Dropout для регуляризації
        layers.Dense(sequence_length * num_classes, activation='softmax'),
        layers.Reshape((sequence_length, num_classes))
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

# Оновлена модель для символів
symbol_model = create_improved_symbol_recognition_model()

# Завантаження чи навчання моделі
if os.path.exists(symbol_ckpt_path):
    print("Завантаження моделі символів з чекпоінту...")
```

```
    symbol_model.load_weights(symbol_ckpt_path)
else:
    print("Навчання моделі символів...")
    symbol_model.fit(train_images, train_labels, epochs=20, batch_size=32,
validation_data=(val_images, val_labels))
    symbol_model.save(symbol_ckpt_path)

symbol_model = create_symbol_recognition_model()

if os.path.exists(symbol_ckpt_path):
    print("Завантаження моделі символів з чекпоінту...")
    symbol_model.load_weights(symbol_ckpt_path)
else:
    print("Навчання моделі символів...")
    symbol_model.fit(train_images, train_labels, epochs=10, batch_size=32,
validation_data=(val_images, val_labels))
    symbol_model.save(symbol_ckpt_path)
def create_fcn_model(input_shape=(64, 64, 3), num_classes=24,
sequence_length=8):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=input_shape),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
```

```
        layers.UpSampling2D((2, 2)),
        layers.Conv2DTranspose(64, (3, 3), activation='relu', padding='same'),
        layers.UpSampling2D((2, 2)),
        layers.Conv2DTranspose(num_classes, (3, 3), activation='softmax',
padding='same')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

def create_fcn_model_improved(input_shape=(128, 128, 3), num_classes=24,
sequence_length=8):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=input_shape),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
```

```
        layers.BatchNormalization(),

        layers.UpSampling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),

        layers.UpSampling2D((2, 2)),
        layers.Conv2D(num_classes, (3, 3), activation='softmax', padding='same')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
# One-hot encoding the labels
train_labels_one_hot = to_categorical(train_labels, num_classes=24)
val_labels_one_hot = to_categorical(val_labels, num_classes=24)

# Resize the labels to match the model's output (64x64)
train_labels_resized = tf.image.resize(train_labels_one_hot, size=(128, 128))
val_labels_resized = tf.image.resize(val_labels_one_hot, size=(128, 128))

# Now create and train the model
model = create_fcn_model_improved(input_shape=(128, 128, 3), num_classes=24)
##fcn_model.fit(train_images, train_labels_resized, epochs=10, batch_size=32,
validation_data=(val_images, val_labels_resized))

if os.path.exists(fcn_ckpt_path):
    print("Завантаження FCN моделі з чекпоінту...")
    model.load_weights(fcn_ckpt_path)
```

```
else:
    print("Навчання FCN моделі...")
    model.fit(train_images, train_labels_resized, epochs=10, batch_size=32,
validation_data=(val_images, val_labels_resized))
    model.save(fcn_ckpt_path)


# Аугментація даних для покращення тренування моделей
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,  # Додано вертикальний фліп
    fill_mode='nearest'
)

train_gen = datagen.flow(train_images, train_labels, batch_size=32)



def postprocess_predictions(predictions):
    # Приклад постобробки для виправлення помилок
    corrected_predictions = []
    for prediction in predictions:
        # Перевірка передбачених символів за допомогою словника або алгоритму
        corrected_prediction = correct_using_dictionary(prediction)
```

```
        corrected_predictions.append(corrected_prediction)
    return corrected_predictions



history = symbol_model.fit(train_gen, epochs=10, validation_data=(val_images,
val_labels))

# Відображення результатів
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Точність моделі')
plt.xlabel('Епохи')
plt.ylabel('Точність')
plt.legend(['Тренувальний набір', 'Валідаційний набір'])
plt.show()
output_dir = "all_predictions"
os.makedirs(output_dir, exist_ok=True)

# 1. Завантаження моделей
yolo_model = YOLO('yolov5su.pt')
symbol_model = tf.keras.models.load_model('symbol_model.keras')
fcn_model = tf.keras.models.load_model('fcn_model.keras')



def process_image(img_path, index):
    img = cv2.imread(img_path)
```

```
    if img is None:
        print(f"⚠ Неможливо завантажити зображення: {img_path}")
        return False  # Якщо зображення не завантажено, повертаємо False

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    ### --- YOLO: Виявлення номерного знаку ---
    results = yolo_model(img_rgb)
    boxes = results[0].boxes.xyxy.cpu().numpy()
    confidences = results[0].boxes.conf.cpu().numpy()

    if len(boxes) == 0:
        print(f"ℹ Номерний знак не знайдено на: {img_path}")
        return False  # Якщо номерний знак не знайдений, повертаємо False

    for i, (box, conf) in enumerate(zip(boxes, confidences)):
        x1, y1, x2, y2 = map(int, box)
        license_plate_img = img_rgb[y1:y2, x1:x2]

        if license_plate_img.size == 0:
            print(f"⚠ Порожній виріз номерного знаку в: {img_path}")
            continue

        try:
            # 3. Підготовка до символної моделі
            resized = cv2.resize(license_plate_img, (64, 64))  # Змінити розмір до (128,
32) або той, що очікує модель
            input_tensor = resized.astype(np.float32) / 255.0
            input_tensor = np.expand_dims(input_tensor, axis=0)
```

```
            # 4. Розпізнавання символів
            pred = symbol_model.predict(input_tensor)
            pred_label = np.argmax(pred, axis=-1)
            pred_str = ''.join([str(p) for p in pred_label[0]])
        except Exception as e:
            print(f"❌ Помилка при розпізнаванні символів: {e}")
            return False  # Якщо сталася помилка при розпізнаванні символів, повертаємо False

        # 5. Збереження результатів
        cv2.rectangle(img_rgb, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img_rgb, pred_str, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

        # Збереження обрізаного номерного знаку
        cv2.imwrite(os.path.join(output_dir, f"plate_crop_{index}_{i}.jpg"),
                    cv2.cvtColor(license_plate_img, cv2.COLOR_RGB2BGR))

        # Збереження тексту номерного знаку
        with open(os.path.join(output_dir, f"plate_text_{index}_{i}.txt"), "w") as f:
            f.write(pred_str)

    # 6. FCN — сегментація повного зображення
    try:
        resized_img = cv2.resize(img_rgb, (128, 128))
        fcn_input = resized_img.astype(np.float32) / 255.0
        fcn_input = np.expand_dims(fcn_input, axis=0)
```

```
        fcn_pred = fcn_model.predict(fcn_input)
        mask = np.argmax(fcn_pred[0], axis=-1)

        # 7. Збереження маски
        plt.imsave(os.path.join(output_dir, f"fcn_mask_{index}.png"), mask,
cmap="jet")
    except Exception as e:
        print(f"❌ Помилка сегментації FCN: {e}")
        return False  # Якщо сталася помилка при сегментації, повертаємо False

    # 8. Збереження зображення з рамками і текстом
    cv2.imwrite(os.path.join(output_dir, f"final_output_{index}.jpg"),
                cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))

    # Якщо всі етапи пройшли успішно, повертаємо True
    return True

# --- Обробити всі зображення ---
def process_dataset(image_folder):
    image_paths = [os.path.join(image_folder, f)
                   for f in os.listdir(image_folder) if f.lower().endswith((".jpg", ".png"))]
    for idx, path in enumerate(image_paths):
        if process_image(path, idx):
            print(f"✅ Оброблено і збережено: {path}")
        else:
            print(f"❌ Пропущено: {path}")

# ☞ Запуск
```

```
process_dataset("D:\\Study\\4Kyrs\\Part2\\Diplom\\code\\Synthetic-Ukrainian-LP-dataset-10k\\test\\images")

import os
import cv2
import numpy as np
import easyocr
from tkinter import Tk, filedialog, Label, Button, Frame, Canvas, Scrollbar, PhotoImage, VERTICAL, RIGHT, LEFT, BOTH, Y, CENTER
from PIL import Image, ImageTk

# OCR Reader
reader = easyocr.Reader(['en'])

def recognize_plate_full(img_path):
    image = cv2.imread(img_path)
    annotated_img = image.copy()
    results = reader.readtext(image)
    plate_crop = None
    plate_text = "Номер не розпізнано"

    for (bbox, text, confidence) in results:
        if len(text.replace(" ", "")) >= 7 and confidence > 0.5:
            pts = np.array(bbox, dtype=np.int32)
            cv2.polylines(annotated_img, [pts], isClosed=True, color=(0, 255, 0), thickness=2)
            x_min = int(min([p[0] for p in bbox]))
            x_max = int(max([p[0] for p in bbox]))
            y_min = int(min([p[1] for p in bbox]))
```

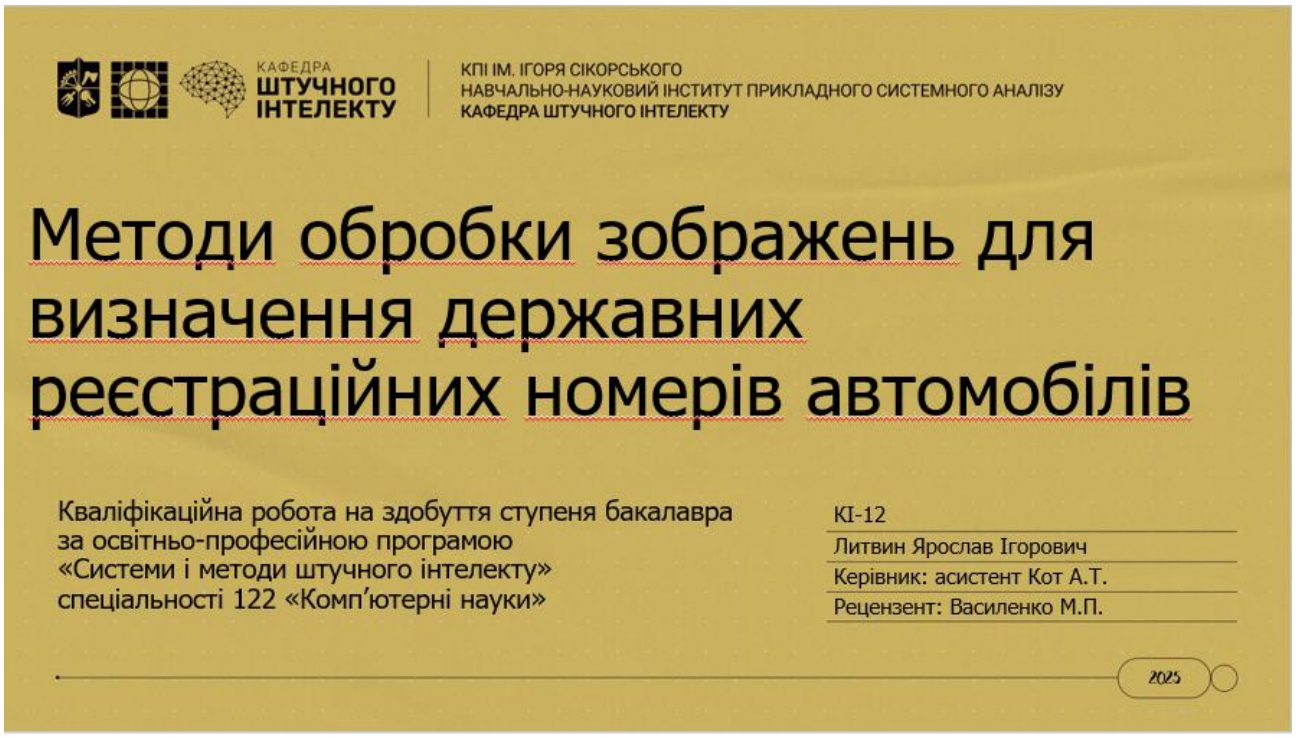
```
            y_max = int(max([p[1] for p in bbox]))
            plate_crop = image[y_min:y_max, x_min:x_max]
            plate_text = text
            break

    if annotated_img is not None:
        cv2.imwrite("annotated_result.jpg", annotated_img)
    if plate_crop is not None:
        cv2.imwrite("cropped_plate.jpg", plate_crop)
    with open("recognized_text.txt", "w", encoding="utf-8") as f:
        f.write(plate_text)

    return annotated_img, plate_crop, plate_text

def upload_image():
    file_path = filedialog.askopenfilename()
    if file_path:
        original = Image.open(file_path)
        original.thumbnail((400, 300))
        img_tk = ImageTk.PhotoImage(original)
        label_original.config(image=img_tk)
        label_original.image = img_tk

        annotated_img, plate_crop, result_text = recognize_plate_full(file_path)

        processed_pil = Image.fromarray(cv2.cvtColor(annotated_img,
cv2.COLOR_BGR2RGB))
        processed_pil.thumbnail((400, 300))
        processed_tk = ImageTk.PhotoImage(processed_pil)
```

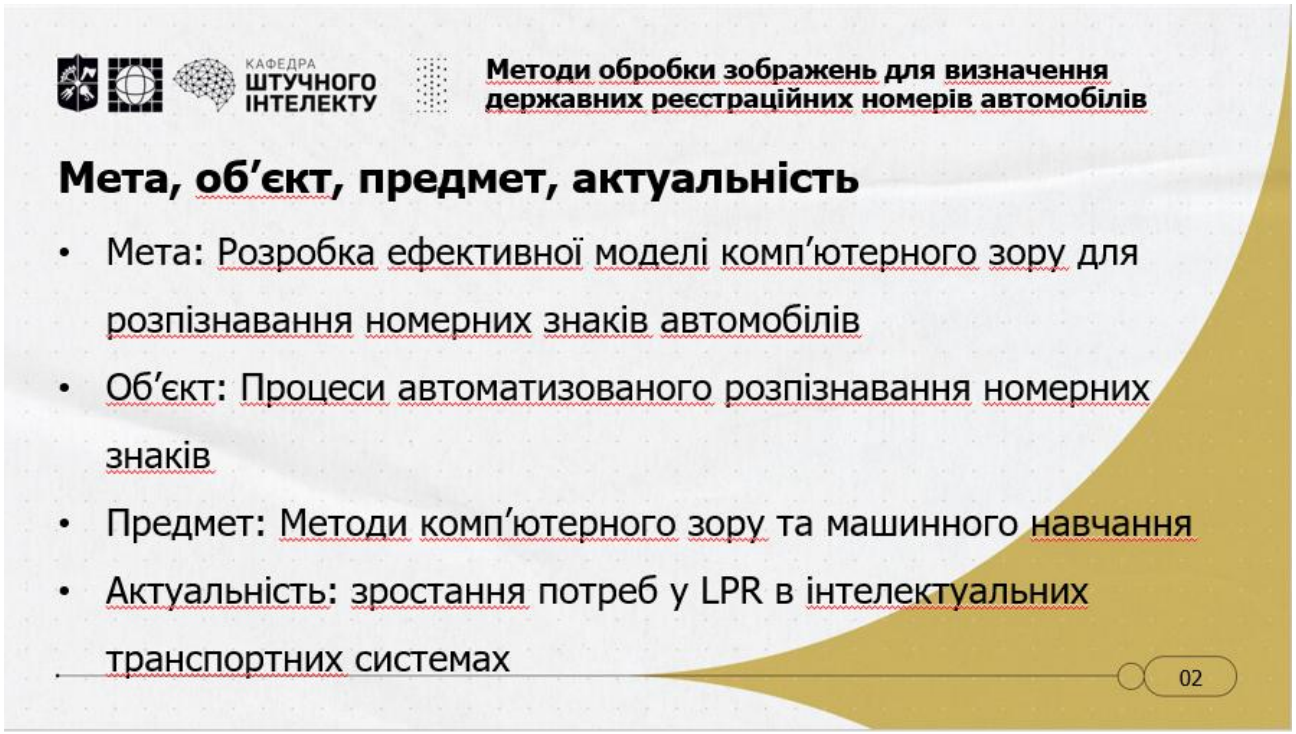
```
        label_processed.config(image=processed_tk)
        label_processed.image = processed_tk

        if plate_crop is not None:
            plate_pil = Image.fromarray(cv2.cvtColor(plate_crop,
cv2.COLOR_BGR2RGB))
        else:
            plate_pil = Image.new("RGB", (200, 50), color="gray")

        plate_pil.thumbnail((200, 50))
        plate_tk = ImageTk.PhotoImage(plate_pil)
        label_plate.config(image=plate_tk)
        label_plate.image = plate_tk

        label_result.config(text=f"Розпізнано: {result_text}")

# === Побудова UI ===
window = Tk()
window.title("Розпізнавання номерного знака")
window.geometry("600x800")
window.configure(bg="#1e1e1e")

main_frame = Frame(window, bg="#1e1e1e")
main_frame.pack(fill=BOTH, expand=1)

canvas = Canvas(main_frame, bg="#1e1e1e", highlightthickness=0)
canvas.pack(side=LEFT, fill=BOTH, expand=1)

scrollbar = Scrollbar(main_frame, orient=VERTICAL, command=canvas.yview)
```

```
scrollbar.pack(side=RIGHT, fill=Y)

canvas.configure(yscrollcommand=scrollbar.set)
canvas.bind_all("<MouseWheel>", lambda event: canvas.yview_scroll(int(-
1*(event.delta/120)), "units"))

scrollable_frame = Frame(canvas, bg="#2c2c2c")
canvas.create_window((0, 0), window=scrollable_frame, anchor="n")

def on_configure(event):
    canvas.configure(scrollregion=canvas.bbox("all"))
scrollable_frame.bind("<Configure>", on_configure)

Label(scrollable_frame, text="1. Завантажене зображення:", font=("Arial", 13),
bg="#2c2c2c", fg="white").pack(pady=10)
label_original = Label(scrollable_frame, bg="#2c2c2c")
label_original.pack(pady=5)

Label(scrollable_frame, text="2. Обробка та виявлення номера:", font=("Arial",
13), bg="#2c2c2c", fg="white").pack(pady=10)
label_processed = Label(scrollable_frame, bg="#2c2c2c")
label_processed.pack(pady=5)

Label(scrollable_frame, text="3. Вирізаний номерний знак:", font=("Arial", 13),
bg="#2c2c2c", fg="white").pack(pady=10)
label_plate = Label(scrollable_frame, bg="#2c2c2c")
label_plate.pack(pady=5)
```

```
label_result = Label(scrollable_frame, text="Результат:", font=("Arial", 14),
bg="#2c2c2c", fg="#00ff00")
label_result.pack(pady=20)

Button(scrollable_frame, text="📤 Завантажити зображення",
command=upload_image, font=("Arial", 12), bg="#0a84ff",
fg="white").pack(pady=20)

window.mainloop()
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ





КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Галузь застосування

- Інтелектуальні транспортні системи (ITS)
- Камери відеофіксації порушень ПДР
- Паркінг-сервіси з автоматичною ідентифікацією
- Контроль в'їзду/виїзду в закритих зонах
- Автоматизовані пункти пропуску (митниця, АЕС, бази)

03

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Огляд предметної області

- Вперше реалізовано у Великобританії у 1976 році.
- Перші покоління: базувалися на класичних алгоритмах — виявлення контурів, проекції, ССА.
- Сучасні — працюють на основі CNN, YOLO, CRNN, FCN та обробляють тисячі зображень у реальному часі.

04

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи обробки зображень для визначення державних реєстраційних номерів автомобілів

Класичні методи обробки

- Edge Detection (Canny, Sobel): виявлення контурів об'єктів.
- Thresholding: фіксоване/адаптивне порогове виділення знаку.
- OCR (Tesseract): текстове розпізнавання після сегментації.
- Недоліки: низька точність, залежність від освітлення, чутливість до шуму.

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Заголовок слайду з рисунками, таблицями тощо або назва дипломної роботи

Глибинне навчання у LPR

- CNN: витяг ознак, розпізнавання шаблонів.
- YOLO (You Only Look Once): одноетапна детекція об'єктів.
- CRNN (Convolutional Recurrent Neural Network): послідовне розпізнавання символів.
- Висока точність при різних умовах зйомки (туман, ніч, рух).

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Обрана архітектура

- YOLOv5 → локалізація знаку
- FCN → піксельна сегментація (фон/символи)
- CNN → розпізнавання символів
- Інтеграція в одному пайплайні забезпечує повний цикл аналізу зображення.

07

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи обробки зображень для визначення державних реєстраційних номерів автомобілів

Схема роботи системи

- Отримання зображення
- Детекція знаку (YOLO)
- Вирізання області
- Попередня обробка (масштабування, контраст, бінаризація)
- Розпізнавання (CNN)
- Вивід результату

08

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Заголовок слайду з рисунками, таблицями тощо або назва дипломної роботи

Датасет і підготовка

- Dataset: Synthetic-Ukrainian-LP-dataset-10k
- Мітки: координати номерів, текст номера
- Аугментації: обертання, масштабування, шум, розмиття, контраст
- Ціль: підвищити узагальнювальну здатність моделей

09

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи обробки зображень для визначення державних реєстраційних номерів автомобілів

Навчання YOLOv5

- Вхід: зображення + bounding boxes
- PyTorch + GPU прискорення
- Augmentations: HSV shift, mosaic
- Вихід: координати та клас (номерний знак)

10

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

CNN для розпізнавання

- Архітектура: 3 блоки Conv + MaxPooling → Dense → Softmax
- Використано TensorFlow/Keras
- Loss: CTC (Connectionist Temporal Classification)
- Вивід: послідовність символів без попередньої сегментації

11

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

FCN-сегментація

- Кожен піксель класифікується як «фон» або «символ»
- Покращення чіткості символів перед OCR
- Приклад: різке відділення символів від білого фону

12

Програмна реалізація

- GUI: Tkinter
- GPU підтримка
- Автоматичне завантаження моделей з чекпоінтів
- Візуалізація проміжних кроків
- Збереження результатів

Тестування та оцінка

- Різні умови: тіні, кут нахилу, затемнення
- 500+ зображень у тесті
- Метрики: Accuracy, Precision, Recall, IoU

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Результати розпізнавання

- YOLOv5: 93.2% точність локалізації
- CNN: 91.4% точність розпізнавання
- Середній час обробки: 0.9 секунди/зображення

15

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Інтерфейс користувача

- Завантаження фото
- Автоматичне оброблення
- Показ:
  - оригінал
  - виявлений знак
  - результат розпізнавання
  - Збереження всіх етапів у локальну папку

16

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Заголовок слайду з рисунками, таблицями тощо або назва дипломної роботи

Функціонально-вартісний аналіз

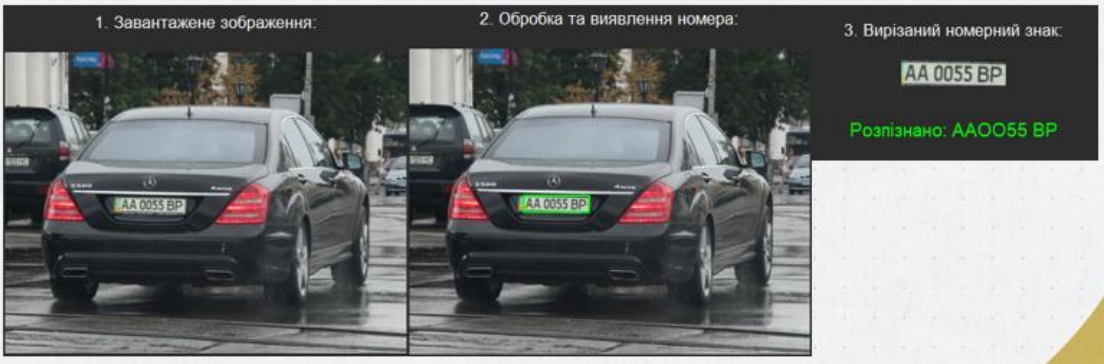
- Порівняння TensorFlow vs PyTorch
- PyTorch: менше ресурсів, зручніший inference
- Загальна ефективність: швидкість, точність, масштабованість

17

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Методи обробки зображень для визначення державних реєстраційних номерів автомобілів

Результати експериментів



18

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Висновки

- Реалізовано ефективну LPR-систему
- Досягнуто високої точності без використання реального датасету
- Система готова до розширення на реальні умови, інтеграції в інфраструктуру

19

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Дякую за увагу

20