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:
«Адаптивна мережа стиснення динамічного діапазону зображень для виявлення дорожніх об'єктів»

Виконав:
студент IV курсу, групи КІ-13
Денисенко Денис Геннадійович _____

Керівник:
доцент кафедри ШІ, к.т.н., доцент,
Тимошенко Юрій Олександрович _____

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., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
доцент кафедри інформаційної безпеки НН ФТІ, к.т.н, доцент,
Гальчинський Леонід Юрійович _____

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 та методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту

Денисенку Денису Геннадійовичу

1. Тема роботи «Адаптивна мережа стиснення динамічного діапазону зображень для виявлення дорожніх об'єктів», керівник роботи Тимошенко Юрій Олександрович, доцент, кандидат технічних наук, 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: термальні фотографії дорожньої сцени.
4. Зміст роботи: аналіз застосування динамічного діапазону та тонового відображення, впровадження методів глибокого навчання для стиснення динамічного діапазону та детекції, програмна реалізація моделей тонового

відображення та аналіз результатів, функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу: електронна презентація.

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	24.04.2025	Виконано
2	Підготовка першого розділу	1.05.2025	Виконано
3	Підготовка другого розділу	09.05.2025	Виконано
4	Розробка програмного продукту	16.05.2025	Виконано
5	Підготовка третього розділу	23.05.2025	Виконано
6	Оформлення дипломної роботи	05.06.2025	Виконано
7	Підготовка презентації доповіді	9.06.2025	Виконано

Студент Денис ДЕНИСЕНКО

Керівник Юрій ТИМОШЕНКО

РЕФЕРАТ

Дипломна робота: 106 с., 21 рис., 8 табл., 20 посилань, 1 додаток.

СИСТЕМИ ВИЯВЛЕННЯ ОБ’ЄКТІВ, ДИНАМІЧНИЙ ДІАПАЗОН, ТОНОВЕ ВІДОБРАЖЕННЯ, МУЛЬТИЗАДАЧНЕ НАВЧАННЯ, БЕЗ’ЯКІРНА ДЕТЕКЦІЯ.

У роботі досліджуються підходи глибокого навчання для стиснення високого динамічного діапазону термальних зображень у низький, аналізується вплив на точність детекції після тонового відображення відмінними між собою підходами для колірних зображень, на основі чого систематизовано необхідні вимоги для створення власної архітектури, орієнтованої на одноканальні термальні зображення, і ставиться на меті покращення виявлення дорожніх об’єктів для використання в системах допомоги водієві.

Об’єкт дослідження – детекція дорожніх об'єктів на термальних зображеннях після тонового відображення. Предмет – мережі для стиснення динамічного діапазону зображень. Метою роботи є створення власної мережі тонового відображення, яка враховує особливості термальних зображень з урахуванням проблем, виявлених при аналізі існуючих методів, що покращить точність виявлення об'єктів на вихідних зображеннях з низьким динамічним діапазоном. Актуальність роботи зумовлена потребою використання для фіксації дорожньої сцени високого динамічного діапазону, який не може бути відображений на сучасних дисплеях і незначною кількістю реалізацій тонового відображення для термальних зображень у літературі.

Результатом роботи є модель, адаптована до природи термальних зображень, що гарантує підвищення точності детекції у порівнянні з існуючими методами за основними метриками.

ABSTRACT

Thesis: 106 pages, 21 figures, 8 tables, 20 sources, 1 appendices.

OBJECT DETECTION SYSTEMS, DYNAMIC RANGE, TONE MAPPING, MULTI-TASK LEARNING, ANCHOR-FREE DETECTION.

The work examines deep-learning approaches for compressing the high dynamic range of thermal images into a lower range, analyses the influence on detection accuracy after tone mapping by different colour-image approaches, on the basis of which the necessary requirements for creating an own architecture oriented to single-channel thermal images are systematised, and sets as its aim the improvement of road-object detection for use in driver-assistance systems.

The object of the study – detection of road objects on thermal images after tone mapping. Subject – networks for compressing image dynamic range. The purpose of the work is to create an own tone-mapping network that takes into account properties of thermal images, regarding to the problems found during the analysis of existing methods, which will improve object-detection accuracy on low dynamic range output images. The relevance of the work is conditioned by the need to capture road scenes with a high dynamic range that can not be displayed on modern screens and by the small number of tone-mapping implementations for thermal images.

The result of the work is a model adapted to the nature of thermal images that guarantees increased detection accuracy compared with existing methods according to the fundamental metrics.

ЗМІСТ

ВСТУП 8

РОЗДІЛ 1 АНАЛІЗ ЗАСТОСУВАННЯ ДИНАМІЧНОГО ДІАПАЗОНУ ТА ТОНОВОГО ВІДОБРАЖЕННЯ 10

1.1 Методи детекції у дорожніх сценах 10

1.2 Загальна характеристика динамічного діапазону 11

1.3 Модель HVS для стиснення динамічного діапазону 15

1.4 Класичні методи стиснення динамічного діапазону 18

1.5 Проблеми детекції на HDR зображеннях 20

1.6 Постановка задачі 22

Висновки до розділу 1 23

РОЗДІЛ 2 ВПРОВАДЖЕННЯ МЕТОДІВ ГЛИБОКОГО НАВЧАННЯ ДЛЯ СТИСНЕННЯ ДИНАМІЧНОГО ДІАПАЗОНУ ТА ДЕТЕКЦІЇ 24

2.1 Методи тонового відображення моделями глибокого навчання 24

2.1.1 Аналіз архітектури TMO_CAN 24

2.1.2 Аналіз архітектури UnCLTMO 27

2.2 Методи детекції 30

2.2.1 Аналіз архітектури FCOS 32

2.2.2 Аналіз архітектури YOLOv8 35

Висновки до розділу 2 38

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТОНОВОГО ВІДОБРАЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 40

3.1 Огляд та попередня обробка набору даних 40

3.2 Запропонована архітектура мережі тонового відображення 45

3.3 Метрики для оцінки якості тонового відображення 49

3.4 Метрики для оцінки якості детекції 52

3.5 Аналіз результатів тонового відображення 55

3.6 Оцінка результатів детекції 61

Висновки до розділу 3 64

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ 66

4.1 Постановка задачі проектування 67

4.2 Обґрунтування функцій програмного продукту 67

4.3 Обґрунтування системи параметрів програмного продукту 71

4.4 Аналіз експертного оцінювання параметрів 74

4.5 Аналіз рівня якості варіантів реалізації функцій 78

4.6 Економічний аналіз варіантів розробки ПП 79

4.7 Вибір кращого варіанту ПП техніко-економічного рівня 85

Висновки ро розділу 4 86

ВИСНОВКИ 87

ПЕРЕЛІК ПОСИЛАНЬ 89

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ 92

ВСТУП

У[1] сучасному автомобільному середовищі, що стрімко рухається до автономії та інтелектуальних систем допомоги водієві, висока якість сприйняття дорожньої обстановки стає критичною складовою безпеки й ефективності руху. Особливо гостро це питання постає в умовах змінного освітлення, коли стандартні камери з обмеженим динамічним діапазоном не здатні одночасно передати деталізовану інформацію про яскраві та темні ділянки кадру. Недостатня розрізнювальна здатність у таких умовах призводить до втрати розпізнавальних ознак транспортних засобів, пішоходів чи дорожньої розмітки, що нині заважає впровадженню безпілотних систем водіння.

Проте із розвитком апаратного забезпечення росте й інтерес до програмних методів підвищення якості вхідних фото/відеопотоків не тільки для систем виявлення. Стиснення динамічного діапазону вже довело свою користь у кіноіндустрії та фотографії, однак його впровадження в сферу дорожньої безпеки стикається з новими викликами. По-перше, класичні оператори компресії яскравості часто створюють артефакти, які знижують точність подальшої автоматизованої обробки. По-друге, алгоритми, орієнтовані на естетичну якість зображення для людини, не враховують вимог машинного зору й можуть спотворювати ознаки, критичні для детекторів об’єктів.

[1]Тут і нижче використано такий інструмент 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.).

Таким чином, мета даної дипломної роботи – провести глибокий аналіз існуючих методів стиснення динамічного діапазону та сформулювати вимоги до адаптивного рішення, оптимізованого під завдання детекції дорожніх об’єктів на термічних зображеннях.

РОЗДІЛ 1 АНАЛІЗ ЗАСТОСУВАННЯ ДИНАМІЧНОГО ДІАПАЗОНУ ТА ТОНОВОГО ВІДОБРАЖЕННЯ

1.1 Методи детекції у дорожніх сценах

Детекція дорожніх об’єктів – завдання автоматичного знаходження та локалізації транспортних засобів, пішоходів і розмітки на знімку чи відеокадрі, пройшла еволюцію від ковзного вікна з каскадом Хаара, що працював у реальному часі на процесорах початку 2000-х рр. до згорткових мереж, які сьогодні безпосередньо оптимізують простір ознак під кінцеву мету керування транспортом [3].

Першим проривом стали гістограми орієнтованих градієнтів, що разом із лінійною SVM класифікували пішоходів із точністю, недосяжною для попередніх інтенсивнісних дескрипторів [2]. Усі ці методи вимагали складного добору порогів і страждали від різноманіття ракурсів, доки не з’явились згорткова мережа R-CNN, яка дозволила навчатись кінцевому критерію без ручного підбору ознак.

Розвиток глибоких згорткових мереж докорінно змінив постановку: R-CNN запропонував використати превісно навчений CNN як універсальний екстрактор і донавчати його на невеликій кількості регіон-пропозицій, автоматично оптимізуючи map-розклад у просторах активацій таким чином, щоб максимізувати середню точність mAP [3]. Проте двоступеневі системи на кшталт R-CNN і Faster R-CNN залишалися затратними, бо генерація пропозицій і повторна прогонка через базову мережу вимагали багато часу. Потреба в низькій латентності привела до появи одноетапних детекторів YOLO, де координати боксів і ймовірності класів регресуються безпосередньо з тензора останнього згорткового шару

Однак, покращення ефективності детекторів є не єдиною проблемою у виявленні об’єктів, точність отриманих передбачень також залежить від якості зображень, що подаються на вхід моделі. У змінних умовах зйомки дорожньої

сцени із постійно перемінною експозицією та погодними умовами можуть виникати як пересвіти, так і глибокі тіні, особливо під час руху транспортного засобу або за швидкої зміни освітлення, що значно погіршує видимість сцени і призводить до втрати інформації та розпізнавальної здатності малих об’єктів, таких як, власне, номерні знаки, обличчя людей. Це може призвести до значних безпекових загроз: втрати важливих деталей при відеореєстрації, наприклад рис обличчя зловмисника чи номери автомобіля, який сприяв утворенню аварійної ситуації. У свою чергу, з точки зору виявлення об’єктів, пересвіти призводять до насичення пікселів, внаслідок чого шари згорткових мереж отримують однорідні ділянки із максимальними значеннями активацій, що ускладнює розпізнавання тонких контурів об’єктів, що призводить до зниження точності локалізації та класифікації.

Ці проблеми вирішуються за допомогою збільшення динамічного діапазону зображення зі стандартного до високого, який збільшує максимальне можливе зняте або відображуване значення яскравості чи колірної гами об’єкта, тим самим згладжуючи перехід між областями з різною інтенсивністю, збільшуючи деталізацію.

1.2 Загальна характеристика динамічного діапазону

У цифрових системах зображення піксель визначається набором числових значень, які інтерпретуються дисплеєм або програмою-реконструктором як яскравість і колір. У традиційному SDR (Standard Dynamic Range) форматі найпоширеніше квантування, інтенсивності поділені на рівні відліки з кроком визначеним числом бітів – 8 біт на канал у просторі RGB, тобто кожен піксель має три цілі числа від 0 до 255 (2^8), які лінійно відображаються на видиму інтенсивність. Таке вибіркове квантування дає приблизно 87 кодів до точки білого, тому між сусідніми областями пікселів в

тінях різниця сягає цілої десятої частки освітленості, що сприймається як злиття деталей у темному. Першим кроком вирішення цієї проблеми стало впровадження формату OpenEXR (1999). Формат дозволив зберігати 16-бітні плаваючі числа для кожного каналу: 10 біт мантиси для R, G, B, 5 біт спільної експоненти та один біт знакового розряду числа, що дало змогу застосовувати ширше кольорове представлення у областях, де є критично важливим явне розрізнення переходів у тоні [1].

Більш загально, динамічний діапазон (DR) можна визначити як відношення між найбільшим і найменшим вимірюваним значенням інтенсивності сигналу чи випромінювання, які система реєстрації здатна передати без втрати у шумі, і саме ця величина встановлює межі чутливості системи візуалізації до контрастів сцени. Однак, для зручності це відношення традиційно виражають у вигляді двійкового логарифму – стопах, де один стоп відповідає подвоєнню потоку випромінювання. Формально ми можемо визначити динамічний діапазон як [8]:

$$DR = \frac{L_{max}}{L_{min}}, \quad N = \log_2\left(\frac{L_{max}}{L_{min}}\right) \tag{1.1}$$

де L_{max} – максимальна потужність сигналу;

L_{min} – мінімальна потужність сигналу.

Традиційне поняття динамічного діапазону можна розширити на бітове представлення, відтак бітова глибина описує, скільки дискретних рівнів може прийняти інтенсивність пікселя. Якщо канал кодується b бітами, то кількість можливих значень дорівнює 2^b, тоді для кольорового зображення з трьома каналами кількість комбінацій стає 2^{3b}. Наприклад, у SDR-відео з 8 бітами на канал це $2^{24} \approx 16{,}7$ млн можливих комбінацій кольорів – перехід до 10 біт збільшує палітру до понад мільярда, але насамперед підвищує точність градацій тону, зменшуючи крок квантування до 1/1023 замість 1/255 [4]. Порівняння HDR та SDR зображень предстаалено на рисунку 1.1:

Рисунок 1.1 – Порівняння HDR та SDR зображень[2]

Слід зазначити, що відповідно об’єкту дослідження, системи виявлення дорожніх об’єктів мають працювати в умовах низької видимості, тобто із нічними сценами та сценами туману, для цього використовують термальні камери, зображення з яких візуалізують у white-hot палітрі, тобто у градаціях сірого: найгарячіші ділянки відображаються білими, найхолодніші – чорними. Приклад зображення у white-hot палітрі ілюстровано на рисунку 1.2.

Для white-hot знімків, де сигнал формується не за рахунок відбитої, а за рахунок власної теплової радіації об’єкта, високий динамічний діапазон дає змогу вирізнити тонкі градації теплового випромінювання, у той час як вузький діапазон зливатиме гарячі та холодні ділянки.

White-hot сенсор замість трійки кольорових каналів фіксує єдиний спектральний канал довгохвильового ІЧ-діапазону в дискретному діапазоні бітової глибини [0; 65535], відповідно, завдяки використанню HDR термальних сенсорів на 14 бітах вздовж діапазону досягається квантувальний крок у 16 разів менший, ніж при 10 бітах, що прямо впливає на точність виявлення дрібних об’єктів на дорозі. Окрім цього, у white-hot різниця між

[2]Джерело зображення: https://www.winxdvd.com/seo-img/convert-4k-video/hdr-vs-sdr.jpg

градаціями кольорів проявляється лише яскравісним кроком, що спрощує статистику градієнтів у карті ознак конволюційної мережі.

Рисунок 1.2 – Приклад white-hot зображення

Проте, через фізичну основу роботи термальних сенсорів на мікроболометрах, до термічних знімків також додається шум, спричинений чутливістю детектора, які характеризує метрика NETD (Noise-Equivalent Temperature Difference), що визначає мінімальну різницю температур, яку детектор здатен розрізнити зі співвідношенням сигнал/шум рівним одиниці, і зазвичай вимірюється в міліКельвінах (mK) [19]. Для більшості камер типовим показником NETD є 30-100mK, що призводить до значного падіння відображення теплових порогів зашумленням, порівняння знімків на сенсорах із NETD 60мК та 80мК відповідно зображено на рисунку 1.3:

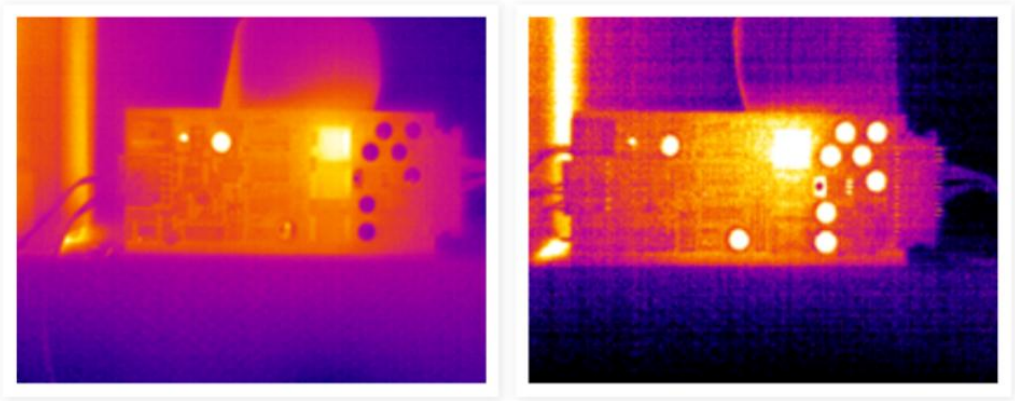

Рисунок 1.3 – Знімки із сенсорів з різним значенням NETD [19]

Наявність високого внутрішнього динамічного діапазону в сучасних термальних сенсорах безпосередньо компенсує ці фізичні обмеження системи, оскільки дає суттєвий запас градацій. Але більшість екранів наразі не підтримує відображення 16 бітних зображень, тому що потребує удвічі більшої швидкості інтерфейсів, значно дорожчих драйверів і підсилювачів керування підпіксельними напругами, через що виникає проблема програмного стиснення високого динамічного діапазону у менший. Якщо виконувати таку операцію лінійно, нормалізувавши HDR діапазон за його кроком квантування, то через значну різницю дискретного рівня, отримані бітові представлення зображення на R, G, B каналах будуть наближатись до чорного кольору. Єдиний вихід – нелінійно стиснути карту температур, аби зберегти помітні градієнти, під що було розроблено низку методів стиснення динамічного діапазону, або ж тонового відображення.

1.3 Модель HVS для стиснення динамічного діапазону

Постановка задачі стиснення динамічного діапазону зображень розвивається з урахуванням реальних потреб як людини, так і особливостей автоматизованих систем. У попередньому підрозділі було розглянуто загальне

поняття динамічного діапазону та причини використання HDR-даних у термальних white-hot камерах. Однак сама задача тонового відображення не обмежується просто стисненням числових значень, а передбачає моделювання сприйняття людиною високо динамічних сцен з максимальною кількістю деталізації. У цьому контексті основним робочим інструментом стає концепція Human Visual System (HVS) як абстрактна модель, що описує яким чином людина сприймає контраст, колір і яскравість у різних умовах освітлення.

Під HVS-орієнтованим тоновим відображенням розуміють набір алгоритмічних етапів, які апроксимують ключові властивості людського зору [12]. З позиції моделі HVS важливими є такі моменти: чутливість до локальних контрастів на різних масштабах, адаптивна компресія світловідповіді залежно від середньої інтенсивності сцени, а також корекція колірного сприйняття, що дозволяє зберегти відносну насиченість і відтінки, хоча реальні дисплеї мають обмежену палітру. Ці етапи формують ядро будь-якого HVS-моделюючого оператору тонового відображення, і зв’зок між ними зображений на рисунку 1.4.

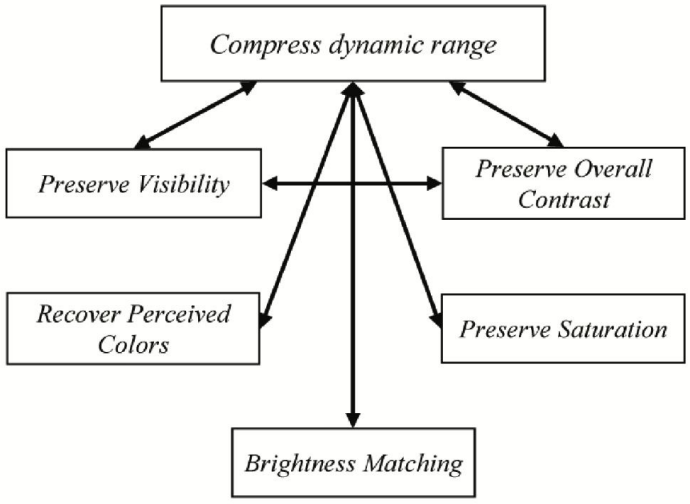

Рисунок 1.4 – Основні складові HVS-тонового відображення[3]

[3]Джерело зображення: https://www.researchgate.net/profile/Massimo-Fierro/publication/260969452/figure/fig1/AS:669089869615130@1536534864922/Graphical-scheme-of-TMOs-trade-off.png

По-перше, зображення розкладається на рівні яскравості, де кожен рівень відповідає різному просторовому масштабу сприйняття контрасту. Така репрезентація дозволяє оцінити локальні зміни інтенсивності, що моделює просторову чутливість людського сприйняття до контрастів різного розміру. По-друге, для кожного рівня застосовується нелінійна регуляція адаптації яскравості: завдяки цьому на ділянках із високим середнім світловим потоком послаблюється вплив найяскравіших пікселів, а в тінях навпаки підвищується чутливість до дрібних перепадів. По-третє, застосовуються просторові фільтри, аби виділити локальні градієнти та контури об’єктів: ці фільтри апроксимують здатність HVS швидко реагувати на різкі зміни в яскравості та підсилювати візуальну увагу на границях. По-четверте, вводиться корекція колірного сприйняття: оскільки людина сприймає кольори не лінійно, а через складні взаємодії між світловими і колірними рецепторами, щоб зберегти відтінки та контраст кольорів у межах обмеженого вихідного діапазону. Останній етап – це узгодження під характеристики конкретного дисплея: реалізується гамма-корекція, встановлення чорної та білої точок (anchoring), а також врахування максимального і мінімального рівня яскравості пристрою, що гарантує, що тонове відображення остаточно відобразить віджаті HDR-дані у формат, прийнятний для екрана [12]. Саме така схема задає вимоги для сучасних моделей тонового відображення: необхідне одночасне збереження як загального контрасту, так і дрібних деталей у тінях і в дуже яскравих ділянках. Однак, така концепція є дуже складною у реалізації і зазвичай використовується у підходах до створення моделей глибокого навчання, і задовго до її появи використовували алгоритмічні методи, такі як глобальні і локальні оператори, що описані у наступному підрозділі.

1.4 Класичні методи стиснення динамічного діапазону

З огляду на алгоритмічний підхід, тоновим відображенням називають керований нелінійний процес $T(L)$, який перетворює HDR радіансну карту на обмежений SDR інтервал, забезпечуючи перехід в придатний для відображення діапазон, зберігаючи при цьому відносну інтенсивність сигналу пікселя в зменшеній бітовій глибині [16].

Класичні алгоритми тонового відображення поділяють на глобальні та локальні оператори. Глобальні оператори задають одну універсальну функцію перетворення яскравості для всіх пікселів. Наприклад, Reinhard у 2002 запропонував фотографічний оператор тонового відображення. Спочатку відбувається масштабування світлочутливості за ключовим коефіцієнтом $\frac{a}{\overline{L_w}}$ [16]:

$$L(x,y) = \frac{a}{\overline{L_w}}\, L_w(x,y), \quad a \approx 0.18, \tag{1.2}$$

де $\overline{L_w}$ – середньогеометрична яскравість сцени;

a – ключове значення.

Потім використовується функція компресії контрасту:

$$\mathrm{L_d(x,y)} = \frac{\mathrm{L(x,y)}}{1+\mathrm{L(x,y)}}, \tag{1.3}$$

$$\mathrm{L_d(x,y)} = \frac{\mathrm{L(x,y)}\left(1+\frac{\mathrm{L(x,y)}}{\mathrm{L_{white}^2}}\right)}{1+\mathrm{L(x,y)}},$$

де L_{white} – поріг засвічення (значення, яке відображається чисто білим).

Такий оператор гарантує, що усі значення потрапляють у діапазон [0,1], хоча найяскравіші пікселі сильно стискаються.

Іншим прикладом глобального методу є адаптивне логарифмічне тонмапування. Воно задає відображення (1.4) [16]:

$$L_d = \frac{\log(L_w+1)}{\log(L_{max}+1)}, \tag{1.4}$$

де L_w – світлочутлива яскравість пікселя;

L_{max} – максимальне значення сцени.

Ця формула відображає максимум сцени як білий, а інші рівні змінює логарифмічно, однак чистий логарифм часто занадто затемнює середні тони, тому зазвичай задають змінну основу логарифму.

У свою чергу, локальні оператори тонового відображення відрізняються від глобальних тим, що параметри перетворення змінюються у просторі зображення: кожен піксель стискається з огляду на статистику власного оточення, а не всієї сцени. Завдяки цьому локальні методи зберігають дрібні деталі у тінях і світлих ділянках, але можуть спричиняти гало артефакти й потребують більше обчислень.

Найпростішу локальну ідею запропонував Ward у середині 1990-х: зображення спершу згладжують (наприклад, гаусовим ядром або усередненням на піраміді), отримуючи базову карту $B(x,y)$, що відповідає локальній адаптації ока. Кожен вихідний піксель обчислюють як [8]:

$$L_d(x,y) = \frac{L_w(x,y)}{k+B(x,y)}, \tag{1.5}$$

де $k \approx 1-3$ визначає контраст сцени;

$B(x,y)$ – операція попіксельного згладжування зображення.

Також Reinhard описав і локальну версію свого алгоритму. Для кожного пікселя шукають мінімальний радіус s, при якому середній градієнт $|B_{s+1} - B_s|/B_s$ опускається нижче порога $\epsilon = 0.05$ – при такому значенні навколишня область стає однорідною [8]:

$$L_d(x,y) = \frac{L_w(x,y)}{1+ L_w(x,y)} \quad \frac{0.18}{1+ B_s(x,y)}, \tag{1.6}$$

де $B_s(x,y)$ – операція попіксельного згладжування зображення у радіусі s.

Варто зазначити, що у локальних операторах тонового відображення часто спостерігається так званий гало-артефакт – поява світлих або темних кільцевих ореолів навколо контрастних меж об’єктів, у свою чергу після глобального тонового відображення, з'являється постеризація – помітні стрибки яскравості у плавних градієнтах, коли неперервний перехід замінюється кількома майже однорідними смугами через те, що одна й та сама нелінійна крива застосовується до всіх пікселів. Відповідно такі методи стиснення динамічного діапазону не підходять для подачі на детектор, оскільки викликають спотворення в картах ознак і призводять до неконсистентних активацій у початкових згорткових шарах. Натомість, методи тонового відображення на основі глибоких нейронних мереж переважають класичні глобальні й локальні оператори тому, що мережа навчається безпосередньо оптимізувати просторово-варіаційну нелінійну функцію, мінімізуючи одночасно втрату контрасту й появу артефактів, чого неможливо досягти єдиною аналітичною формулою, і мають менший час інференсу [16].

1.5 Проблеми детекції на HDR зображеннях

У практиці детекції об’єктів на HDR-кадрах ключовою проблемою стає саме тонове відображення: алгоритми перетворення високодинамічного діапазону в 8-бітний SDR нерідко призводять до підсилення шумових артефактів та розмиття контурів. Алгоритмічні оператори тонового відображення – наприклад, оператор Reinhard компресують відразу весь

діапазон й не враховують локальний контраст, тому дрібні деталі в тінях або яскраві відблиски можуть бути згладжені або засвічені. Він стискає піки та спадні значення інтенсивностей за досить тривіальною нелінійною формою, коли пікові значення зберігають свою інтенсивність, а спадні згасають набагато швидше, що знижує чутливість моделі до швидких градієнтів яскравості, які важливі для точного локалізування об’єктів, тому для отримання якісного тонмапового зображення для ефективної роботи детекторів є необхідним використання моделей глибокого навчання, спроєктованих з урахуванням сценічних особливостей кадру та характеристик отримуваних кадрів, або відеопотоку.

Однак, у результаті тонового відображення за допомогою моделей глибокого навчання часто отримуються розмиті краї об’єктів, де мережа пересвітлює грані й додає надмірну зернистість у середніх тонах, або навпаки робить переходи між різними інтенсивнісними рівнями незначними, щоб згладити різкі переходи відповідно для покращення сприйняття відповідно до задачі HVS. Ці зміни локальної структури зображення зміщують ознаки на вхідних шарах детекторів і, як наслідок, істотно погіршують якість детекції.

По-третє, у відкритому доступі не вистачає HDR-анотованих датасетів. Найпопулярніші набори, як-от MS COCO та KITTI, містять лише SDR-зображення з 8 бітами, оскільки створення високодинамічного діапазону є дороговартісним через тому не вистачає достатньої кількості даних для тренування моделі детекції навчаються в умовах обмеженого діапазону. Щоб це компенсувати, часто створюють псевдо-HDR рішення за допомогою інверсивного тонового відображення або генеративних мереж, але вони в повній мірі не репрезентують нелінійну форму справжніх HDR зображень, тому у реальних ситуаціях на чистих HDR зображеннях модель буде втрачати узагальнювальну здатність. Коли стисненні з HDR в SDR кадри передаються в нейромережеві детектори ключові піксельні властивості спотворюються вже на рівні інтенсивностей. По-перше, при простому обрізанні (clipping) або нормалізації піксельних значень втрачається інформація про екстремальні радіаційні ділянки, що особливо критично для детекторів. Також більшість

моделей використовують якірні рамки, оптимізовані через K-means на гістограмі розмірів об’єктів SDR наборів даних (COCO, VOC), якщо натомість датасет не було розмічено вручну. При зміні контрасту та локальних градієнтів у HDR адаптованих кадрах місця гарячих або холодних пікселів можуть зміщуватися від центрів рамки, що призводить до заниженої IoU та фрагментації предикцій уже на стадії NMS.

1.6 Постановка задачі

У сучасних системах автоматичного виявлення дорожніх об’єктів, що використовуються в рішеннях для допомоги водієві та візуального моніторингу дорожніх умов, висока різниця яскравості між тінями та світлими ділянками сцени ускладнює формування стійких ознак для згорткових нейромереж. Як було зазначено у попередньому розділі, традиційні оператори тонового відображення, як от глобальні нелінійні перетворення або локальні фільтри на основі гауссового розмиття дозволяють зменшити діапазон яскравості, проте їхнє застосування часто спотворює локальні контрасти та призводить до втрати важливих деталей на межах об’єктів. Водночас існуючі підходи до тонового відображення не враховують специфіку подальшої обробки кадру методом детекції, а зазвичай засновані на покращенні перцептивних якостей відповідно до моделі HVS, тому після перетворення динамічного діапазону значна частина критичних ознак втрачається або деформується, що знижує точність локалізації та класифікації транспортних засобів, пішоходів і дорожньої розмітки.

Отже, основна задача полягає в розробці адаптивної нейронної мережі для стиснення динамічного діапазону зображень, яка б підлаштовувалась під особливості термічних зображень. Для досягнення цієї мети необхідно провести комплексний аналіз існуючих методів тонового відображення, визначити їхні переваги та обмеження при роботі з високодинамічними

кадрами, а потім на основі отриманих результатів створити архітектуру, яка б враховувала обмеження існуючих систем. Власне, об’єктом дослідження є процес виявлення дорожніх об’єктів після застосування стиснення динамічного діапазону, а результатом – відлагоджена модель, що забезпечує підвищену точність детекції після тонового відображення.

Висновки до розділу 1

У першому розділі систематизовано фундаментальні аспекти високодинамічних зображень та методів їхнього стиснення. Аналіз фізичної природи високого динамічного діапазону та особливостей white-hot термальних зображень показав, що саме широкий динамічний діапазон надає змогу краще розрізняти тонкі теплові градації, особливо в складних умовах, що є важливим для систем дорожнього розпізнавання.

Проведено огляд класичних глобальних операторів тонового відображення. Показано, що вони дають передбачувану, але грубу нелінійну компресію, що часто призводить до постеризації у плавних градієнтах. Локальні методи дозволяють зберегти дрібні деталі завдяки врахуванню статистики яскравості локального оточення, проте вимагають значних обчислювальних ресурсів і схильні до гало-артефактів поблизу різких меж, тому у складних задачах варто використовувати більш сучасні методи на основі глибоких нейронних мереж.

Узагальнення отриманих результатів окреслює чіткий напрямок для подальших досліджень: вибір оптимальної стратегії просторово-варіаційного згортання діапазону, що зберігає глибину тону, що буде систематизовано в наступному розділі.

РОЗДІЛ 2 ВПРОВАДЖЕННЯ МЕТОДІВ ГЛИБОКОГО НАВЧАННЯ ДЛЯ СТИСНЕННЯ ДИНАМІЧНОГО ДІАПАЗОНУ ТА ДЕТЕКЦІЇ

2.1 Методи тонового відображення моделями глибокого навчання

Для стиснення динамічного діапазону наразі основними є підходи на основі глибоких нейронних мереж, оскільки вони здатні навчитися складним нелінійним відображенням інтенсивностей, що відтворюють властивості людського зору, і водночас адаптувуватись до конкретних завдань, як-от збереження дрібної текстури чи контрасту. Серед існуючих архітектур для тонового відображення було обрано дві принципово відмінні за філософією роботи моделі: TMO_CAN та UnCLTMO, які будуть описані у даному розділі. Обидві побудовані з урахуванням перцептивних метрик, орієнтованих на модель HVS та вимагають, аби вхідні HDR-зображення мали правильно розподілені значення яскравості. Проте, при роботі з термальними white-hot кадрами, де межі інтенсивності звично не наближаються до пікових значень SDR-діапазону, виникає питання: чи достатньо збалансованого стиснення для термальних зображень для того, щоб детектори могли чітко розрізняти краї об’єктів, адже моделі, що зосереджені на більшому згладжуванні тонів, можуть залишати границі об'єктів мало розрізненими від фону, тоді як ті, що підсилюють світлові ознаки, здатні більше підсилювати шум і можуть розмивати контури.

2.1.1 Аналіз архітектури TMO_CAN

Відповідно до цілей даного дослідження було обрано оператор TMO_CAN, що поєднує багатомасштабне представлення зображення у вигляді нормалізованої піраміди Лапласа з двома надзвичайно компактними контекст-

агрегаційними мережами (Context Aggregation Network) [7]. Узагальнену архітектуру моделі можна побачити на рисунку 2.1

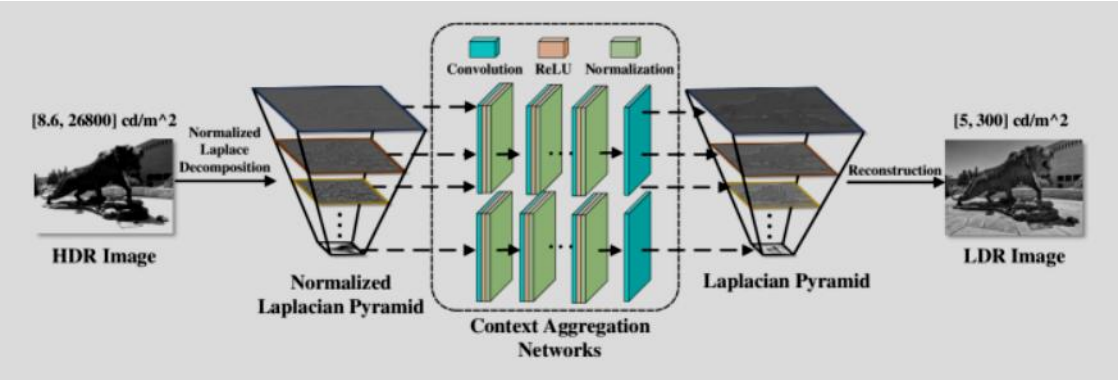

Рисунок 2.1 – Архітектура мережі TMO_CAN [7]

На першому етапі вхідне попередньо лінеаризоване HDR-зображення I розкладається за рекурсивною операцією [7]:

$$G^0 = I, G^\ell = REDUCE\left(G^{\ell-1}\right), L^\ell = G^\ell - EXPAND\left(G^{\ell+1}\right), \quad (2.1)$$

де REDUCE та EXPAND реалізують згортку гаусовим ядром і двократне зменшення/збільшення роздільності.

Коефіцієнти кожного рівня додатково нормалізуються:

$$\tilde{L}^\ell(p) = \frac{L^\ell(p)}{\theta + \sum_{q\in\mathcal{N}(p)}^{n} w(q)\cdot L^\ell(q)}, \quad (2.2)$$

де $L^\ell(p)$ – коефіцієнт Лапласового розкладу;

$\mathcal{N}(p)$ – локальне вікно навколо пікселя p;

$w(q)$ – гаусові ваги в цьому вікні;

$\theta = 10^{-3}$ – стаілізуючий член, який запобігає діленню на нуль і регулює розподіл значень піраміди.

Це усуває залежність від абсолютних величин і робить подальшу обробку масштабно-інваріантною. Отримана K-канальна тензорна репрезентація $\mathcal{P} = \tilde{L}^{\ell}_{\ell=0}{}^{L-1} \cup G^L$ подається на дві незалежні CAN: перша приймає усі високо- та середньочастотні рівні, друга – низькочастотний G^L канал.

Отриманий вихід передається до контекстно-агрегаційних модулів, що складається лише з чотирьох згорткових шарів із ядром $3x3$ без зсувів (bias = 0), але з експоненційним розширенням поля сприйняття за рахунок дилатацій d={1,2,4,1} [7]. Така конфігурація забезпечує одночасне охоплення локальних (1–3 пікселі) і глобальних (до 31 пікселя) контекстів без зменшення просторової роздільності. У трьох перших шарах використовується 32 фільтри, четвертий звужує тензор до одного каналу, безпосередньо формуючи прогнозовані коефіцієнти $\hat{P}^{\mathrm{l}}$. Між згортками застосовується адаптивна нормалізація:

$$AdaNorm(x) = \alpha x + (1\alpha)BN(x), \tag{2.3}$$

де α – навчуваний параметр (початково 0.9);

BN – пакетна нормалізація.

Такий підхід поєднує переваги ідентичного відображення й статистичної центрованості, стабілізуючи градієнт.

Активаційна функція після перших трьох шарів – Leaky ReLU, після чого відбувається перехід на четвертий шар, який не містить нелінійної активаційної функції, аби не спотворювати спектрально-частотну структуру високочастотних шарів.

Після прямого проходу обох CAN об’єднані оцінки пірамідних рівнів зливаються у фінальне SDR зображення:

$$\hat{I}_{SDR} = \sum_{\ell=0}^{L-1} EXPAND\left(\hat{P}^{\mathrm{l}}\right) + EXPAND\left(\hat{P}^{L}\right). \tag{2.4}$$

Варто зазначити, що результати даної моделі залежать від точності апроксимації маскувальних чутливостей для нормалізованої піраміди Лапласа, також можливе перетонування окремих надяскравих пікселів, якщо їхній радіус поширення перевищує прийомне поле четвертого шару.

2.1.2 Аналіз архітектури UnCLTMO

UnCLTMO (Unpaired Contrastive Learning Tone-Mapping Operator) реалізує перехід від HDR-термограми I_{HDR} до SDR представлення I_{LDR} без потреби у парних HDR/LDR зразках, використовуючи мінімальний, але цілеспрямований набір структурних компонентів – попередню криву компресії, генератор G з U-Net-топологією, ансамбль легких дискримінаторів D_k та багатокомпонентну функцію втрат [18]. Узагальнена архітектура мережі наведена на рисунку 2.2:

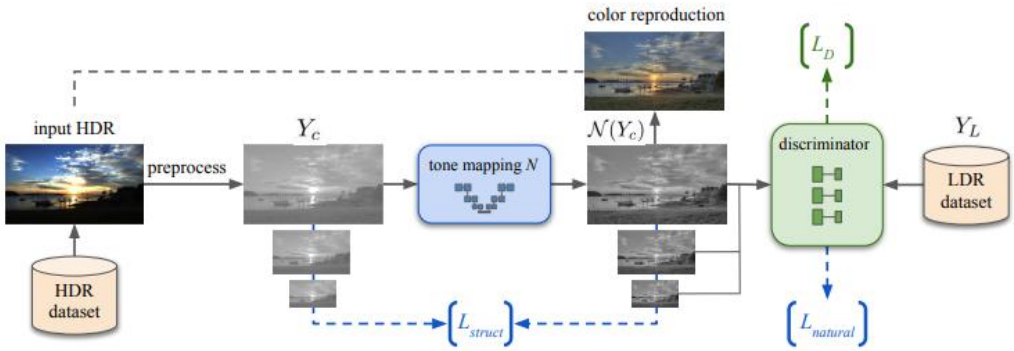

Рисунок 2.2 – Архітектура мережі UnCLTMO [18]

Спочатку вхідна карта яскравості Y приводиться до визначеного інтервалу [0,1] за допомогою адаптивної логарифмічної TRC-функції:

$$\hat{Y}(x) = \frac{log\left(\lambda \frac{Y(x)}{Y_{\max} + \varepsilon}\right)}{log(\lambda + \varepsilon)}, \tag{2.5}$$

де параметр α обчислюється шляхом мінімізації крос-ентропії між гістограмою $\hat{Y}$ і середньою гістограмою нативних LDR-кадрів.

Тепер розглянемо роботу генератора G (U-Net), що складається з чотирьох рівнів кодувальника – декодувальника із пропускними зв'язками (skip-connections) [19]. У кожному блоці кодувальника виконуються дві послідовні згортки з ядром 3×3, зсувом 1 (stride = 1) і доповненням країв (padding = 1), після кожної з яких слідує ReLU-активація: така конфігурація забезпечує компроміс між локальним рецептивним полем і кількістю параметрів, оптимально захоплюючи крайові та тональні ознаки на рівнях від 32 до 256 каналів. Після двох згорток у кожному рівні застосовується операція збільшення роздільності (upsampling) 2×2, що зменшує роздільність у два рази та дозволяє подвоїти кількість фільтрів на наступному рівні, збільшуючи зону сприйняття кадру до 6×6 пікселів на другому рівні та до 12×12 на третьому [18].

У звужувальному блоці (bottleneck) розміщено два послідовні блоки Conv-ReLU-BN-Conv-ReLU з 512 фільтрами, які згортають інформацію глобального контексту у 512-вимірний латентний простір, формуючи компактне представлення фонових структур.

Декодер симетрично відновлює роздільність за допомогою транспонованих згорток 2×2 зі зсувом 2, та об'єднанням з ознаками енкодера через пропускні з'єднання. На кожному рівні до активації енкодера додається її квадратний корінь $\sqrt{f}$ що подвоює розмірність перед наступною згорткою й покращує згладжування тональних переходів без створення смуг на перетворених кадрах. Після кожного блоку декодера застосовується пакетна нормалізація BatchNorm для стабілізації градієнтів, а в кінці шляху – 1×1

згортка з сигмоїдальною активацією, яка гарантує вихідні значення в діапазоні [0,1].

Ансамбль дискримінаторів $\{D_0, D_1, D_2\}$ реалізовано як три незалежні мережі з чотирьох шарів: перші дві – згортки 4×4 із 16 та 32 фільтрами і LeakyRELU, третій – 1×1 згортка без зсуву, останній – повнозв'язний шар із сигмоїдою на виході [18]. Використання легких дискримінаторів усуває нестабільності глибоких DCGAN-архітектур та прискорює навчання без шкоди для якості відтворення низькорівневих текстурних атрибутів LDR-сцен.

Цільова функція поєднує три складові [18]:

$$\mathcal{L} = \mathcal{L}_{SGAN} + \alpha\,\mathcal{L}_{struc} + \beta\,\mathcal{L}_{hist}, \tag{2.6}$$

де $\mathcal{L}_{SGAN}$ – квадратична змагальна функція втрат, що мінімізує $\chi 2$-дивергенцію між розподілами $G(Y_c)$ і нативних LDR-кадрі;

$\mathcal{L}_{struc}$ - структурно-зберігаюча втрата, яка обмежує відхилення локальних патчів 5×5 між $log\ Y$ та виходом $G(Y_c)$;

$\mathcal{L}_{hist}$ – перехресна ентропія між гістограмами Y і $G(Y_c)$, що зберігає глобальну тональну композицію сцени.

Таким чином, на кожному кроці обчислювального шляху відбувається чітко визначене перетворення: модель лінеаризує початковий діапазон, мінімізація, вибирає ступінь компресії, подвійні згортки витягують локальні патерни, максимізаційне агрегування акумулює контекст, транспоновані згортки відновлюють геометрію, пропускні з'єднання передають точні координати контурів, ансамбль дискримінаторів навчає мережу наслідувати статистику живих LDR-сцен, а багатокомпонентна втрата балансує між фотометричною правдоподібністю, структурною точністю та колірною узгодженістю.

2.2 Методи детекції

Оптика дорожньої сцени, що пройшла через алгоритм тонового відображення і втратила частину початкового динамічного діапазону, накладає подвійні вимоги на детектор: з одного боку, модель мусить залишатись інваріантною до неминучої зміни гістограми яскравості, з іншого – утримувати високий рівень локалізації дрібних об’єктів, які згорнулись в новому SDR-просторі.

Це є проблемою для детекторів, заснованих на підході з фіксованими рамками (якорями), які виникли раніше й спираються на жорстко зафіксований набір попередньо заданих рамок різних масштабів і відношень сторін, що підстилають кожен піксель ознаки. З таких рамок мережа прогнозує 4 параметри зсуву [5]:

$$t_x = \frac{x - x_a}{w_a},\ t_y = \frac{y - y_a}{h_a},\ t_w = \ln\left(\frac{w}{w_a}\right),\quad t_h = \ln\left(\frac{h}{h_a}\right), \tag{2.7}$$

де (x, y, w, h) – координати центру, ширини і висоти істинного об’єкта;

(x_a, y_a, w_a, h_a) – параметри вибраної рамки;

t_x , t_y – нормалізовані зсуви центру об’єкта відносно центру рамки;

t_w , t_h – нормалізовані зсуви центру об’єкта відносно центру рамки.

Зміщення оптимізуються разом із категоріальною перехресною ентропією у сумарній багатозадачій втраті, яка чутлива до збою, якщо динамічний діапазон зображення суттєво зменшено, оскільки контраст усередині якоря впливає на обчислення IoU й, отже, на вибір позитивних і негативних прикладів [6]:

$$\mathcal{L} = \mathcal{L}\text{cs} + \lambda \sum_{k \in \{x,y,w,h\}} |t_k - \widehat{t_k}|, \tag{2.8}$$

де $\mathcal{L}$cs – категоріальна перехресна ентропія, що оцінює правильність класифікації об’єкта;

t_k – цільове значення кожного з параметрів (x, y, w, h);

$\widehat{t_k}$,- відповідний прогноз мережі;

λ – ваговий коефіцієнт, що узгоджує вклад регресійної складової відносно класифікаційної;

Без’якірні моделі (YOLOv8, FCOS) відмовляються від попередніх рамок і тлумачать детекцію як прогноз чотирьох відстаней до меж прямокутника або теплової карти центрів, що одразу формує гнучку систему вибору позитивних прикладів і мінімізує гіперпараметричну залежність від масштабу [20]:

$$d = (l, t, r, b), \tag{2.9}$$
$$l = x - x_{\text{l}},$$
$$t = y - y_t,$$
$$r = x_r - x,$$
$$b = y_b - y,$$

де x_{l}, y_t – координати лівої та верхньої меж істинного прямокутника;

x_r, y_b – координати правої та нижньої меж істиного прямокутника;

l, t, r, b – відстані від центру (x,y) до відповідно лівої, верхньої, правої й нижньої сторін прямокутника;

вектор d повністю описує рамку без використання якорів.

Перехід від HDR-кадрів до SDR-версії зазвичай виконується оператором тонового відображення, що логарифмічно стискає інтенсивність високих світло-тіней, збільшуючи густину пікселів у середніх сірин, але водночас посилює квантування темних деталей.

Якірному детектору, що покладається на IoU-пороги τ, необхідно коригувати τ під нову статистику яскравостей [9]:

$$\mathcal{P} = \{\, i \mid IoU\big(A_i, B_j\big) \geq \tau\}, \quad \tau = \mu_{\text{IoU}} + \sigma_{\text{IoU}}, \tag{2.10}$$

A_i – i-й якір із зафіксованого набору;

B_j – j-й істинний прямокутник;

$IoU(A_i, B_j)$ – показник Intersection-over-Union між якірним та істинним боксами;

$\mathcal{P}$ – множина індексів якірних боксів, що вважаються позитивними (відповідають об’єкту);

τ – порогове значення IoU, обчислене як сума середнього та стандартного відхилення поточної статистики.

Без’якірні методи, що регресують відстані, уникають жорсткого порога й виявилися більш стійкими до втрати локального контрасту після тонового відображення, оскільки центр-пікселі обчислюються з пікового відгуку теплової карти, а не з порівняння з фіксованою сіткою якорів [13].

Саме тому для детекції об’єктів на зображеннях після тонового відображення важливо використовувати без'якірні детектори, серед яких FCOS та YOLOv8, які ми надалі і розглянемо.

2.2.1 Аналіз архітектури FCOS

FCOS (Fully Convolutional One-Stage detector) узяв за основу просту, майже сегментаційну ідею: якщо для кожного пікселя на багатомасштабній карті ознак навчити мережу напряму передбачати вектор відстаней до країв об’єкта й імовірність класу, то десятки чутливих гіперпараметрів і витрати на попереднє накладання якорів не потрібні [14]. Архітектуру FCOS представлено на рисунку 2.3:

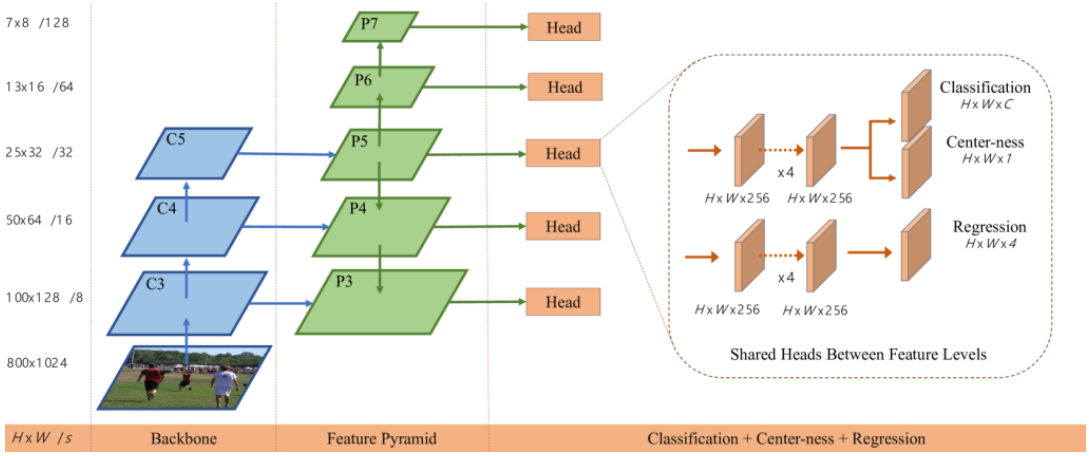

Рисунок 2.3 – Архітектура мережі FCOS[4]

FCOS послідовно перетворює кадр на набори пірамідальних ознак і далі на карти класифікації, показник центральності й регресії. Саме рисунок 2.3 з першого погляду демонструє три центральні зони – синя залишкова базова мережа ResNet-50 з рівнями C3, C4, C5, зелений блок пірамідальної мережі ознак (Feature Pyramid Network), що породжує карти P3…P7, та помаранчеву область спільної голови, підвішену над кожним рівнем FPN. У базовій мережі зображення спершу проходить початкову згортку й максимізаційне агрегування (max pooling), а далі потрапляє у блок C3 – тут об’єм даних має просторовий розмір 100 × 128 із зсувом 8, тобто кожен елемент C3 відповідає плитці 8 × 8 пікселів початкового кадру [14]. У структурі ResNet блок C3 містить кілька залишкових підблоків: у кожному підблоці дві згортки 3×3 з пакетною нормалізацією BatchNorm і ReLU обробляють ознаки, а потім незмінений вхід додається до виходу, утворюючи суму x+F(x) – така залишковість гарантує, що похідна по шляху 1 ніколи не обнулюється, і тому навіть глибокі каскади зберігають градієнт [14]. Після C3 іде C4: зсув тут стає 16, простір знову ділиться навпіл до 50 × 64, а кількість каналів подвоюється, що дозволяє моделі фіксувати ширший контекст, за рахунок чого краще

[4]Джерело зображення: https://www.researchgate.net/publication/378814105/figure/fig2/AS:11431281232087305@1711624167089/The-network-architecture-of-FCOS-C3-C4-and-C5-were-from-the-backbone-network-For.tif

фіксувати ознаки у сполучних ділянках між об’єктами. Блок C5 повторює ту саму архітектуру ще раз, знову ділячи простір і розширюючи канали. У результаті в кожному 25 × 32 елементі C5 зібрано інформацію з області понад 30 % кадру, і саме цю семантично глибоку карту потім передають у FPN [14].

Пірамідальна мережа ознак починається з бічних 1×1 згорток, що перетворюють C3, C4, C5 у вектори фіксованої товщини 256 каналів, далі запускається низхідний канал передачі ознак: C5 збільшує розмір згортки у два рази звичайним методом найближчого сусіда і поелементно підсумовує з обчисленою раніше бічною проєкцією C4, утворюючи проміжну карту, яку ще очищує 3×3 згортка, аби прибрати артефакти збільшення роздільності (upsampling). Процес повторюється з C3, тому на виході ми маємо рівні P3, P4, P5, кожен з яких тепер містить і дрібну геометрію нижнього шару, і сильну семантику верхнього, після чого на рівнях P6 і P7 йдуть дві послідовні згортки із зсувом 2.

Коли карти P3…P7 готові, кожна подається у спільну голову – це означає, що чотири згорткові блоки в кожній із трьох гілок мають одні й ті самі ваги незалежно від рівня P, що економить пам’ять і змушує мережу вчити масштабо-інваріантні фільтри [20]. Дані заходять у перший згортковий блок, залишаючись того ж просторового розміру H × W, і чотири рази проходять такий блок – глибина тут постійна 256 каналів, аби не втрачати виразність, але й не роздувати параметри. Після четвертого блоку гілка розгалужується: у класифікаційному кінці стоїть 3×3 згортка на C виходів (по одному логіту на клас), у гілці центральності (center-ness) – така сама згортка на один канал, а регресор видає чотири канали, що кодують логарифми відстаней до центру. Важливо, що саме логарифм гарантує чисельну стабільність: будь-який реальний вихід експонентою стає позитивним, отже мережа завжди може представити відстані різних величин.

Усі втрати поєднуються у загальну формулу [14]:

$$\mathcal{L} = \frac{1}{N_{pos}} \sum_{p \in P_{\text{pos}}} \mathcal{L}_{FL(p,y)} + \lambda_1 \mathcal{L}_{CIoU(\hat{\ell},\hat{t},\hat{r},\hat{b})} + \lambda_2 \mathcal{L}_{BCE(\hat{c},c*)}, \qquad (2.11)$$

де $\mathcal{L}_{FL(p,y)}$ – фокальна перехресна ентропія;

$\mathcal{L}_{CIoU(\hat{\ell},\hat{t},\hat{r},\hat{b})}$ – регресійна втрата;

$\mathcal{L}_{BCE(\hat{c},c*)}$– звичайна бінарна перехресна ентропія для центральності;

λ_1 та λ_2 зазвичай беруть 1.0. Нормування на N_{pos} вирівнює вплив неоднорідних пакетів (batch).

2.2.2 Аналіз архітектури YOLOv8

YOLOv8 – найновіша ітерація родини «You Only Look Once», презентована Ultralytics на початку 2023 року як універсальний фреймворк для класифікації, сегментації й детекції без попередньо заданих якорів [17]. Модель базується на комбінації компактного залишкового бекбону, піраміди багатомасштабних ознак і безякорної голови-регресора, що прямо прогнозує координати, об’єктність та вектор класів [17]. Архітектуру моделі YOLOv8 детально зображено на рисунку 2.4:

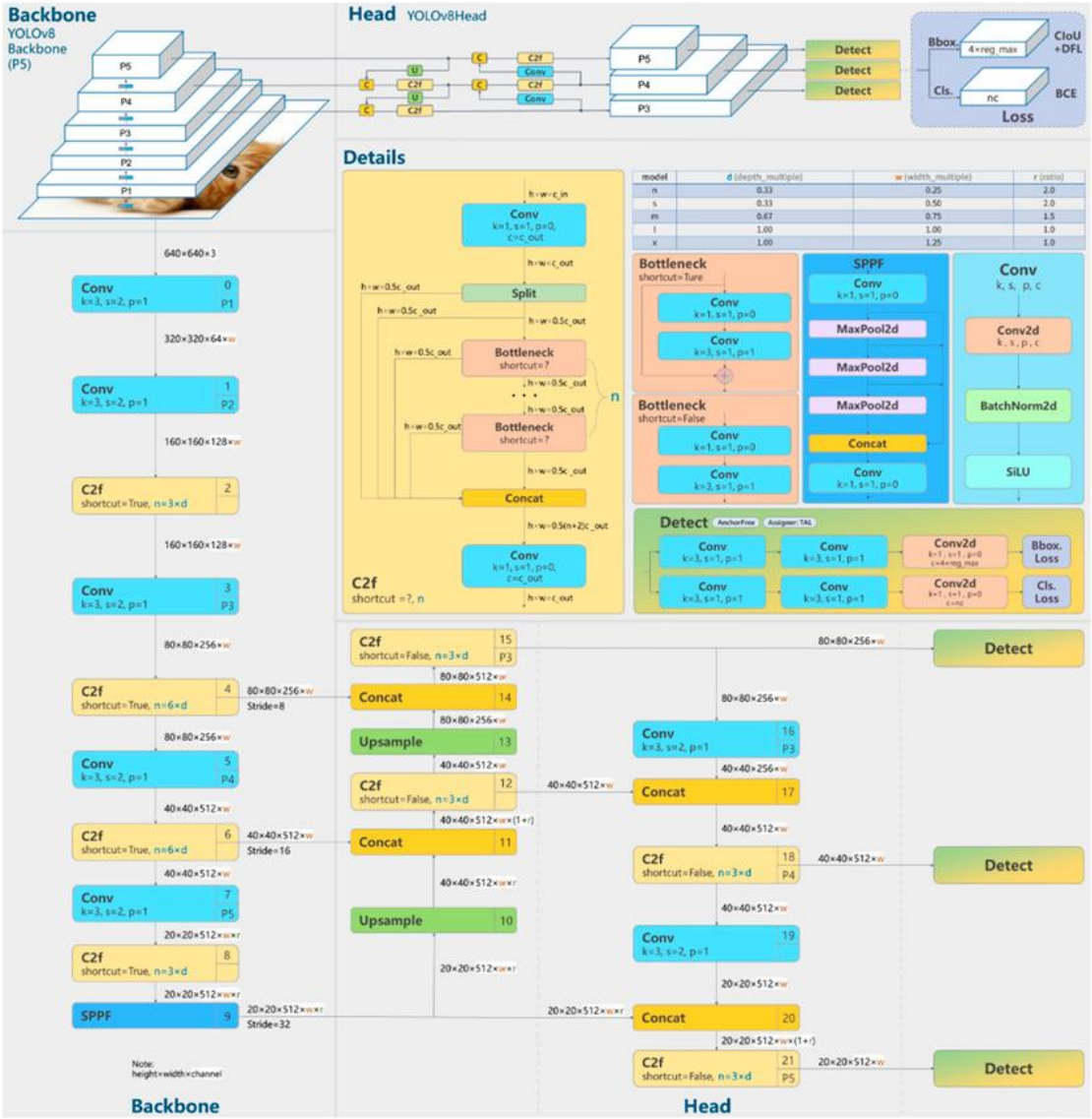

Рисунок 2.4 – Архітектура мережі YOLOv8[5]

Базова мережа починається зі звичайної згортки 3×3 із зсувом 2 (крок, на який ядро пересувається по зображенню, тому при значенні 2 висота й ширина карти одразу діляться навпіл, а рецептивне поле кожного фільтра збільшується без жодних нових ваг) [15]. Уже наступним елементом виступає C2f-модуль: половина каналів F_{id} проходить без додаткових перетворень, половина F_{split} стискається 1×1 згорткою й кілька разів проходить крізь ядра

[5]Джерело зображення: https://www.researchgate.net/publication/375373858/figure/fig7/AS:11431281255187625@1719410520922/The-architecture-and-visualisation-of-YOLOv8-Contributors-2023.tif

шарів звужувального блоку (bottleneck), після чого обидві гілки зливаються, що формально можна записати як:

$$F_{out} = Concat\left(F_{id}, BottleNeck^{\otimes^{n}(F_{split})}\right), \quad (2.12)$$

де F_{id} – неперетворена половина каналів;

F_{split} – стислий потік;

$\otimes^{n}$ означає послідовне застосування n звужувальних блоків.

Такий поділ дає змогу передавати чистий сигнал без спотворень, водночас економлячи до третини параметрів порівняно з попереднім C3, а залишковий зв'язок всередині звужувального блоку не дає градієнту згасати [11]. Коли карти ознак стають найглибшими, вихід передається до SPPF – Spatial Pyramid Pooling Fast. Три максимізаційне агрегування з ядром 5×5 і зсувом 1 накладаються послідовно, після чого результати об'єднуються і стискаються швидкою одиничною згорткою. Ідея полягає в тому, що перше агрегування ухоплює локальну структуру, другий дивиться вже на більшу область, а третій охоплює майже повний кадр. Так за одну операцію накопичується контекст різних масштабів. Далі ознаки різних рівнів збирає звужувальний блок, де збільшення роздільної здатності транспонованою згорткою переносить малі карти на рівень угору, а операція зменшення роздільної здатності із зсувом 2 – переносить великі карти на нижчий рівень, після чого C2f-вузли змішують канали в обох напрямах [15].

Кожен піксель карти звужувального блоку трактують як можливий центр, і мережа одразу регресує вектор:

$$\left(\sigma(x), \sigma(y), e^{\Delta l}, e^{\Delta t}, e^{\Delta r}, e^{\Delta b}, \sigma\left(p_{obj}\right), p_{cls}\right), \quad (2.13)$$

де σ – логістична функція, що стискає координати в [0, 1];

$e^{\Delta l}, e^{\Delta t}, e^{\Delta r}, e^{\Delta b}$ гарантують позитивність відстаней до меж.

Щоб голова збігалася стабільно, Ultralytics застосували SimOTA: для всіх пар передбачення – істина розраховують класичну втрату перетину за об’єднанням, до якої додається нормована відстань між центрами ρ та різницю пропорцій v [10]:

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2}{c^2} + \alpha v, \tag{2.14}$$

$$C = \mathcal{L}_{CIoU} + \mathcal{L}_{BCE}. \tag{2.15}$$

Це робить обмежувальні рамки чутливим до зміщень, які виникають, коли локальна гамма розширила один бік об’єкта.

Після отримання передбачення моделі координати чистяться за допомогою зваженого подавлення немаксимальних значень (Weighted-NMS): кожному перекривному боксу призначають вагу і усереднюють, тож слабкі, але погоджені гіпотези не губляться.

Висновки до розділу 2

У другому розділі сформовано комплексне методичне підґрунтя, необхідне для побудови адаптивної системи стиснення динамічного діапазону та подальшого виявлення дорожніх об’єктів. Здійснено огляд нейронних операторів тонового відображення. Оператор TMO_CAN, що поєднує нормалізовану піраміду Лапласа з компактними контекст-агрегаційними мережами, охарактеризований здатністю одночасно обробляти низько- й високочастотні компоненти сцени без втрати просторової роздільності. У свою чергу, UnCLTMO реалізує безпарне контрастивне навчання й дозволяє наближати статистику природних LDR-зображень без потреби у парних HDR/LDR прикладах.

Також було показано, що схеми з фіксованими рамковими прототипами (якірні) чутливі до змін гістограми, спричинених тоновим відображенням, оскільки коригування IoU-порогів і масштабу якорів стає нетривіальним. Натомість без’якірні архітектури FCOS та YOLOv8 оперують відстанями від центру пікселя або тепловими картами й тому демонструють вищу стійкість до локальних нелінійних перетворень контрасту. Окремо підкреслено, що обидві моделі сформовані на багатомасштабних пірамідах ознак, що дозволяє їм адаптивно обробляти інформацію.

Сформовані теоретичні основи у другому розділі забезпечують цілісне уявлення про реалізацію власної глибокої мережі для стиснення динамічного діапазону і її якісного порівняння з вже існуючими методами.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТОНОВОГО ВІДОБРАЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Огляд та попередня обробка набору даних

Вибір датасету відповідно до поставленої мети дослідження стикається з декількома проблемами: мають бути 8-бітні Ground Truth мітки для HDR-зображень, оскільки побудова несупервізованих моделей тонового відображення зазвичай передбачає застосування безджерельних метрик якості або перцепційних втрат, створення яких потребує чіткого розуміння статистичної природи піксельного розподілу та освітленості кадру. Наявність цільових зображень в стисненому динамічному діапазоні також полегшує створення і розробку власної архітектури, яка була б одночасно ефективною для відтворення тонового відображення за SDR міткою та перенесення теплової карти вхідного HDR зображення за R, G, B каналами у 8-бітний простір. Окрім цього для навчання на обраних моделях детекції, датасет має бути анотованим за стандартом COCO, YOLO, або KITTI, які також у свою чергу формулюють задачу детекції у різних площинах: COCO містить більш уніфіковану розмітку з зазначенням категорії (за офіційною стандартизацією усього 80 категорій) об’єкту разом з метаданими і координати обмежувальних рамок відносно верхнього лівого кута (x, y, width, height) у форматі JSON; у свою чергу KITTI стандартизація розроблена спеціально для розробки детекційних конвеєрів для задачі автономного водіння і використовує табличні текстові файлі, і кодує координати обмежувальних рамок у тривимірному просторі, що зазвичай потребує додаткової передобробки для створення відповідної індексації для подачі в модель. Створення власного датасету для урахування цих потреб є нереалістичним через високу вартість обладнання, саме тому довелось обирати існуючий датасет у відкритому доступі, що в першу чергу призначений для задачі детекції.

З урахуванням вищепоставлених поставлених вимог обрано FLIR Thermal Dataset for Algorithm Training, розробленого компанією FLIR Teledyne, яка займається створенням сенсорів для систем детекції для дорожнього руху, військової справи та патрулювання, для заохочення створення розробниками ф'южн методів детекції, тому містить розмічені знімки у високому та низькому динамічному діапазоні, як з термальної камери так і сценічні знімки в RGB просторі. Загалом датасет містить 10228 14-бітних термальних знімків, аналогічну кількість вручну розмічених 8-бітних термальних та RGB у роздільній здатності 640x512, покадрово отриманих із серії коротких (до 30 секунд) та 144-секундних відео із частотою 30 кадрів/секунду, причому 8-бітні зображення були отримані з 14 бітних шляхом нелінійного стиснення. Зображення були вручну відібрані і розбиті компанією-розробником на тренувальну і валідаційну вибірку – тренувальна містить 8862 зображення, тестова – 1366, робоча директорія датасету зображена на рисунку 3.1, у ньому містяться файли анотацій, папка містить «thermal_16_bit» 14-бітні термічні .tiff зображення, «thermal_8_bit», відповідно 8-бітні .png:

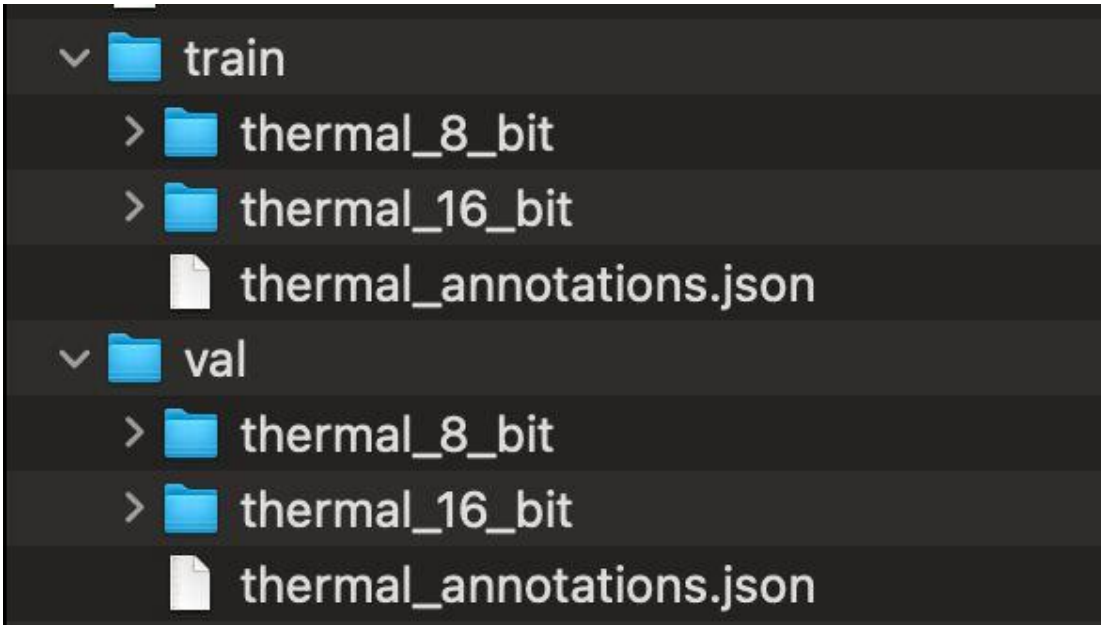

Рисунок 3.1 – Структура датасету

Спочатку проаналізуємо вхідні .tiff зображення на рівень інтенсивності пікселів та справжню бітову глибину для обрання підходу нормалізації та створення псевдо-RGB триканального входу на основі карти інтенсивностей для створення кастомного датасету, оскільки як зазначалось у 1 розділі – термальні зображення є одноканальними, а обрані моделі тонового відображення TMO_CAN та UnCLTMO приймають на вхід трьохканальні RGB зображення.

Спочатку відобразимо зображення та обрахуємо його рівні інтенсивності. Завантаження та відображення 14-бітного TIFF-файлу здійснюється за допомогою бібліотеки tifffile, котра через функцію tiff.imread повертає масив типу uint16, що містить значення інтенсивності для кожного пікселя в діапазоні до 16383. Для візуалізації використовується модуль matplotlib.pyplot: функція plt.imshow() приймає масив img16 і відображає його у тонах сірого з параметрами vmin та vmax, що відповідають мінімальному та максимальному значенню інтенсивності в масиві. Візуалізацію 14-бітного зображення продемонстровано на рисунку 3.2.

Слід зауважити, що хоча вихідні дані є 14-бітовими, їхнє відображення в Matplotlib відбувається у 8-бітному представленні за допомогою min-max нормалізації, тобто фактично бачимо масштабовану інтерпретацію 14-бітового діапазону. Як можемо побачити з обчислених рівнів інтенсивностей (мінімум – 6110, максимум – 7993, тобто 1846 унікальних рівнів інтенсивностей, кодують значно більше інформації, ніж 8-бітні зображення, проте будуть вирізнятись низьким рівнем контрасту, що прекрасно передає отримана візуалізація). За допомогою отриманих даних, використовуючи формулу (3.1) обчислено дійсне значення бітової глибини:

$$bit_{range} = log^2(max_{val} + 1), \tag{3.1}$$

де max_{val} – максимальне значення інтенсивності пікселей уздовж всього датасету.

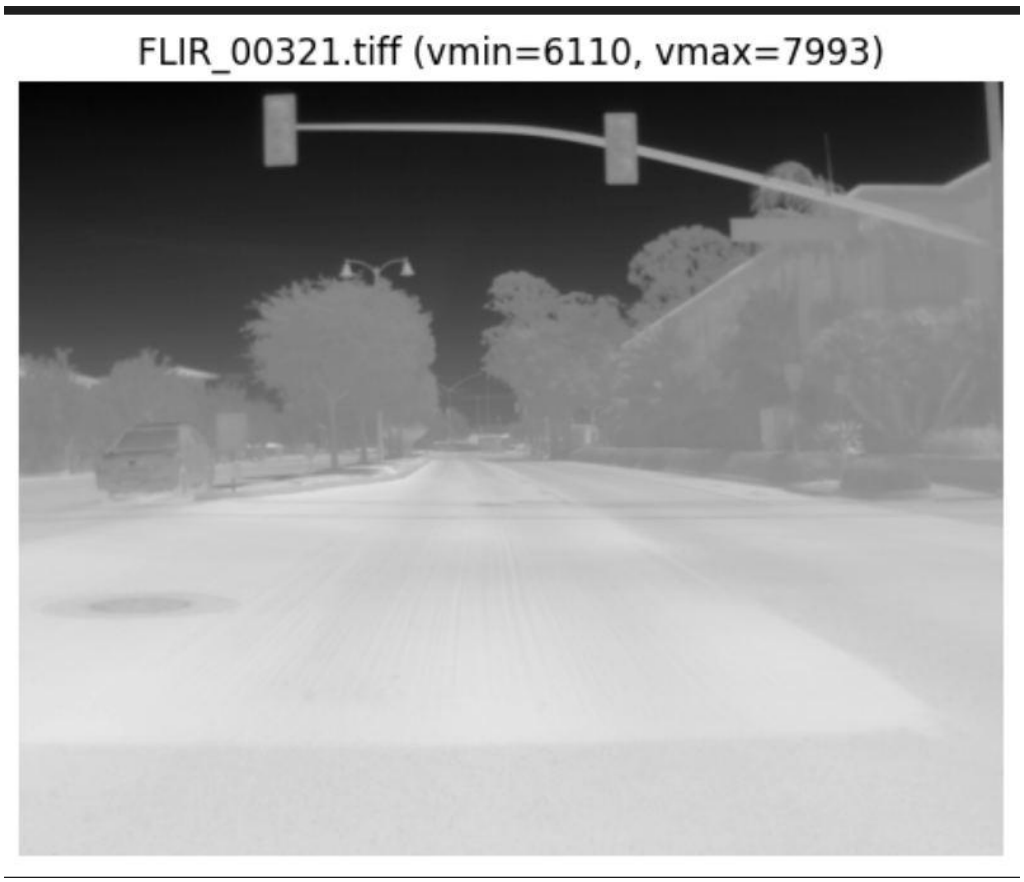

Рисунок 3.2 – Відображення вхідного 14-бітного кадру

У результаті отримано значення 13.36 біт, що ближче всього знаходиться до 14-бітного рівня, отже вхідні зображення будемо нормалізувати у діапазоні [0;1] для чого створено кастомний ThermalDataset.

Підготовка кастомного датасету ThermalDataset для моделей UnCLTMO та TMO_CAN починається з читання термальних TIFF-зображень 14-бітного формату як одноканальних, оскільки сенсори тепловізорів формують кадри у відтінках білого та представляють лише інтенсивність випромінювання без колірної інформації. Функція cv2.imread з прапором cv2.IMREAD_UNCHANGED завантажує зображення без змін у кількості

каналів, зберігаючи 14-бітну глибину, і повертає масив NumPy з розмірами H×W та відповідним dtype.

Нормалізація зображень у діапазон [0;1] виконується шляхом ділення значень пікселів на 16383 (максимальне значення $2^{14} – 1$ у 14-бітних даних), що забезпечує єдину шкалу інтенсивностей для всіх зображень і сприяє стабільності та збалансованості градієнтів під час навчання мережі. Така стратегія нормалізації є кращою за інші, наприклад, лінійне обрізання по мінімальному й максимальному значенню в кадрі, оскільки вона гарантує фіксований масштаб без залежності від змін контенту, що може викривляти динаміку градуювання міні-батчу й призводити до нестабільності навчання.

Оскільки оригінальні термальні кадри є одноканальними, а обидві моделі – UnCLTMO та TMO_CAN – спроектовані для прийому триканальних зображень, необхідно створити псевдо-RGB шляхом реплікації єдиної матриці інтенсивностей у три канали. Метод np.repeat(img[..., None], 3, axis=2) дублює кожне одноканальне значення по всіх каналах, формуючи тензор форми H×W×3, що відповідає вимогам моделі, водночас зберігаючи інформацію про інтенсивність вихідного кадру. Така стратегія реплікації є кращою за інші варіанти (наприклад, випадкову ініціалізацію додаткових каналів або застосування складної колірної мапи), оскільки вона не вводить артефактів чи відмінностей у статистиці пікселів та дозволяє модельній архітектурі зосередитися виключно на обробці теплових інтенсивностей.

Далі псевдо-RGB-зображення передаються в набір трансформацій, реалізований через torchvision.transforms, який відповідає за зміни роздільності та конвертацію у тензори. Зокрема, функція torchvision.transforms.Resize((H, W)) змінює розміри зображення до 256×256 без спотворення пропорцій за допомогою бікубічної інтерполяції. Після зміни розміру використовуємо torchvision.transforms.ToTensor(), яка конвертує NumPy-зображення (H×W×C, діапазон [0;1] після нормалізації) у тензор форми C×H×W з float32 значеннями в [0.0;1.0]. Усі ці трансформації групуються в

torchvision.transforms.Compose([...]) для послідовного застосування, після чого дані готові до входу до моделей.

3.2 Запропонована архітектура мережі тонового відображення

Враховуючи цілі дослідження, у попередньому розділі були розглянуті сучасні методи глибокого навчання для тонового відображення зображень, зокрема UnCLTMO та TMO_CAN, які базуються на мінімізації перцептивних метрик. Ці підходи суттєво залежать від коректного розподілу яскравості вхідного зображення: вони припускають, що інтенсивності охоплюють широкий діапазон яскравісних значень, що часто трапляється зі звичайними HDR-зображеннями. Однак у випадку термальних white-hot кадрів, інтенсивності зазвичай лежать у вузькому діапазоні та не досягають пікових значень, притаманних видимому світловому діапазону. Крім того, звичні оцінки контрасту, наприклад метрики на основі градієнтів чи просторових частот, не відповідають особливостям термальних даних: термальні об’єкти часто мають плавно змінювані температури без різких границь, і тому класичні критерії контрасту некоректно відображають їхню деталізацію. Щоб врахувати ці проблеми, запропоновано архітектуру TCNN (Thermal-Contrast Neural Network), яка поєднує можливість ефективного відтворення локальної контрастності з одночасним підсиленням інтенсивності термічних об’єктів за допомогою інтегрованого перенесення термальних вбудованих представлень (embedding) та контрастних характеристик істинних 8-бітних RGB-кадрів. Узагальнену архітектуру моделі TCNN представлено на рисунку 3.3:

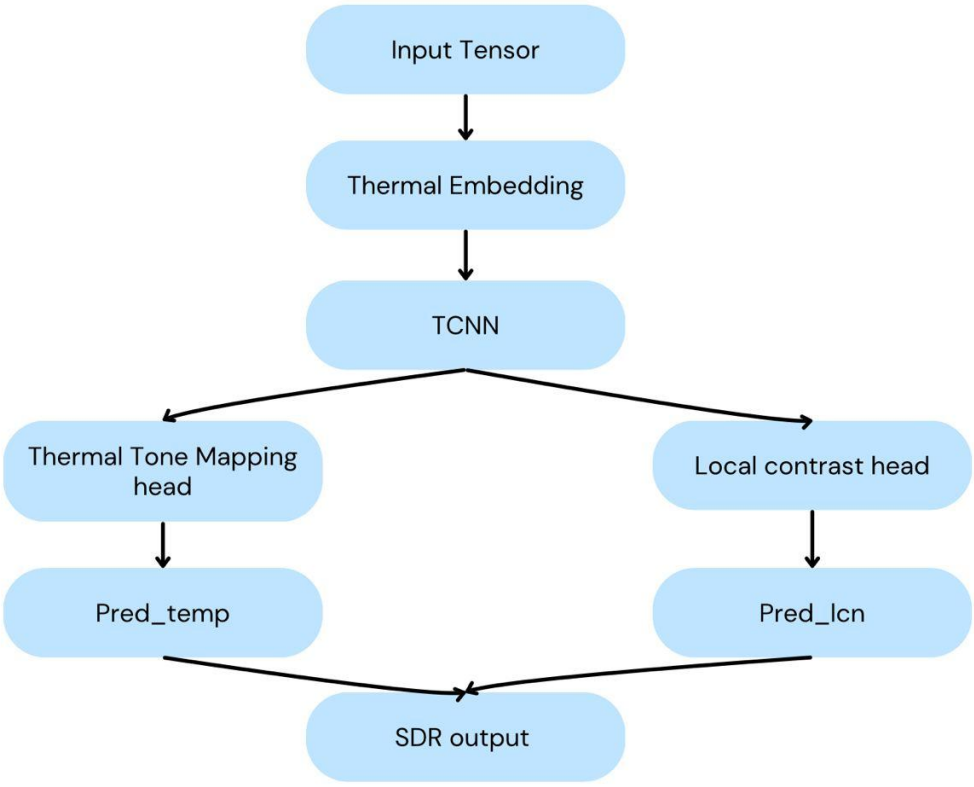

Рисунок 3.3 – Узагальнена архітектура мережі TCNN

Спершу вхідне 14-бітне термальне зображення проходить через модуль ThermalMultiChannelEmbedding, який для кожного вхідного пікселя $T_{raw(i,j)}$ обчислює десять значень:

$$E(i,j,c) = 0.5 \cdot sin\left(\frac{2\pi}{6} \cdot \frac{T_{raw(i,j)}}{D_c}\right) + 0.5, \quad (3.2)$$

де кожний D_c- випадковий коефіцієнт для каналу c.

Результатом є тензор розміру N×10×H×W, що уже містить відтінки локальних деталей у різних масштабах температури.

Отриманий 10-канальний тензор подається у згортковий кодувальник із двома групованими блоками. У кожному блоці є групова згортка 3×3 із 10 групами і зсувом 2. Перший шар приймає 10 каналів і для кожного застосовує окремі 3 ядра 3×3, тому вихід складається з 30 карт розміром $\frac{H}{2} \times \frac{W}{2}$. Після

кожної згортки йде групова нормалізація, що ділить 30 каналів на 10 груп по 3 та вирівнює їхню статистику, а далі – ReLU, що залишає лише позитивні активації. Другий блок повторює те саме на вхідних 30 каналах, знову зменшуючи просторовий розмір до $\frac{H}{4} \times \frac{\mathrm{W}}{4}$. Після другої ReLU кінцевий вихід кодувальника – тензор розміру $N \times 30 \times \frac{H}{4} \times \frac{W}{4}$.

Далі застосовується адаптивне усереднене зведення (AdaptiveAvgPool2d): кожний із 30 каналів згортається до одного числа m_c, середнього по всіх пікселях карти. Формально:

$$m_c = \frac{1}{\frac{H}{4} \cdot \frac{W}{4}} \cdot \Sigma_{i=1}^{\frac{H}{4}} \Sigma_{j=1}^{\frac{W}{4}} A_c(i,j), \tag{3.3}$$

де $A_c(i,j)$ – активація в проміжному каналі c.

В результаті маємо вектор N×30, що відображає середню інтенсивність кожного каналу. Цей вектор передається у двошаровий персептрон: перший лінійний шар 30×64 з ReLU, а потім 64×30. Вихідний 30-вимірний вектор ділимо на матрицю N×10×3: по 10 лінійних вихідних значень для кожного з трьох кольорів R,G,B. Після застосовуємо Softmax уздовж термальних каналів і отримуємо ваги $w_{c,i}$, сума яких по каналам $c = 1..10$ дорівнює 1. Тобто

$$w_{c,i} = \frac{exp(\lambda_{c,i})}{\Sigma_{d=1}^{10} exp(\lambda_{d,i})}, \tag{3.4}$$

де $\lambda_{c,i}$ – логіт для каналу c і кольору i.

Далі через оператор розгорнутого скороченого множення einsum робимо зважене сумування вбудованих представлень:

$$\mathrm{out}_{\mathrm{linear[n,i,h,w]}} = \Sigma_{\mathrm{i=1}}^{\frac{\mathrm{H}}{4}} \Sigma_{j=1}^{\frac{W}{4}} A_c(i,j), \tag{3.5}$$

де x[n,c,h,w] – вхідне вбудоване представлення;

$\mathrm{out}_{\mathrm{linear}}$ – RGB-вихід.

Щоб обмежити його до [0,1], застосовуємо sigmoid:

$$Pred_{RGB_{[nihw]}} = \frac{1}{1+\exp(-out_{linear[n,i,h,w]})}. \quad (3.6)$$

Таким чином формуємо тонмаповане 3-канальне зображення $pred_{rgb} \in (0,1)^{N\times 3\times H\times W}$.

Утім, оскільки тонове відображення використовує глобальні ваги, невеликі локальні деталі можуть бути втрачено або згладжено. Щоб компенсувати цю втрату, в архітектуру вбудовано локально-контрастну голову Local Contrast Normalization. Вона приймає на вхід істинне 8-бітне зображення і створює з нього локально контрастний варіант $pred_{LCN}$. Реалізація LCN полягає у поканальній згортці з ядром 3×3, де ваги фіксовані й рівномірно розподілені. Для кожного кольору і та пікселя (h,w) обчислюється локальне середнє:

$$local_{mean[n,i,h,w]} = \frac{1}{9} \cdot \Sigma_{u=-1}^{1}\Sigma_{v=-1}^{1} pred_{rgb[n,i,h+u,w+v]}, \quad (3.7)$$

Обидві гілки – тонового відображення та LCN – навчаються одночасно, використовуючи мультизадачну функцію втрат:

$$loss_{tone} = MSE(pred_{rgb}, targets_{rgb}), \quad (3.8)$$

$$loss_{lcn} = MSE(pred_{lcn}, targets_{rgb}). \quad (3.9)$$

Кожну складову зважуємо параметрами невизначеності:

$$precision_{tone} = exp(-log_{var_{tone}}), \quad (3.10)$$

$$precision_{lcn} = exp(-log_{var_{lcn}}), \quad (3.11)$$

$$weighted_{tone} = precision_{tone} \cdot loss_{tone} + \log var_{tone}, \quad (3.12)$$

$$weighted_{lcn} = precision_{lcn} \cdot loss_{lcn} + \log var_{lcn}. \quad (3.13)$$

І отримуємо функцію втрат:

$$loss = weighted_{tone} + weighted_{lcn}. \quad (3.14)$$

Параметри мережі та функція втрат оновлюються алгоритмом Adam з learning rate = 10^{-4}, коефіцієнтом регуляризації ваг = 10^{-6}. Так TCNN здатен водночас відтворювати бажану карту температур $pred_rgb$ і підсилювати дрібні локальні деталі через $pred_lcn$.

3.3 Метрики для оцінки якості тонового відображення

Оскільки тонове відображення HDR-зображень у SDR-формат виконує завдання нелінійного стиснення широкого діапазону інтенсивностей у межі 8 біт, важливо користуватися тими засобами оцінювання, які дозволяють не лише виявити абсолютні відмінності пікселів, а й перевірити, наскільки локальна структура й середній тон залишилися незмінними. У попередніх розділах було викладено, що HDR-кадри з 14–16 біт зазвичай нормалізуються до інтервалу [0,1], а результати тонового відображення переводяться у 8-бітний формат, тож тепер варто розглянути три основні метрики, які найчастіше використовуються для оцінювання якості перетворення з 14-бітного формату у 8-бітний: Structural Similarity Index (SSIM), Absolute Mean Brightness Error (AMBE) та Root Mean Square Error (RMSE). Кожна з цих метрик відіграє особливу роль: SSIM дозволяє зрозуміти наскільки тонмаповане зображення зберігає локальні контрасти й текстури оригіналу,

AMBE показує наскільки середня яскравість після стиснення змістилася відносно вихідного середнього, а RMSE оцінює загальну середньоквадратичну помилку піксель в піксель.

Structural Similarity Index (SSIM) метрика, яка більше імітує властивості людського сприйняття, орієнтуючись не на абсолютні відмінності інтенсивностей, а на збереження локальних структурних патернів кожного фрагмента зображення. Формально SSIM обчислюється у межах кожного маленького вікна розміру N×N пікселів за формулою:

$$SSIM(x,y) = \frac{(2\,\mu_x\,\mu_y + C_1)(2\,\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (3.15)$$

де x та y – два фрагменти зображень;

μ_x і μ_y – середні значення інтенсивності пікселів у цих фрагментах;

σ_x^2 та σ_y^2 – дисперсії фрагментів зображення;

σ_{xy} – коваріація між пікселями цих фрагментів;

C_1 і C_2 – невеликі стабілізаційні константи, які запобігають діленню на нуль (за умов порожніх або одноманітних областей).

Значення SSIM завжди лежить у діапазоні [0,1], де одиниця вказує на те, що локальна структура кадру після тону відповідає тій самій у HDR-оригіналі. Високе значення SSIM означає, що тонове відображення зберігло градації та контури, що є критично важливим для подальшої роботи моделей детекції, адже саме на цих структурних переходах базуються ознаки, які фіксують контури пішоходів чи транспортних засобів. Натомість низький SSIM свідчить про те, що локальні текстури були спотворені, що може призвести до заниженої чутливості детекторів до дрібних об’єктів.

Absolute Mean Brightness Error (AMBE), або похибка середньої яскравості, обчислюється як абсолютна різниця між середніми значеннями інтенсивностей нормалізованих HDR-пікселів h_i та тонмапованих SDR-пікселів s_i. Формула має наступний вигляд:

$$AMBE = |\bar{H} - \bar{S}|, \quad (3.16)$$

$$\bar{H} = \frac{1}{N} \sum_{i=1}^{N} h_i,$$

$$\bar{S} = \frac{1}{N} \sum_{i=1}^{N} s_i ,$$

де N – загальна кількість пікселів у кадрі.

Оскільки обидва середніх беруться із нормалізованих масивів [0,1], AMBE відображає зміщення глобальної яскравості. У white-hot термальних кадрах навіть невелике зміщення середньої (скажімо, 0,03) означає, що тонове відображення надто затемнює або, навпаки, пересвічує середню ділянку кадру, що може призвести до втрати контурів та заниження видимості малих температурних об’єктів. Тому бажано утримувати AMBE у межах не більше 0,02 для забезпечення реалістичної передачі середнього тону.

Root Mean Square Error (RMSE) обчислює середньоквадратичну різницю між кожним нормалізованим пікселем оригінального HDR-зображення та відповідним пікселем SDR. Формула RMSE наведена нижче:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (h_i - s_i)^2}. \quad (3.17)$$

Ця метрика обчислюється у тих самих нормалізованих одиницях [0,1], але надає більшу вагу пікселям із великими відхиленнями завдяки квадратичному піднесенню. Якщо, наприклад, оригінальне значення $\tilde{h}_i = 0.9$ було тонмаповано до $\tilde{s}_i = 0.6$, вклад у RMSE становитиме $\sqrt{(0.9 - 0.6)^2} = 0.3$, тоді як відхилення на темному фоні $\tilde{h}_j = 0.2$ до $\tilde{s}_j = 0.1$ дає вклад лише 0.1. Значення RMSE до 0.2,тобто внесок у середню похибку до 20 %, вважається прийнятним для white-hot кадрів, оскільки це відповідає не найбільшим втратам інтенсивності на критичних контурах. Якщо ж RMSE перевищує 0.3, це свідчить про те, що суттєва частина кадру була спотворена

в піксельній інтенсивності, що неприйнятно для алгоритмів детекції, чутливих до локальної різниці температур.

Слід зазначити, що у програмному рішенні для SSIM використано невелике локальне вікно 7×7 пікселів, оскільки більші вікна (11×11 або 15×15) призводять до зменшення чутливості до вузьких контурів, які можуть відповідати тісно прилягаючим теплим та холодним зонам. Одночасно не використовувались гаусові ваги, оскільки для біло-гарячих кадрів рівновага важливості кожного пікселя в малому сусідстві краще віддзеркалює умови задачі: однакові зміни на будь-якій із сімох позицій можуть означати зміну температури, на яку має реагувати детектор.

У свою чергу, для обчислення AMBE треба відзначити, що жодних додаткових зсувів чи масштабувань середнього не застосовувалось: достатньо порівняти $\tilde{h}_j$ та $\tilde{s}_i$ у природному масштабі [0,1], оскільки white-hot зображення вже на етапі нормалізації відображають усі крайні значення температури – від найхолоднішого пікселя (0) до найгарячішого (1). Будь-яке агресивне вирівнювання гістограми чи зміщення середньої призводило б до штучного пересвітлення або затемнення, яке в контексті детекції небажане.

3.4 Метрики для оцінки якості детекції

Так як після тонового відображення спотворюється піксельна та інтенсивнісні структури зображень, необхідно врахувати такі показники, які дозволяють кількісно оцінювати, наскільки точно модель детекції розмічує об’єкти за урахуванням того, що переходи між об'єктами будуть спотворені, але все ж модель повинна мати змогу правильно визначати об'єкти одного та різних класів. Зазвичай процес оцінювання детекції будується на основі порівняння передбачених мережевими алгоритмами bounding boxes із ground-truth анотаціями. Кожна передбачена рамка супроводжується ймовірністю

належності до певного класу, а також координатами її положення. Щоб визначити, чи виявлена рамка є вірною, вводять поріг перетину з оригінальною рамкою: якщо Intersection over Union між передбаченою рамкою та анотацією перевищує, зазвичай обирають, рівень 0.5, вважають, що спрацьовування є правильним. Для такої оцінки використано три основні метрики, які найчастіше використовуються у задачах об'єктної детекції: Precision, Recall та mean Average Precision (mAP).

Precision визначає відсоток правильно знайдених об'єктів серед усіх передбачених моделей. Зважаючи, що детектор може позначити частину шуму чи фонових фрагментів як об'єкт, Precision показує, наскільки чистими є передбачення. Формально Precision записується так:

$$Precision = \frac{TP}{(TP + FP)}, \tag{3.18}$$

де *TP* – кількість вірно класифікованих позитивних прикладів (тобто кількість передбачених боксів, що коректно відповідають реальним об'єктам);

FP – кількість хибно класифікованих як позитивні (тобто передбачення, які не перетинаються з анотацією або мають IoU менше за поріг).

На відміну від точності класифікації, у детекції треба враховувати не тільки фактичну наявність об'єкта, а й його просторові межі. Якщо модуль передбачив рамку, що перетинається з анотацією лише частково (IoU < 0,5), це вважається помилкою, і такий випадок відноситься до FP.

Recall, або повнота, у свою чергу, вимірює частку істинно існуючих об'єктів, які були знайдені детектором серед загальної кількості об'єктів у кадрі. Формула записується так:

$$Recall = \frac{TP}{(TP + FN)}, \tag{3.19}$$

де *TP* – кількість вірно знайдених (true positives);

FN – кількість хибно класифікованих як негативні (тобто реальні об’єкти, які не були знайдені або знайдені з IoU < поріг).

Оптимальною поведінкою є прагнення і до високої точності, й до високої повноти, проте в реальності між цими двома характеристиками часто існує компроміс: підвищення чутливості (Recall) може супроводжуватися зростанням хибних спрацьовувань (FP), а прагнення мінімізувати FP (покращити Precision) призводить до зростання FN (зниження Recall).

Для комплексної оцінки балансу між Precision і Recall вводять поняття точнісно-повнотної кривої (Precision-Recall curve), з якої вираховують площу під кривою (Average Precision, AP). Щоб узагальнити оцінювання у багатокласовій задачі, обчислюють AP окремо для кожного класу та беруть середнє. Саме це й є mean Average Precision (mAP). Формально для одного класу AP визначається як інтеграл Precision від Recall на відрізку [0,1]:

$$AP = \int^1 Precision(r)dr, \tag{3.20}$$

де $Precision(r)$ – значення точності за умов, що повнота дорівнює r.

На практиці обчислення AP здійснюють дискретно, нарощуючи Recall від 0 до 1 і беручи середнє значень Precision у контрольних точках, що відповідають змінам Recall. Нарешті, mAP у загальному випадку визначається як середнє зі значень AP для всіх C класів:

$$mAP = \frac{1}{C}\sum_{c=1}^{C} AP_c, \tag{3.21}$$

де AP_c – Average Precision для c-го класу;

C – загальна кількість класів.

Важливо зазначити, що при визначенні AP критичну роль відіграє вибір порогів IoU, за якими вважається, що передбачення є вірним. Зазвичай, поріг IoU приймають 0.5 (AP50) або усереднюють AP за кількома порогами 0.5:0.95

для більш жорсткої оцінки. У контексті оцінки якості детекції після тонового відображення важливо вказувати саме той набір порогів IoU, за якими проводиться обчислення mAP, оскільки стиснення діапазону може трохи зміщувати кордони контуру, що впливає на точність розрахунку IoU.

3.5 Аналіз результатів тонового відображення

Для того, щоб якомога об’єктивніше оцінити якість роботи кожної з трьох моделей тонового відображення – TCNN, TMO_CAN та UnCLTMO – спочатку доцільно розглянути піксельні характеристики отриманих SDR-зображень. Адже лише вивчення самих тонмапових результатів, без аналізу їх інтенсивнісних розподілів і локальних текстурних особливостей, не дасть повного уявлення про те, наскільки достовірно кожна модель відобразила контрасти вихідного HDR-кадру. Результати стиснення динамічного діапазону моделями TCNN, TMO_CAN та UnCLTMO наведено на рисунку 3.4.

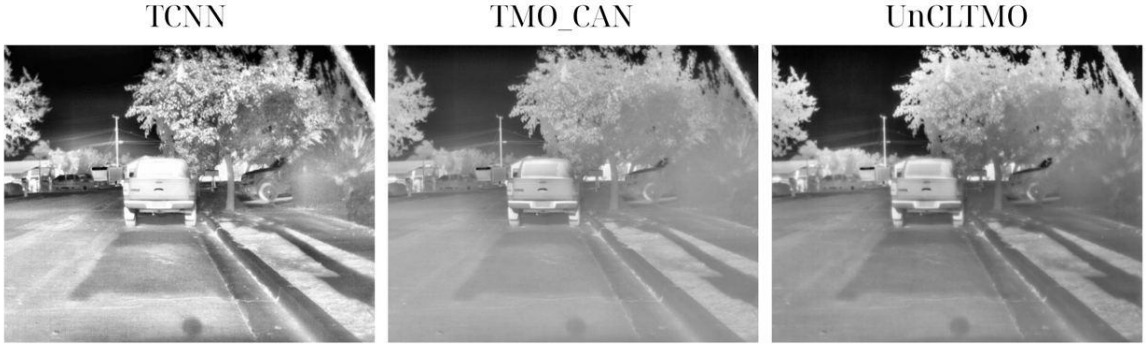

Рисунок 3.4 – Результат роботи моделей тонового відображення

З візуалізації помітно, що TCNN генерує відносно контрастний кадр із досить глибокими темними зонами та жорстким вирізненням гарячих об’єктів. У цьому результаті видно добре виражені контури автомобіля і сильні

переходи між асфальтом та тротуарною зоною, проте деталі у тінях, виявляються надто затемненими і втрачають текстурну виразність. Однак, кожна з деталей є більш деталізованою, ніж в інших моделях, особливо на віддалі: краще видно лінії електромережі – для TCNN виразно видно 3 лінії, тоді як для інших моделей – лише 2, краще виражені межі і деталі автомобілей, припаркованих у лівому куту та значно деталізованішим є бампер виїзджаючої машини зправа – гарно видно радіатор та фари, в той час як у інших моделей ці елементи зливаються.

У свою чергу, на тонмаповому зображені TMO_CAN спостерігається м’якіша передача темних і середніх тонів: контур автомобіля залишається чітким, але тіні під деревами не надто затемнені, проте модель значно посилює шум і сильно розмиває текстури з меншим температурним фоном, особливо це помітно у правому куті у кущах біля автомобілю. Але така більш згладжена передача тону також зумовлює і незначний кольоровий перехід між об'єктами, тому межі об'єктів на дистанції є не особливо розрізненими, так 2 автівки, що припарковані зліва можна візуально зчитати як 1, і як бачимо автомобіль на віддалі також зливається з тоном асфальту.

Найменше спотворень проявляється у UnCLTMO. Тут як темні, так і світлі області залишаються досить збалансованими: контур дерева, дорожньої розмітки та металу автомобіля добре промальовані, а відблиски фар не затирають навколишніх деталей. При цьому середні тони в асфальті виглядають природніше, зберігаючи плавний градієнт, і не утворюючи плями. Проте, аналогічно модель сильно посилює шум і тепловий фон автомобіля на віддалі дещо зливається з тоном асфальту, що ускладнить детекцію у сценаріях з щільним автомобільним потоком на автострадах.

Щоб структурно оцінити відображення тону для методів тонового відображення, побудовано гістограму інтенсивностей, зображену на рисунку 3.5.

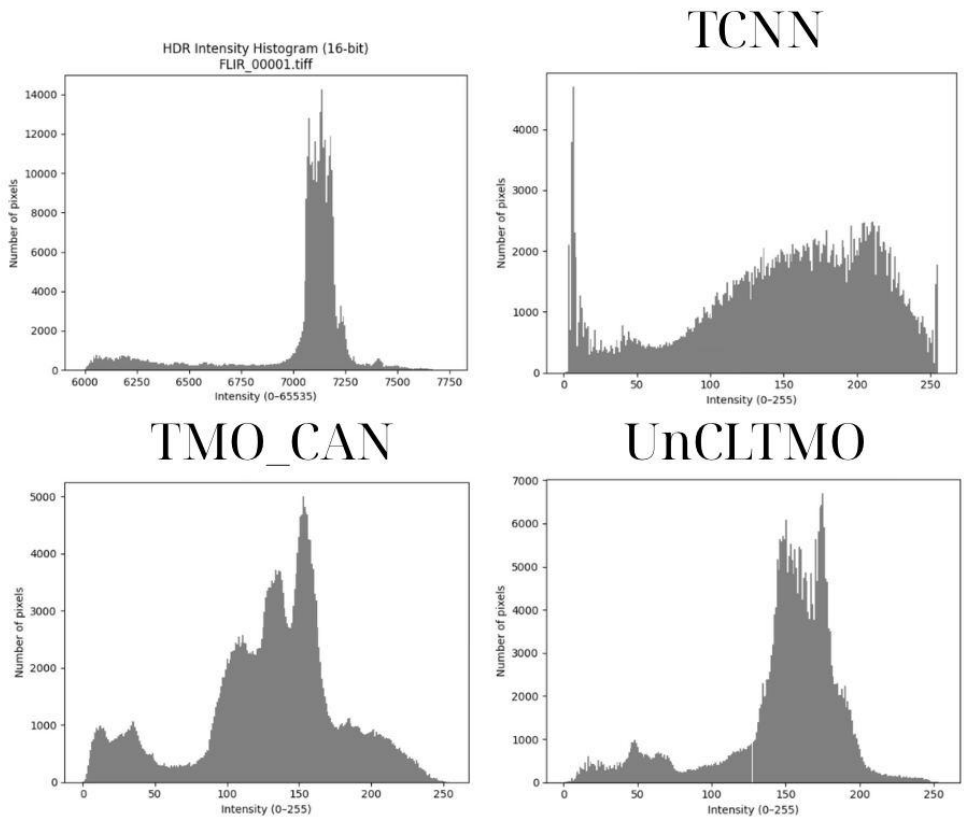

Рисунок 3.5 – Гістограми інтенсивностей отриманих зображень

Бачимо, що для HDR кадру більша частина пікселів знаходиться у діапазоні приблизно 7000–7200, тобто фон сцени кодується у середніх рівнях, із невеликою кількістю піків у найгарячіших областях та невеликою кількістю холодних об’єктів, що природно пояснюється тим, що більшість об’єктів на дорозі нагрівається виключно від енергії сонця, тому теплова різниця між об’єктами буде незначною, тому кадр є тусклим і при такому відображенні важливі деталі будуть не особливо вирізнятись

Праворуч у першому ряді зображено гістограму SDR-кадру, отриманого за TCNN – видно, найбільша кількість значень знаходиться у 0, тобто найгарячішій зоні, що відповідно, є протилежним до розподілу HDR-кадру, дана модель орієнтована на посилення текстурованих об’єктів з температурним фоном таким чином, щоб вони значно виділялись від інших,

тому мають різкий перехід до менш світлих значень (що пояснюється тим, що ми також тренували модель таким чином, щоб вона враховувала контраст Ground Truth 8 бітного зображення для кращого розрізнення об’єктів, тому температурний фон на виході зважувався втратою тону, що спричиняє такі розриви, але дає кращу розрізненість за рахунок насичення явних деталей), надалі до середніх тонів інтенсивність плавно зростає і також плавно падає, що дає змогу добре розрізнювати всі об’єкти зображення відразу, в тому числі із другорядними, оскільки маємо плавні температурні градації в середніх і низьких тонах, в яких лежить найбільша кількість пікселів, що відповідно описано вище на основі гістограми HDR-кадру, це дає глибину зображенню і дозволяє краще бачити об’єкти на великих дистанціях – в цьому і полягає ідея моделі.

У свою чергу, у другому рядку рисунка 3.4 подано гістограми TMO_CAN та UnCLTMO. У розподіленні TMO_CAN (лівий графік) видно більш плавні піки, що перекриваються в діапазоні 50–200, а також виокремлену групу темних пікселів у районі 0-20. Це означає, що TMO_CAN залишає більше середніх відтінків у SDR та забезпечує певну текстурність у тінях, але точка білої зони є менш виразною. У UnCLTMO видно, що гістограма зосереджена в інтервалі близько 50-200 із піком поблизу 150, а відносно темні та надто світлі значення займають менше місця. Такий розподіл підказує, що UnCLTMO здійснює більш збалансоване стискання динамічного діапазону, зберігаючи більшу частину середніх тонів і уникаючи надмірного завалювання у білі чи чорні зони, однак відповідно до особливостей дорожніх термальних зображень, де як уже було сказано, об’єкти не мають сильної різниці у температурах, буде мати меншу роздільну якість, ніж інші 2 моделі, проте краще збереження структури зображення.

Для попереднього аналізу того, наскільки добре було відображено границі об'єктів, які мають бути розрізнені детекторами, побудовано карту ентропії, яка відображає рівень різноманітності локальних текстур. Високі значення ентропії – жовті й зелені, свідчать про жирні текстури з багатьма

дрібними деталями, а низькі, що зображуються темно-фіолетовими – про однорідні області, детальну шкалу та отримані карти ентропії зображені нижче на рисунку 3.6:

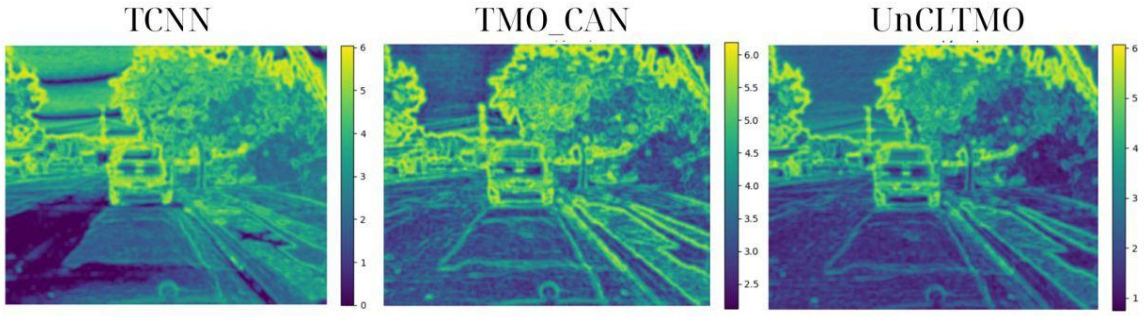

Рисунок 3.6 – Карта ентропії отриманих зображень

На карті TCNN видно, що найхолодніші фрагменти – ті, де ентропія найнижча, розташовані переважно на центральній ділянці дороги, де поверхня порівняно однорідна. Це свідчить про низьку ступінь локальних коливань яскравості в таких областях, що логічно відображає однаковість температурного фону без явної деталізації. Натомість уздовж контурів автомобіля, вздовж рейок і смуг на дорозі, а також у місцях із більш вираженою кроною дерев помітні жовто-зелені плями ентропії, що позначає високий рівень локальної інформаційної складності та багатство дрібних текстур. Це також підтверджує те, що TCNN виконує поставлену задачу на посилення контурів і текстурованих елементів: у цих зонах мережа суттєво підвищує контрасти, але внаслідок цього на фоні утворюються порожні ділянки з практично нульовою ентропією.

Середня карта, що відображає зображення, отримане з TMO_CAN, демонструє більш згладжений перехід кольорів: хоча модель, як видно також підсилює інформацію на краях, холодні ділянки тепер не опускаються до найнижчого рівня як в TCNN, а мають відтінок, близький до темно-синього чи зеленувато-синього. Це означає, що TMO_CAN залишає у темних зонах трохи

більше текстурної інформації, надаючи тіням насиченості середнього рівня навіть у тих місцях, де фон має бути порівняно однорідним. Зокрема, ділянки дороги та неба в центральних частинах карти не згасають до мінімального значення інформаційної різноманітності, а містять помірні коливання. Це також видно біля кордонів між стоп-лінією та асфальтом, де високі піки ентропії зміщені трохи вгору по шкалі порівняно з TCNN, але без різких стрибків.

Праворуч розташована карта UnCLTMO, в якій переважають фіолетово-сині відтінки з відносно небагатьма яскраво-жовтими плямами. Цей розподіл свідчить про те, що UnCLTMO прагне уникнути як повного вигорання тих ділянок, де повинна бути висока текстурність, так і глибоких провалів у плоских зонах. Це означає, що максимальні значення ентропії локалізовані лише на найгостріших контурах автомобіля та лініях на дорозі, тоді як фон і середні тональні переходи переважно мають помірні значення.

Оцінювання метриками якості тонового відображення, наведені в таблиці 3.1, підтверджують попередні візуальні спостереження.

Таблиця 3.1 – Результати тонового відображення

Модель	*Метрика*		
	SSIM	*AMBE*	*RMSE*
TCNN	0.696	0. 207	0.2893
TMO_CAN	0.737	0.144	0.2102
UnCLTMO	0.813	0.073	0.0944

Значення SSIM демонструють, що UnCLTMO (0.813) краще за інших моделей зберігає структуру вхідного зображення, тоді як TCNN незначним чином гірші результати (0.696). Аналогічно, AMBE для UnCLTMO (0.073) вдвічі менше, ніж у TMO_CAN (0.144), і втричі менше, ніж у TCNN (0.207), що свідчить про найменші відхилення середнього тону. RMSE найменший для UnCLTMO (0.0944), а найбільший для TCNN (0.2893), що підтверджує високу

рівномірність передачі інтенсивностей у UnCLTMO та суттєві, але не значні втрати інформації в TCNN.

3.6 Оцінка результатів детекції

Отримавши вихідні датасети тонмапових зображень з моделей TCNN, TMO_CAN, UnCLTMO, на кожному наборі було проведено детекцію моделями FCOS та YOLOv8 на 35 епохах для кожної моделі з однаковими параметрами навчання.

Для більш критичної оцінки результатів першочергово було візуалізовано результати детекції з прогнозованими обмежувальними рамками, червоним позначені істинні перекривальні рамки, зеленим – предиктовані. Роботу детектора FCOS зображено на рисунку 3.7.

Із візуалізації для всіх трьох моделей видно, що для TCNN отримується найбільш щільний набір передбачень: передбачені бокси (зелені рамки) отримано на кожному істинному об’єкті, охоплюючи і передній план, і віддалені автомобілі, хоча бокси іноді не накладаються на контурах істинних обмежувальних рамок, що менш виражено в результатах інших моделей через занадто інтенсивне підкреслення текстур у SDR-кадрі. Модель TMO_CAN, навпаки, демонструє трохи меншу кількість передбачень на відстані: бокси менш агресивно прилягають до найдрібніших об’єктів уздовж горизонту, але водночас рідше помиляються щодо площі автомобіля, оскільки тонове відображення моделю TMO_CAN, як було зазначено в минулому розділі, зберігає більше однорідного фону й середніх тонів, що полегшує FCOS-класифікатору ідентифікацію чітких контурів. UnCLTMO показує ще більш стримані передбачення: найвіддаленіші машини іноді зовсім пропускаються, а бокси найближчих автівок розташовані трохи далі від контурів – це свідчить про те, що UnCLTMO, хоча й зберігає збалансовану градацію тонів, водночас

позбавляє FCOS частини виражених контурів, необхідних для розпізнавання дрібних об’єктів на віддалі.

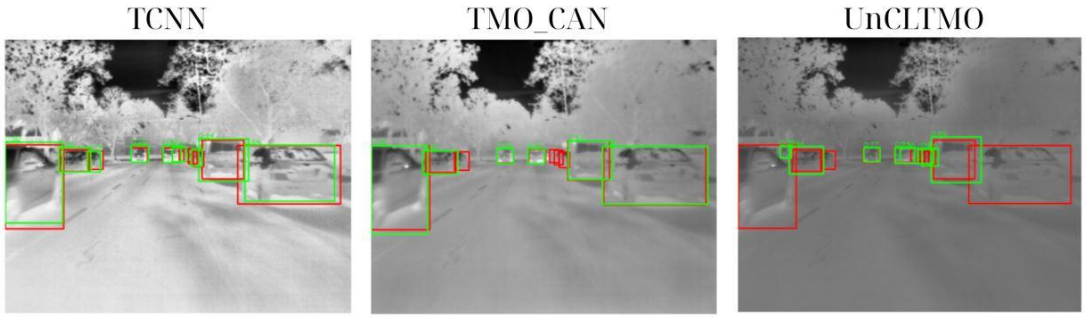

Рисунок 3.7 – Візуалізація детекції FCOS

Далі проведено детекцію моделю YOLOv8, приклад її результатів для більш віддалених об'єктів можна побачити на рисунку 3.8.

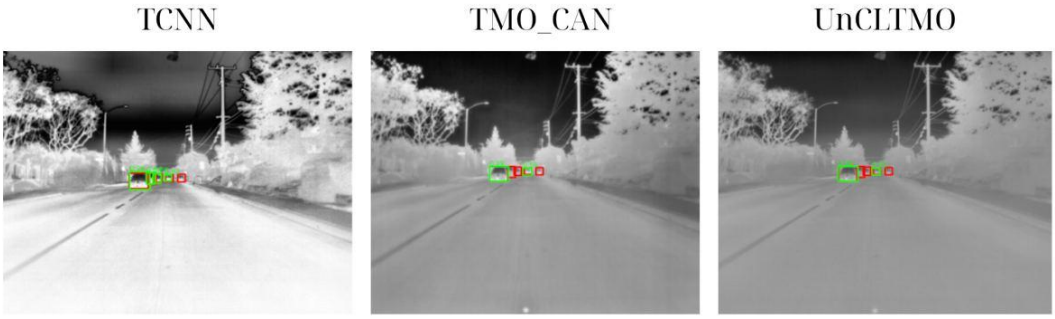

Рисунок 3.8 – Візуалізація детекції YOLO

Для отриманої візуалізації детекції YOLO зберігаються аналогічні тенденції, але в більш вираженому вигляді. На TCNN зображеннях YOLO виявляє основні машини на передньому плані й захоплює майже всі віддалені об’єкти, хоча в деяких випадках рамки трохи ширші, ніж у Ground Truth – що аналогічно зумовлено перенасиченням деталей після тонового відображення. На TMO_CAN, YOLO показує менше помилкових накладок і майже для всіх

детектованих випадків повністю перекриває істинну рамку, але й разом з тим інколи пропускає найвіддаленіші автомобілі, відображаючи лише ті, що не закриваються іншими машинами. На UnCLTMO YOLO продемонстрував аналогічні результати до TMO_CAN, але краще перекриває бокс автомобіля на задньому плані, оскільки має менше розмиття кадру (менший рівень шуму).

Після проведення візуального аналізу, було систематизовано. Нижче наведено оцінку якості детекції за обраними метриками для всіх комбінацій моделей (табл. 3.2).

Таблиця 3.2 – Результати детекції

Модель	*Детектор*	*Метрика*		
		mAP	*Precision*	*Recall*
TCNN	FCOS	0.386	0.401	0.368
TMO_CAN	FCOS	0.341	0.363	0.351
UnCLTMO	FCOS	0.334	0.333	0.314
TCNN	YOLO	0.294	0.296	0.279
TMO_CAN	YOLO	0.265	0.268	0.253
UnCLTMO	YOLO	0.239	0.247	0.228

Отже, підтверджено закономірність: моделі, які під час тонового відображення максимально підсилюють локальну різницю інтенсивностей, забезпечують кращі результати детекції як для FCOS, так і YOLO, завдяки наявності виразніших контурів та більш насичених текстур. Запропонована модель TCNN отримала найкращі результати за всіма метриками, тому усі цілі дослідження були досягнуті.

Висновки до розділу 3

У третьому розділі детально описані етапи підготовки та попередньої обробки даних, що лягли в основу експериментальної частини роботи. Обгрунтовано вибір датасету, викладено процедуру створення псевдо-RGB зображень та здійснено аналіз рівнів інтенсивності вхідних кадрів, визначено реальну бітову глибину та виправдано обрану стратегію нормалізації й реплікації каналів, що забезпечили узгодженість вхідних даних для обох розглянутих обраних моделей тонового відображення.

Наступним етапом було впровадження програмної реалізації власної архітектури у порівнянні з існуючими підходами. Запропоновано побудову багатокального представлення температурної карти, яка краще враховує термальну природу зображення і температурні переходи. Для покращення візуального сприйняття і розрізрізнення переходів між об’єктами дорожньої сцени, запропоновано механізм локально-контрастної голови як доповнення глобальної температурної гілки, яка буде забезпечувати більш явні переходи між об’єктами що дійсно містять власні температурні показники і фоном. Для балансування ваг гілок застосована і описана багатозадачна втрата з невизначеними вагами, щоб модель не на надавала явної переваги стратегії підвищення контрасту.

Для кількісного та якісного оцінювання тонового відображення обрано метрики структурної подібності та квадратичних похобик, результати яких доповнені аналізом побудованої карти ентропії для побудови більш обширного теоретичного базису для подальшого проведення детекції та її оцінки. У заключення дослідження проведено детекцію на отриманих зображеннях у стисненому динамічному діапазоні з обраних та власної моделі, побудовано візуалізацію та охарактеризовано вплив кожного з підходів на виявлення об’єктів в різних умовах сцени. Отримані результати підтвердили

покращення результатів детекції у підході, запропонованому в дослідженні, що відповідає поставленій меті роботи.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі здійснюється оцінка ключових характеристик програмного продукту, призначеного для створення власної демонстративної мережі стиснення динамічного діапазону і аналізу впливу методів тонового відображення на результати детекції.

Також розглянуто кілька варіантів технічної реалізації системи, з метою вибору найефективнішого підходу до розробки. Це дозволяє досягти кращої якості кінцевого продукту та оптимізувати економічні показники проєкту. Для цього було використано метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це підхід, що дає змогу оцінити реальну цінність розробки програмного продукту з урахуванням його функціональних можливостей. Основна мета ФВА — виявити шляхи зниження витрат завдяки впровадженню більш ефективних технічних рішень, а також досягти балансу між функціональністю системи та необхідними ресурсами.

Аналіз базується на економічних, технічних та архітектурних даних. Його етапи включають: визначення послідовності розробки системи, обчислення загальних витрат і кількості робочих годин, ідентифікацію джерел витрат та фінальний розрахунок загальної вартості створення програмного рішення.

4.1 Постановка задачі проектування

У даній роботі метод функціонально-вартісного аналізу (ФВА) застосовується для проведення техніко-економічної оцінки процесу розробки власної демонстративної мережі тонового відображення і аналізу впливу методів тонового відображення на результати детекції. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на функціонування всієї системи, важливо, щоб кожен компонент задовольняв загальні вимоги до ефективності, точності та зручності використання. Тому аналіз зосереджено на функціональних можливостях майбутнього програмного продукту.

Технічні вимоги до програмного продукту:

- підтримка роботи на персональних комп’ютерах із базовими технічними характеристиками;
- висока швидкість обробки та прогнозування даних;
- можливість зручного масштабування та обслуговування;
- мінімізація витрат на впровадження та підтримку програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту для аналізу впливу моделей тонового відображення на детекцію. Вона охоплює аналіз і реалізацію ключових компонентів, що впливають на ефективність та точність моделі.

Виходячи з цієї функції, можна виділити такі підфункції:

- F_1 – вибір мови програмування,

- F_2 – вибір методу реалізації алгоритмів,
- F_3 – вибір середовища розробки.

Кожна з цих функцій має кілька можливих варіантів реалізації:

Функція F_1:

а) Python;

б) C++.

Функція F_2:

а) застосування готових бібліотек для побудови моделей;

б) написання алгоритмів власноруч.

Функція F_3

а) Visual Studio Code;

б) Google Colab

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рис. 4.1.

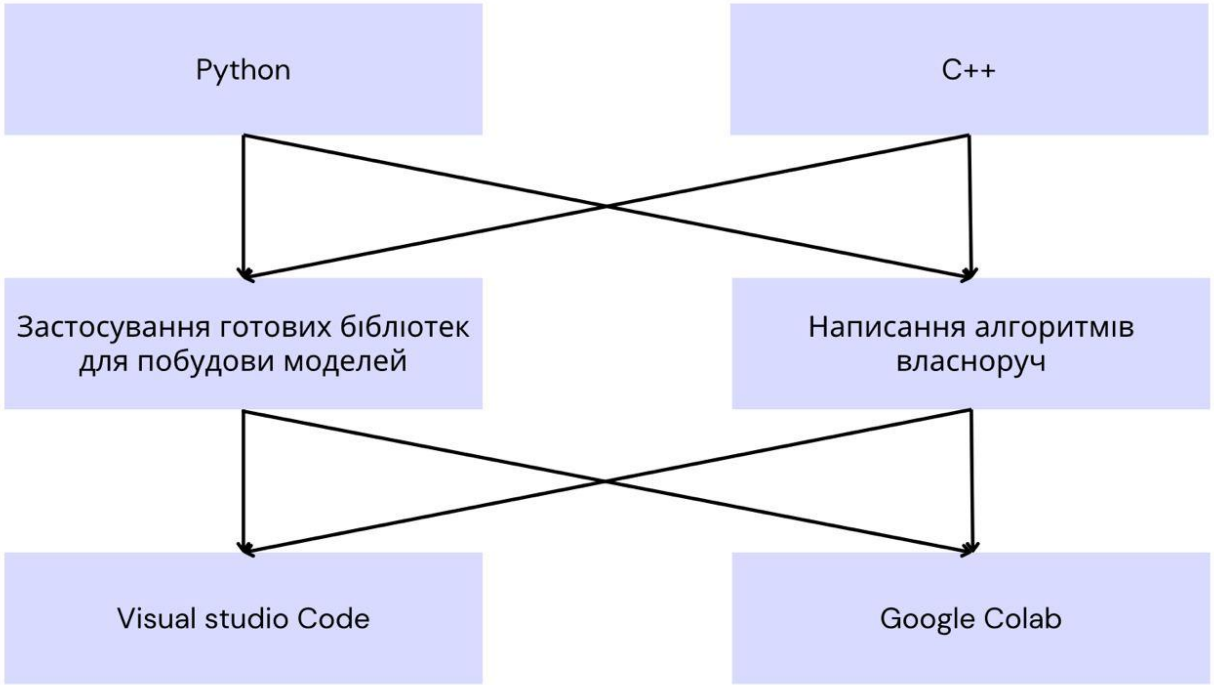

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в табл. 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	*А*	Проста мова для освоєння і використання, містить велику кількість бібліотек для машинного навчання та доступної документації.	Інтерпретована мова програмування, тому має нижчу швидкість виконання коду.
F_1	*Б*	Компільована мова програмування, тому має вищу швидкість виконання.	Має складніший синтаксис та в доступі міститься менше готових рішень для задач машинного навчання.
F_2	*А*	Швидша та перевірена реалізація програмного продукту. Пропонує широку спільноту для обговорення помилок/порад по імплементації.	Обмежена гнучкість.
F_2	*Б*	Можливість гнучкого написання програмного продукту під поставлену задачу. Допомагає повністю освоїти та зрозуміти принципи роботи алгоритму.	Потребує значної кількості часу та знань. Існує великий ризик помилки.

Кінець таблиці 4.1

Функції	Варіанти реалізації	Переваги	Недоліки
F_3	А	Зручне автодоповнення коду за допомогою влаштованих розширень. Збереження результатів після припинення взаємодії	Потребує обчислювальних ресурсів.
F_3	Б	Безкоштовний доступ до GPU/TPU. Працює в хмарному середовищі. Інтеграція з Google Drive.	Обмежений час сеансу. Скидання усіх результатів після припинення роботи з середовищем.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1.

Перевагу надано мові Python як простішій та доступнішій для реалізації задач машинного навчання. Варіант Б доцільно відхилити через складність у використанні.

Функція F_2.

Існуючі бібліотеки для машинного навчання є надійними та вже перевіреними, тому логічно обрати готові рішення. Написання власних алгоритмів (варіант Б) займає більше часу та потребує глибших технічних знань, тому його відхилено.

Функція F_3.

Вибір середовища розробки не є критичним, оскільки обидва варіанти мають свої переваги. У процесі розробки доцільно використовувати і Visual

Studio Code, і Google Colab, комбінуючи зручність локальної роботи з можливостями хмарних обчислень.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3 a,$$
$$F_1\text{а} - F_2\text{а} - F_3\text{б}.$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі наведених вище даних визначено ключові параметри, які будуть використані для розрахунку коефіцієнта технічного рівня програмного продукту.

Для оцінки ефективності програмного продукту обрано такі основні параметри:

X_1 – потенційний обсяг програмного коду;

X_2 – обсяг пам’яті, необхідний для обробки та збереження даних;

X_3 – час, необхідний для навчання моделі;

X_4 – точність моделі.

Для кожного параметра визначено три рівні: гірше, середнє та краще. Значення підібрані з урахуванням практичних вимог до продуктивності, ресурсів і якості розробки. Усі значення наведено в таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Потенційний обсяг програмного коду	*X*1	кількість рядків коду	3000	1800	1000
Необхідний обсяг пам’яті	*X*2	ГБ	10	5	2
Час навчання моделі	*X*3	год	2	1	0.5
Точність моделі	*X*4	%	80	87	95

На основі таблиці 4.2 побудовано графіки зміни кожного параметра, які наведені на рис. 4.2 – 4.5.

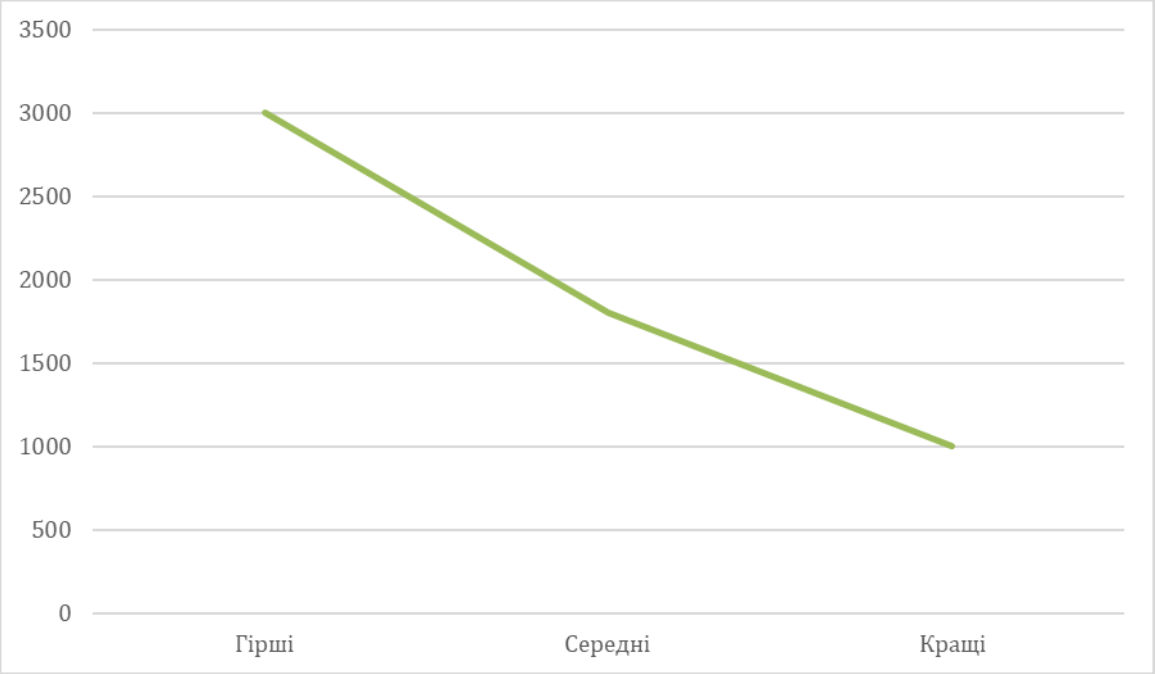

Рисунок 4.2 – X_1, потенційний обсяг програмного коду

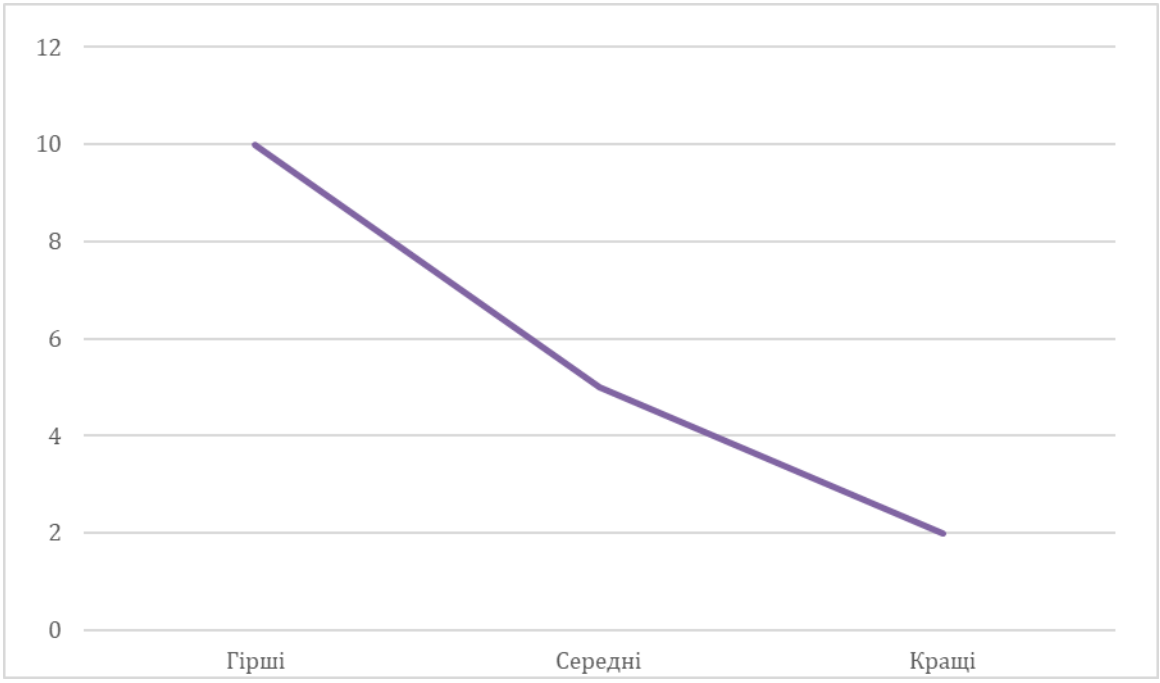

Рисунок 4.3 – X_2, необхідний обсяг пам’яті

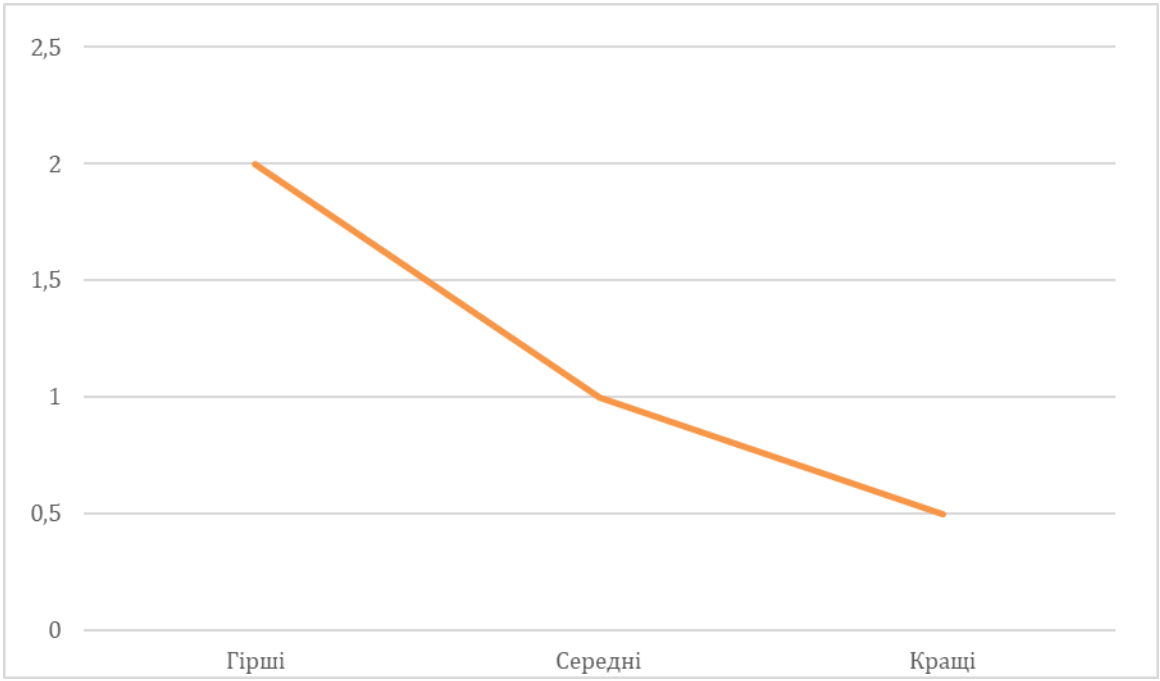

Рисунок 4.4 – X_3, час навчання моделі

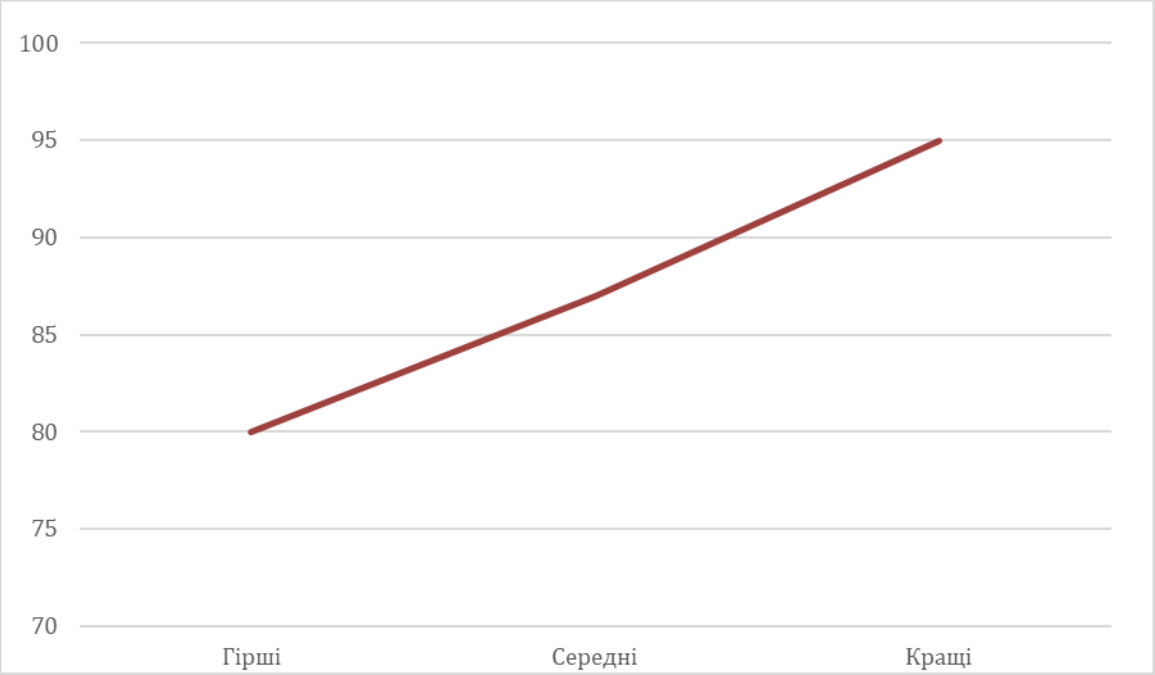

Рисунок 4.5 – X_4, точність моделі

4.4 Аналіз експертного оцінювання параметрів

Після глибокого аналізу та обговорення кожен з експертів оцінив ступінь важливості основних параметрів, що впливають на ефективність програмного продукту для аналізу впливу методів тонового відображення на результати детекції. Ці оцінки базуються на тому, наскільки кожен параметр сприяє досягненню основної цілі – створенню точної, швидкої та оптимізованої системи.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводила експертна комісія з 7 фахівців. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;

- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування експертів

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			
*X*1	Потенційний обсяг програмного коду	кількість рядків коду	1	2	2	1	1	1	2	10	-7,5	56,25
*X*2	Необхідний обсяг пам’яті	ГБ	3	4	3	3	4	3	4	24	6,5	42,25
*X*3	Час навчання моделі	год	2	1	1	2	2	2	1	11	-6,5	42,25
*X*4	Точність моделі	%	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки ступеня достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T, \tag{4.3}$$

сума відхилень по всім параметрам повинна дорівнювати 0.

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{4.4}$$

Розрахунок коефіцієнта узгодженості за наступною формулою:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 197}{7^2(4^3-4)} = 0{,}754 > W_k = 0{,}67 \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67. Скориставшись результатами ранжування, проведено попарне порівняння всіх параметрів і результати внесено у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	2	3	4	5	6	7		
*X*1 і *X*2	<	<	<	<	<	<	<	<	0,5
*X*1 і *X*3	<	>	>	<	<	<	>	<	0,5
*X*1 і *X*4	<	<	<	<	<	<	<	<	0,5
*X*2 і *X*3	>	>	>	>	>	>	>	>	1,5
*X*2 і *X*4	<	>	<	<	>	<	<	<	0,5
*X*3 і *X*4	<	<	<	<	<	<	<	<	0,5

Ступінь переваги *i*–го параметра над *j*–м визначається числовим параметром a_{ij} і визначається за формулою:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складено матрицю А=$\| a_{ij} \|$.

Для кожного параметра розраховано коефіцієнт вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 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*	
	*X*1	*X*2	*X*3	*X*4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
*X*1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
*X*2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
*X*3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
*X*4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначено рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X_2 (необхідний обсяг пам’яті), X_3 (час навчання моделі) та X_4 (точність моделі) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X_1 (потенційний обсяг програмного коду) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується наступним чином (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.11}$$

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	*Варіанти реалізації функції*	*Параметри*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F1	А	X1	1500	7	0,16	1,12
F2	А	X2	2,8	8	0,28	2,24
F3	А	X3	1,7	3	0,2	0,6
	Б	X4	93,5	8	0,36	2,88

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначено рівень якості кожного з варіантів:

$$K_{K1} = 1{,}12 + 2{,}24 + 0{,}6 = 3{,}96;$$
$$K_{K2} = 1{,}12 + 2{,}24 + 2{,}88 = 6{,}24$$

Помітно з розрахунків, кращим виявився варіант 2, для нього коефіцієнт технічного рівня має більше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки програмного продукту спочатку необхідно розрахувати його трудомісткість. Кожен із розглянутих варіантів реалізації включає два окремі етапи: розробку проєктної частини програмного продукту та створення програмної оболонки.

Перше завдання за рівнем новизни належить до групи А, друге – до групи Б. За складністю алгоритмів завдання 1 віднесено до групи 1, тоді як завдання 2 – до групи 3.

Для реалізації першого завдання використовується довідкова інформація, а для другого – інформація у вигляді вхідних даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\mathrm{Т_О} = \mathrm{Т_Р} \cdot \mathrm{К_П} \cdot \mathrm{К_{СК}} \cdot \mathrm{К_М} \cdot \mathrm{К_{СТ}} \cdot \mathrm{К_{СТ.М}}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП, $K_П$ – поправочний коефіцієнт, $K_{СК}$ – коефіцієнт на складність вхідної інформації, $K_М$ – коефіцієнт рівня мови програмування, $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм, $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 30 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.75. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 30 \cdot 1.75 \cdot 1 \cdot 0.9 = 47{,}25 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_p = 27 людино-днів, $K_П$ = 0.9, $K_{СК}$ = 1, $K_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19{,}44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їхню трудомісткість:

$$T_{\text{I}} = (47{,}25 + 19{,}44 + 3{,}96 + 19{,}44) \cdot 8 = 720{,}72 \text{ людино-годин,}$$
$$T_{\text{II}} = (47{,}25 + 19{,}44 + 6{,}24 + 19{,}44) \cdot 8 = 738{,}96 \text{ людино-годин}$$

Найбільш високу трудомісткість має варіант II.

У розробці беруть участь один програміст з окладом 22000 грн. та один аналітик в області даних з окладом 18000 грн. Визначено середню зарплату за годину за формулою:

$$С_{\text{ч}} = \frac{\text{М}}{T_m \cdot t} \text{грн,} \qquad (4.14)$$

де M – місячний оклад працівників, T_m – кількість робочих днів на місяць, t – кількість робочих годин в день.

$$С_{\text{ч}} = \frac{22000+18000}{2 \cdot 18 \cdot 8} = 138{,}89 \text{ грн.}$$

Розрахунок заробітньої плати відбувається за формулою:

$$С_{\text{ЗП}} = С_{\text{ч}} \cdot T_i \cdot К_{\text{Д}}, \qquad (4.15)$$

де $C_ч$ – величина погодинної оплати праці програміста, T_i – трудомісткість відповідного завдання; $K_д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{ЗП} = 138{,}89 \cdot 720{,}72 \cdot 1.2 = 120120{,}96 \text{ грн},$$
$$\text{II. } C_{ЗП} = 138{,}89 \cdot 738{,}96 \cdot 1.2 = 123160{,}985 \text{ грн}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{ВІД} = C_{ЗП} \cdot 0.22 = 120120{,}96 \cdot 0.22 = 26426{,}61 \text{ грн}$$
$$\text{II. } C_{ВІД} = C_{ЗП} \cdot 0.22 = 123160{,}985 \cdot 0.22 = 27095{,}42 \text{ грн}$$

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:

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

З урахування додаткової заробітньої плати:

$$C_{ЗП} = C_Г \cdot (1 + K_З) = 52800 \cdot (1 + 0.2) = 63360 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0.22 = 63360 \cdot 0{,}22 = 13939{,}2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 24000 грн.

$$C_A = K_{TM} \cdot K_A \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 24000 = 8400 \text{ грн.},$$

де K_{TM}– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; K_A – річна норма амортизації; $Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуються за формулою:

$$C_P = K_{TM} \cdot Ц_{ПР} \cdot K_P = 1.4 \cdot 24000 \cdot 0.08 = 2688 \text{ грн},$$

де K_p – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (Д_K - Д_B - Д_C - Д_P) \cdot t_З \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.4 = 745{,}6 \text{ години},$$

де $Д_K$ – календарна кількість днів у році; $Д_B$, $Д_C$ – відповідно кількість вихідних та святкових днів; $Д_P$ – кількість днів планових ремонтів устаткування; t – кількість робочих годин в день; K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуються за формулою:

$$C_{ЕЛ} = T_{ЕФ} \cdot N_C \cdot K_З \cdot Ц_{ЕН} = 745{,}6 \cdot 0{,}3 \cdot 0{,}5 \cdot 9{,}43 = 1054{,}65 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу; $K_З$ – коефіцієнт зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуються за формулою:

$$C_H = Ц_{ПР} \cdot 0.67 = 24000 \cdot 0{,}67 = 16080 \text{ грн.}$$

Тоді річні експлуатаційні витрати розраховуватимуться за формулою:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н} \quad (4.16)$$

$C_{ЕКС}$ =63360 + 13939,2 + 8400+ 2688+ 1054,65 + 16080= 105521,85 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ} = 105521,85 / 745,6 = 141,53 \text{ грн/год.}$$

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:

$$С_{М} = С_{М-Г} \cdot Т \quad (4.17)$$

I. C_M = 141,53 · 720,72 = 102003,5 грн.,

II. C_M = 141,53 · 738,96 = 104585 грн.

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0,67 \quad (4.18)$$

I. C_H = 120120,96 · 0,67= 80481,04 грн.,

II. C_H = 123160,985 · 0,67= 82517,86 грн.,

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н}, \quad (4.19)$$

I. $C_{ПП}$ = 120120,96 + 26426,61 + 102003,5 + 80481,04 = 329032,11 грн,

II. $C_{ПП}$ = 123160,985 + 27095,42 + 104585+ 82517,86 = 337359,27 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Коефіцієнт техніко-економічного рівня розраховано за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.20)$$

$$K_{ТЕР1} = 3,96 / 329032,11 = 1,20353 \cdot 10^{-5},$$
$$K_{ТЕР2} = 6,24 / 337359,27 = 1,84966 \cdot 10^{-5}$$

Найбільш ефективним є другий варіант реалізації програмного продукту з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 1,84966 \cdot 10^{-5}$.

Отже, у результаті проведеного функціонально-вартісного аналізу програмного комплексу, що розробляється, встановлено, що з-поміж двох варіантів реалізації, відібраних після попереднього оцінювання, оптимальним є саме другий. Цей варіант передбачає використання мови програмування Python, реалізацію алгоритмів за допомогою готових бібліотек, а також розробку у хмарному середовищі Google Colab. Такий підхід забезпечує користувачеві зручний інтерфейс, швидке впровадження функціоналу та доступність інструментів для ефективної роботи з програмним продуктом.

Висновки ро розділу 4

У даному розділі було здійснено повний функціонально-вартісний аналіз програмного продукту з урахуванням економічних, технічних та архітектурних показників. У рамках цього аналізу визначено та системно оцінено основні функції програмного комплексу (вибір мови програмування, методів реалізації алгоритмів і середовища розробки), а також встановлено ключові параметри, що їх характеризують (обсяг коду, необхідний обсяг пам’яті, час навчання моделі, точність роботи).

На підставі отриманих результатів було обрано оптимальний варіант реалізації програмного продукту, який передбачає використання мови Python, готових бібліотек для побудови моделей та хмарного середовища Google Colab.

ВИСНОВКИ

У дипломній роботі розроблено та експериментально перевірено адаптивну нейронну мережу для стиснення динамічного діапазону термальних зображень із метою підвищення точності детекції дорожніх об’єктів. Результатом дослідження стало побудова власної архітектури TCNN, яка поєднує кроки передобробки HDR-даних із багатомасштабною обробкою локальних і глобальних характеристик кадру, що дозволяє зберегти важливі контури та текстурні деталі, критичні для подальшої роботи детекторів.

Перший розділ визначив фундаментальні аспекти високого динамічного діапазону, описано і обгрунтовано ключові складові white-hot теплових знімків, також оглянуто класичні глобальні й локальні оператори тонового відображення і зазначено, що вони часто викликають артефакти або заниження локальних контрастів, що негативно позначається на якості детекції, тому для вирішення поставленої проблеми треба застосовувати методи машинного навчання.

У другому розділі представлено огляд сучасних моделей тонового відображення – TMO_CAN та UnCLTMO, а також розглянуто без’якірні детектори FCOS і YOLOv8.

Третій розділ присвячено практичній реалізації: налаштовано кастомний датасет, проведено тренування моделей TCNN, TMO_CAN і UnCLTMO, а потім виконано детекцію за допомогою FCOS і YOLOv8. Показано, що запропонована архітектура TCNN забезпечує найвищі метрики. Запропонована мережа TCNN у поєднанні з FCOS дала змогу досягти найвищого значення mAP@0.50 = 0,386 серед усіх проведених експериментів, що на 13 % перевищує результат TMO_CAN (mAP = 0,341) і на 15 % – UnCLTMO (mAP = 0,334).

У четвертому розділі виконано детальний функціонально-вартісний аналіз реалізації програмного комплексу: обгрунтовано вибір мови реалізації

Python, а також розгортання у хмарному середовищі Google Colab із оптимізацією витрат часу та обчислювальних ресурсів. Встановлено, що оптимальний варіант розробки допускає швидке впровадження й подальшу масштабованість рішення у вбудованих системах .

У свою чергу, подальше вдосконалення моделі можливе за кількома напрямками: створення багатоканальної інтеграції термальних кадрів з видимим спектром (RGB) у рамках єдиного модуля тонового відображення та об’єднання інформації з двох спектральних діапазонів, застосовуючи механізм уваги, дозволить досягти більш точного розділення об’єкту-фону та підвищить стійкість системи в складних умовах. Окрім цього, апаратна інтеграція також підвищить результати моделі, оскільки з відеокамери буде можливо точно отримати функцію її відгуку, що підвищить точність кодування температурної карти зображень. Більш складними, але і більш ефективними у порівнянні з вищеописаними пропозиціями також буде власна розмітка датасету і використання нативних кадрів у низькому динамічному діапазоні, оскільки гілка локального контрасту використовує стиснені до 8-біт зображення, які містять контраст вже спотворений нелінійними перетвореннями. Однак, найважливішим напрямом, який буде задавати вектор подальшого дослідження є підбір кращої стратегії багатозадачного навчання з епістемічною невизначеністю, що підвищить узгодженість між гілками тонового відображення та локально-контрастної нормалізації, та підвищить як метрики якості тонового відображення, особливо структурної подібності, так і точність детекції.

ПЕРЕЛІК ПОСИЛАНЬ

1. ASWF Deep Dive: OpenEXR Origin Story – Part 1. Academy Software Foundation, 2021 – URL: https://www.aswf.io/news/aswf-deep-dive-openexr-origin-story-part-1/ (дата звернення: 18.04.2025).
2. Dalal N., Triggs B. Histograms of Oriented Gradients for Human Detection. Proceedings of CVPR ’05. IEEE, 2005. – С. 886–893 – URL: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf (дата звернення: 15.04.2025).
3. Girshick R., Donahue J., Darrell T., Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. arXiv preprint arXiv:1311.2524, 2013 – URL: https://arxiv.org/abs/1311.2524 (дата звернення: 20.04.2025).
4. High Dynamic Range (HDR) Images FAQ. HDRsoft, 2024 – URL: https://www.hdrsoft.com/resources/dri.html (дата звернення: 18.04.2025).
5. Kawale S., Thorat S. Faster R-CNN and YOLOv3: A General Analysis between Popular Object Detection Networks. Preprint, 2024 – URL: https://www.researchgate.net/publication/373941669_Faster_R-CNN_and_YOLOv3_a_general_analysis_between_popular_object_detection_networks (дата звернення: 8.05.2025).
6. Law H., Deng J. CornerNet: Detecting Objects as Paired Keypoints. arXiv preprint arXiv:1808.01244, 2018 – URL: https://arxiv.org/abs/1808.01244 (дата звернення: 09.06.2025).
7. Luo B., Fan K., Wei C. Perceptually Optimised Deep HDR Tone Mapping. Computers & Graphics, 2023 – DOI: 10.1016/j.cag.2023.03.012 – URL: https://www.sciencedirect.com/science/article/abs/pii/S0097849323001632 (дата звернення: 2.05.2025).
8. Reinhard E., Stark M., Shirley P., Ferwerda J. Photographic Tone Reproduction for Digital Images. Tech. Rep. UUCS-02-001, University of

Utah, 2002 – URL: https://www.cs.utah.edu/docs/techreports/2002/pdf/UUCS-02-001.pdf (дата звернення: 23.04.2025).

9. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of CVPR 2016. IEEE, 2016 – URL: https://arxiv.org/abs/1506.02640 (дата звернення: 09.06.2025).
10. Redmon J., Farhadi A. YOLO8: Better, Faster, Stronger. arXiv preprint arXiv:1612.08242, 2016 – URL: https://arxiv.org/abs/1612.08242 (дата звернення: 09.06.2025).
11. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767, 2018 – URL: https://arxiv.org/abs/1804.02767 (дата звернення: 09.06.2025).
12. Rizzi A., Gatta C., Piacentini B., Fierro M. Human-visual-system-inspired tone mapping algorithm for HDR images. Proceedings of SPIE 5292. The International Society for Optical Engineering, 2004 – С. 57–68 – DOI: 10.1117/12.525777 – URL: https://www.researchgate.net/publication/260969452_Human-visual-system-inspired_tone_mapping_algorithm_for_HDR_images (дата звернення: 21.04.2025).
13. Shao L., Wang Y., Li J. Anchor-Free Object Detection with Scale-Aware Networks for Autonomous Driving. Electronics, 2022. Vol. 11(20), 3303 – URL: https://www.mdpi.com/2079-9292/11/20/3303 (дата звернення: 7.05.2025).
14. Tian Z., Shen C., Chen H., He T. FCOS: Fully Convolutional One-Stage Object Detection. arXiv preprint arXiv:1904.01355, 2019 – URL: https://arxiv.org/abs/1904.01355 (дата звернення: 9.05.2025).
15. Timilsina A. YOLOv8 Architecture Explained! Medium, 2024 – URL: https://abintimilsina.medium.com/yolov8-architecture-explained-a5e90a560ce5 (дата звернення: 10.05.2025).

16. Tone Mapping | δelta – URL: https://64.github.io/tonemapping/ (дата звернення: 21.04.2025).
17. Ultralytics YOLO Documentation. Ultralytics, 2024 – URL: https://docs.ultralytics.com/ (дата звернення: 10.05.2025).
18. Vinker Y., Mlechkov P., Holtzman G. та ін. Unpaired Learning for High Dynamic Range Image Tone Mapping. Proc. ICCV 2021 – С. 12444–12453 – URL: https://openaccess.thecvf.com/content/ICCV2021/papers/Vinker_Unpaired_Learning_for_High_Dynamic_Range_Image_Tone_Mapping_ICCV_2021_paper.pdf (дата звернення: 6.05.2025).
19. What is NETD in a Thermal Camera? MoviTHERM Blog, 2023 – URL: https://movitherm.com/blog/what-is-netd-in-a-thermal-camera/ (дата звернення: 20.04.2025).
20. Zhang S., Chi C., Yao Y. та ін. Bridging the Gap Between Anchor-Based and Anchor-Free Detection via Adaptive Training Sample Selection. Proc. CVPR 2020 – С. 9759–9768 – URL: https://openaccess.thecvf.com/content_CVPR_2020/papers/Zhang_Bridging_the_Gap_Between_Anchor-Based_and_Anchor-Free_Detection_via_Adaptive_CVPR_2020_paper.pdf (дата звернення: 7.05.2025).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import os
import glob
import cv2
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import itertools
import cv2
import json
import numpy as np
import yaml
import shutil
import os
import torch
from torch.utils.data import Dataset, DataLoader
from tqdm.auto import tqdm
from thermal_map import ThermalMultiChannelEmbedding
import os
from pathlib import Path
import tifffile as tiff
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim
from tqdm.auto import tqdm
import importlib
import types
import sys
import scipy.signal as signal
from torch.utils.data import DataLoader, Subset
from torchvision.datasets import CocoDetection
from torchvision.transforms import ToTensor
from torchvision.models.detection import fcos_resnet50_fpn
from torchvision.models.detection.retinanet import RetinaNetClassificationHead
from tqdm.auto import tqdm
from pathlib import Path
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval
from torchvision.transforms.functional import to_tensor
from pathlib import Path
from tqdm.auto import tqdm
from pycocotools.coco import COCO
from ultralytics import YOLO


folder_path = Path("thermal_16_bit")

tiff_files = sorted([f for f in folder_path.iterdir() if f.suffix.lower() == ".tiff"])
```

```
first_path = tiff_files[320]
img16 = tiff.imread(first_path)

print("Файл:", first_path.name)
print("Мінімум/максимум у масиві:", img16.min(), img16.max())

plt.figure(figsize=(6, 6))
plt.imshow(img16, cmap="gray", vmin=img16.min(), vmax=img16.max()+200)
plt.title(f"{first_path.name} (vmin={img16.min()}, vmax={img16.max()})")
plt.axis("off")
plt.show()

unique_vals = np.unique(img16)
print("Кількість унікальних значень:", unique_vals.size)

zero_count = np.sum(img16 == 0)
print("Кількість пікселів із інтенсивністю 0:", zero_count, "(", zero_count/img16.size*100,
"% )")

counts = img16.ravel()
unique_vals = np.unique(counts)
bit_range = int(1 + np.ceil(np.log2(unique_vals.max() + 1)))
print("Приблизна «дійсна» розрядність (біт), зайнята значеннями:", bit_range)

unique_vals = np.unique(img16)
print("Кількість унікальних значень:", unique_vals.size)

zero_count = np.sum(img16 == 0)
print("Кількість пікселів із інтенсивністю 0:", zero_count, "(", zero_count/img16.size*100,
"% )")

counts = img16.ravel()
unique_vals = np.unique(counts)
bit_range = int(1 + np.ceil(np.log2(unique_vals.max() + 1)))
print("Приблизна «дійсна» розрядність (біт), зайнята значеннями:", bit_range)


class Thermal16BitDataset(Dataset):
    def __init__(self,
                 root_16bit: str = "thermal_16_bit",
                 root_8bit: str = "thermal_8_bit",
                 pattern: str = "FLIR_{:05d}.tiff",
                 first_n: int = 400,
                 device: torch.device = torch.device("cpu")):
        super().__init__()
        self.root_16bit = root_16bit
        self.root_8bit = root_8bit
        self.pattern = pattern
        self.first_n = first_n
        self.device = device
        self.samples = [self.pattern.format(i) for i in range(1, self.first_n + 1)]
```

```
        self.remapper = ThermalMultiChannelEmbedding(
            num_output_channels=10,
            img_type='tiff',
            sync=False,
            device=self.device
        )

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        name_16 = self.samples[idx]
        path_16 = os.path.join(self.root_16bit, name_16)
        img16 = cv2.imread(path_16, cv2.IMREAD_UNCHANGED)
        if img16 is None:
            raise FileNotFoundError(f"Не знайдено 16-бітний TIFF: {path_16}")
        if img16.ndim != 2:
            raise ValueError(f"{name_16} не є 2D одночастотним TIFF")
        raw = img16.astype(np.float32)
        raw_tensor = torch.from_numpy(raw)[None, ...].to(self.device)
        remapped = self.remapper(raw_tensor)
        remapped = remapped.squeeze(0)
        base = os.path.splitext(name_16)[0]
        jpeg_name = base + ".jpeg"
        path_8 = os.path.join(self.root_8bit, jpeg_name)
        img8 = cv2.imread(path_8, cv2.IMREAD_GRAYSCALE)
        if img8 is None:
            raise FileNotFoundError(f"Не знайдено 8-бітний JPEG: {path_8}")
        img8 = img8.astype(np.float32) / 255.0
        tgt_3c = np.stack([img8, img8, img8], axis=0)
        tgt_tensor = torch.from_numpy(tgt_3c).to(self.device)
        return {
            'input': remapped,
            'target': tgt_tensor,
            'raw_int16': raw_tensor,
            'hdr_name': name_16
        }


class TCNN(nn.Module):
    def __init__(self,
                 in_channels: int = 10,
                 out_channels: int = 3,
                 mlp_hidden: int = 64):
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.embed_dim = in_channels * out_channels
        self.conv1 = nn.Conv2d(
            in_channels,
            self.embed_dim,
            kernel_size=3,
```

```
            stride=2,
            padding=1,
            groups=in_channels
        )
        self.norm1 = nn.GroupNorm(num_groups=in_channels,
                                  num_channels=self.embed_dim)
        self.conv2 = nn.Conv2d(
            self.embed_dim,
            self.embed_dim,
            kernel_size=3,
            stride=2,
            padding=1,
            groups=in_channels
        )
        self.norm2 = nn.GroupNorm(num_groups=in_channels,
                                  num_channels=self.embed_dim)
        self.relu = nn.ReLU(inplace=True)
        self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.mlp = nn.Sequential(
            nn.Flatten(),
            nn.Linear(self.embed_dim, mlp_hidden),
            nn.ReLU(inplace=True),
            nn.Linear(mlp_hidden, self.embed_dim)
        )
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if x.dim() == 4 and x.shape[1] != self.in_channels:
            x = x.permute(0, 3, 1, 2).contiguous()
        N, C, H, W = x.shape
        assert C == self.in_channels, f"Expected {self.in_channels} channels, got {C}"
        feat = self.relu(self.norm1(self.conv1(x)))
        feat = self.relu(self.norm2(self.conv2(feat)))
        pooled = self.global_pool(feat)
        pooled = pooled.view(N, self.embed_dim)
        weights = self.mlp(pooled)
        weights = weights.view(N, C, self.out_channels)
        weights = self.softmax(weights)
        output = torch.einsum('nci,nchw->nihw', weights, x)
        return output


class LocalContrastNormalization(nn.Module):
    def __init__(self, in_channels: int = 3, kernel_size: int = 3):
        super().__init__()
        padding = kernel_size // 2
        self.lcn_conv = nn.Conv2d(
            in_channels,
            in_channels,
            kernel_size=kernel_size,
            stride=1,
            padding=padding,
```

```
            bias=False,
            groups=in_channels
        )
        kernel = torch.ones(in_channels, 1, kernel_size, kernel_size) / (kernel_size * kernel_size)
        self.register_buffer("lcn_kernel", kernel)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        self.lcn_conv.weight.data = self.lcn_kernel.clone()
        local_mean = self.lcn_conv(x)
        x_norm = x – local_mean
        return x_norm


class TCNNMultiTask(nn.Module):
    def __init__(self,
                 in_channels: int = 10,
                 out_channels_tone: int = 3,
                 mlp_hidden: int = 64):
        super().__init__()
        self.base_net = TCNN(in_channels=in_channels,
                             out_channels=out_channels_tone,
                             mlp_hidden=mlp_hidden)
        self.embed_dim = in_channels * out_channels_tone
        self.lcn = LocalContrastNormalization(in_channels=out_channels_tone, kernel_size=3)

    def forward(self, x: torch.Tensor) -> (torch.Tensor, torch.Tensor):
        out_linear = self.base_net(x)
        pred_rgb = torch.sigmoid(out_linear)
        pred_lcn = self.lcn(pred_rgb)
        return pred_rgb, pred_lcn


if __name__ == "__main__":
    batch_size = 4
    extra_epochs = 30
    batches_limit = 99
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    train_set = Thermal16BitDataset(
        root_16bit="thermal_16_bit",
        root_8bit="thermal_8_bit",
        pattern="FLIR_{:05d}.tiff",
        first_n=4000,
        device=device
    )
    train_loader = DataLoader(
        train_set,
        batch_size=batch_size,
        shuffle=True,
        num_workers=4,
        pin_memory=True
    )
    model = TCNNMultiTask(in_channels=10, out_channels_tone=3, mlp_hidden=64).to(device)
```

```
criterion_tone = nn.MSELoss().to(device)
criterion_lcn = nn.MSELoss().to(device)
log_var_tone = torch.zeros(1, requires_grad=True, device=device)
log_var_lcn = torch.zeros(1, requires_grad=True, device=device)
optimizer = optim.Adam(
    list(model.parameters()) + [log_var_tone, log_var_lcn],
    lr=1e-4,
    weight_decay=1e-6
)
os.makedirs("Checkpoint", exist_ok=True)
checkpoint_file = "Checkpoint/Checkpoint_path.temp"
if os.path.isfile(checkpoint_file):
    with open(checkpoint_file, "r", encoding="utf-8") as f:
        last_ckpt = f.readline().strip()
    ckpt_path = os.path.join("Checkpoint", last_ckpt)
    if os.path.isfile(ckpt_path):
        chk = torch.load(ckpt_path, map_location=device)
        model.load_state_dict(chk["model_state_dict"])
        log_var_tone.data.copy_(chk["log_var_tone"])
        log_var_lcn.data.copy_(chk["log_var_lcn"])
        try:
            last_epoch = int(last_ckpt.split("_")[-1].split(".")[0])
        except:
            last_epoch = 0
    else:
        last_epoch = 0
else:
    last_epoch = 0
start_epoch = last_epoch + 1
end_epoch = last_epoch + extra_epochs
for epoch in range(start_epoch, end_epoch + 1):
    model.train()
    running_loss = 0.0
    pbar = tqdm(range(1, batches_limit + 1),
                total=batches_limit,
                desc=f"Epoch {epoch}/{end_epoch}",
                leave=False)
    limited_loader = itertools.islice(enumerate(train_loader, 1), batches_limit)
    for batch_idx, raw_batch in zip(pbar, limited_loader):
        if isinstance(raw_batch, dict):
            batch = raw_batch
        else:
            item = raw_batch[1]
            if isinstance(item, dict):
                batch = item
            else:
                inp_tensor, tgt_tensor, raw_tensor, hdr_nm = item
                batch = {
                    'input': inp_tensor,
                    'target': tgt_tensor,
                    'raw_int16': raw_tensor,
                    'hdr_name': hdr_nm
```

```
                }
            inputs = batch['input'].to(device)
            targets_rgb = batch['target'].to(device)
            pred_rgb, pred_lcn = model(inputs)
            loss_tone = criterion_tone(pred_rgb, targets_rgb)
            loss_lcn = criterion_lcn(pred_lcn, targets_rgb)
            precision_tone = torch.exp(-log_var_tone)
            precision_lcn = torch.exp(-log_var_lcn)
            weighted_tone = precision_tone * loss_tone + log_var_tone
            weighted_lcn = precision_lcn * loss_lcn + log_var_lcn
            loss = weighted_tone + weighted_lcn
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            mean_loss = running_loss / batch_idx
            pbar.set_postfix({
                'loss': f"{loss.item():.4f}",
                'tone': f"{loss_tone.item():.4f}",
                'lcn': f"{loss_lcn.item():.4f}",
                'avg': f"{mean_loss:.4f}"
            })
        ckpt_name = f"tcnn_epoch_{epoch:04d}.pth"
        ckpt_path = os.path.join("Checkpoint", ckpt_name)
        torch.save({
            'model_state_dict': model.state_dict(),
            'log_var_tone': log_var_tone.detach(),
            'log_var_lcn': log_var_lcn.detach()
        }, ckpt_path)
        with open("Checkpoint/Checkpoint_path.temp", "w", encoding="utf-8") as temp_f:
            temp_f.write(ckpt_name + "\n")
        print(f"Epoch {epoch}/{end_epoch} завершено — avg_loss {mean_loss:.4f}")
    print("Тренування завершено, чекпоінт оновлено.")



ROOT_PNG = Path("thermal_sdr_png")
ANN_FILE = Path("thermal_annotations.json")
LIMIT = 240
TRAIN_FRAC = 0.8
BATCH_SIZE = 4
NUM_EPOCHS = 50
LR = 1e-3

augs = A.Compose(
    [
        A.LongestMaxSize(640),
        A.PadIfNeeded(640, 640, border_mode=cv2.BORDER_CONSTANT),
        A.HorizontalFlip(p=0.5)
    ],
    bbox_params=A.BboxParams(format="coco", label_fields=["category_ids"]),
)
```

```
class CocoAug(CocoDetection):
    def __init__(self, root, ann_file, tfm):
        super().__init__(root, ann_file)
        self.tfm = tfm
        self.root = Path(root)

    def _find_path(self, fn):
        p = self.root / fn
        if p.is_file():
            return p
        p = self.root / Path(fn).with_suffix(".png").name
        return p if p.is_file() else None

    def _load_img(self, fn):
        p = self._find_path(fn)
        if p is None:
            raise FileNotFoundError(fn)
        img = cv2.imread(str(p), cv2.IMREAD_GRAYSCALE)
        return cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    def __getitem__(self, idx):
        for _ in range(10):
            img_id = self.ids[idx]
            anns = self.coco.loadAnns(self.coco.getAnnIds(imgIds=img_id))
            fn = self.coco.loadImgs(img_id)[0]["file_name"]

            img = self._load_img(fn)
            bboxes = [a["bbox"] for a in anns]
            labels = [a["category_id"] for a in anns]

            aug = self.tfm(image=img, bboxes=bboxes, category_ids=labels)
            img, bboxes, labels = aug["image"], aug["bboxes"], aug["category_ids"]

            boxes = np.array([[x, y, x + w, y + h] for x, y, w, h in bboxes], dtype=np.float32)
            w, h = boxes[:, 2] – boxes[:, 0], boxes[:, 3] – boxes[:, 1]
            keep = (w > 1) & (h > 1)
            if keep.any():
                boxes = torch.as_tensor(boxes[keep], dtype=torch.float32)
                labels = torch.as_tensor(np.array(labels)[keep], dtype=torch.int64)
                return ToTensor()(img), {"boxes": boxes, "labels": labels}
            idx = (idx + 1) % len(self.ids)
        raise RuntimeError("no valid sample in 10 tries")


def collate(b):
    return tuple(zip(*b))


full_ds = CocoAug(ROOT_PNG, ANN_FILE, augs)
```

```
indices = []
for idx, img_id in enumerate(full_ds.ids):
    if not full_ds.coco.getAnnIds(imgIds=img_id):
        continue
    fn = full_ds.coco.loadImgs(img_id)[0]["file_name"]
    if full_ds._find_path(fn) is None:
        continue
    indices.append(idx)
    if len(indices) == LIMIT:
        break

if len(indices) < LIMIT:
    print(f" Evaluating {len(indices)} images ")

random.seed(42)
random.shuffle(indices)
split = int(TRAIN_FRAC * len(indices))
train_ds = Subset(full_ds, indices[:split])
val_ds = Subset(full_ds, indices[split:])
print(f"Train {len(train_ds)} | Val {len(val_ds)}")

train_ld = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True, num_workers=0,
collate_fn=collate)
val_ld = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False, num_workers=0,
collate_fn=collate)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = fcos_resnet50_fpn(weights=None)
num_classes = len(full_ds.coco.cats) + 1
anchors = model.head.classification_head.num_anchors
model.head.classification_head = RetinaNetClassificationHead(256, anchors, num_classes)
model.to(device)

optim = torch.optim.SGD(model.parameters(), lr=LR, momentum=0.9, weight_decay=5e-4)
torch.autograd.set_detect_anomaly(True)

for ep in range(1, NUM_EPOCHS + 1):
    model.train()
    tr_losses = []
    for imgs, tgts in tqdm(train_ld, desc=f"Epoch {ep}/{NUM_EPOCHS}", ncols=80):
        imgs = [i.to(device) for i in imgs]
        tgts = [{k: v.to(device) for k, v in t.items()} for t in tgts]
        loss = sum(model(imgs, tgts).values())
        if torch.isnan(loss):
            continue
        optim.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 10)
        optim.step()
        tr_losses.append(loss.item())
    mean_tr = np.mean(tr_losses)
```

```
    model.train()
    val_losses = []
    with torch.no_grad():
        for imgs, tgts in val_ld:
            imgs = [i.to(device) for i in imgs]
            tgts = [{k: v.to(device) for k, v in t.items()} for t in tgts]
            val_losses.append(sum(model(imgs, tgts).values()).item())
    mean_val = np.mean(val_losses)
    print(f"Epoch {ep}/{NUM_EPOCHS} finished — avg_loss {mean_tr:.4f} | val_loss
{mean_val:.4f}")

torch.save(model.state_dict(), "fcos_thermal_sub.pth")
print("Weights saved in fcos_thermal_sub.pth")



assert 'val_ds' in globals()

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

cat_ids_sorted = sorted(val_ds.dataset.coco.cats.keys())
label2cat = {i+1: cid for i, cid in enumerate(cat_ids_sorted)}

model.eval()

val_loader = DataLoader(val_ds, 1, False, num_workers=0, collate_fn=collate)
results = []

for ds_idx, (imgs, _) in enumerate(tqdm(val_loader, desc="Infer", ncols=80)):
    coco_img_id = val_ds.dataset.ids[val_ds.indices[ds_idx]]
    img_t = imgs[0].to(device)

    with torch.no_grad():
        pred = model([img_t])[0]

    boxes  = pred["boxes"].cpu().numpy()
    scores = pred["scores"].cpu().numpy()
    labels = pred["labels"].cpu().numpy()
    if not len(boxes):
        continue
    boxes[:,2:] -= boxes[:,:2]

    for bb, sc, lb in zip(boxes, scores, labels):
        cid = label2cat.get(int(lb), None)
        if cid is None:
            continue
        results.append({
            "image_id":   int(coco_img_id),
            "category_id": int(cid),
            "bbox":  [float(x) for x in bb],
            "score": float(sc),
        })
```

```
Path("fcos_val_results.json").write_text(json.dumps(results))

coco_gt = val_ds.dataset.coco
coco_dt = coco_gt.loadRes("fcos_val_results.json")
ev = COCOeval(coco_gt, coco_dt, iouType='bbox')
ev.evaluate()
ev.accumulate()
ev.summarize()

print(f"\nCOCO mAP@[.50:.95] = {ev.stats[0]:.4f}   mAP@.5 = {ev.stats[1]:.4f}")

ious = []
for img_id in coco_gt.getImgIds():
    gt_ids = coco_gt.getAnnIds(imgIds=img_id)
    dt_ids = coco_dt.getAnnIds(imgIds=img_id)
    if not gt_ids or not dt_ids:
        continue
    gt = np.array([coco_gt.anns[i]["bbox"] for i in gt_ids])
    dt = np.array([coco_dt.anns[i]["bbox"] for i in dt_ids])
    gt[:,2:]+=gt[:,:2]
    dt[:,2:]+=dt[:,:2]
    for d in dt:
        inter = np.maximum(0,np.minimum(gt[:,2],d[2])-np.maximum(gt[:,0],d[0])) * \
                np.maximum(0,np.minimum(gt[:,3],d[3])-np.maximum(gt[:,1],d[1]))
        union = ((d[2]-d[0])*(d[3]-d[1]) +
                 (gt[:,2]-gt[:,0])*(gt[:,3]-gt[:,1]) – inter)
        if union.size:
            ious.append((inter/union).max())
mean_iou = np.mean(ious)
print(f"Mean IoU (best match) = {mean_iou:.4f}")

CONF_THRES = 0.5
show_ids   = val_ds.dataset.ids[:6]

fig, axes = plt.subplots(2,3, figsize=(15,8))
for ax, cid in zip(axes.ravel(), show_ids):
    info = coco_gt.loadImgs(cid)[0]
    pth  = ROOT_PNG / Path(info["file_name"]).with_suffix(".png").name
    g    = cv2.imread(str(pth), cv2.IMREAD_GRAYSCALE)
    vis  = cv2.cvtColor(g, cv2.COLOR_GRAY2BGR)

    gt_b = [coco_gt.anns[i]["bbox"] for i in coco_gt.getAnnIds(imgIds=cid)]
    gt_b = np.array([[x,y,x+w,y+h] for x,y,w,h in gt_b])
    for x1,y1,x2,y2 in gt_b:
        cv2.rectangle(vis,(int(x1),int(y1)),(int(x2),int(y2)),(0,0,255),2)

    inp = to_tensor(g).unsqueeze(0).to(device)
    with torch.no_grad():
        p = model(inp)[0]
    keep = p["scores"]>=CONF_THRES
    pb   = p["boxes"][keep].cpu().numpy()
```

```
    scr  = p["scores"][keep].cpu().numpy()
    for (x1,y1,x2,y2), sc in zip(pb, scr):
        cv2.rectangle(vis,(int(x1),int(y1)),(int(x2),int(y2)),(0,255,0),2)
        cv2.putText(vis,f"{sc:.2f}",(int(x1),int(y1)-2),
                    cv2.FONT_HERSHEY_SIMPLEX,.4,(0,255,0),1)

    ax.imshow(cv2.cvtColor(vis, cv2.COLOR_BGR2RGB))
    ax.set_title(Path(pth).name)
    ax.axis('off')

plt.tight_layout()
plt.show()



ROOT_PNG   = Path("thermal_sdr_png")
ANN_FILE   = Path("thermal_annotations.json")
OUT_DIR    = Path("yolo_subset")
LIMIT      = 300
TRAIN_FRAC = 0.8
BATCH      = 4
EPOCHS     = 65
IMGSZ      = 640

if OUT_DIR.exists():
    shutil.rmtree(OUT_DIR)
(OUT_DIR / "images/train").mkdir(parents=True)
(OUT_DIR / "images/val").mkdir(parents=True)
(OUT_DIR / "labels/train").mkdir(parents=True)
(OUT_DIR / "labels/val").mkdir(parents=True)

coco = COCO(str(ANN_FILE))
sel = []
for img_id in coco.getImgIds():
    if not coco.getAnnIds(imgIds=img_id):
        continue
    fn = coco.loadImgs(img_id)[0]["file_name"]
    png = ROOT_PNG / Path(fn).with_suffix(".png").name
    if png.is_file():
        sel.append(img_id)
    if len(sel) == LIMIT:
        break
if len(sel) < LIMIT:
    print(f"Лише {len(sel)} кадрів – беремо всі.")

random.seed(42)
random.shuffle(sel)
split = int(TRAIN_FRAC * len(sel))
train_ids = sel[:split]
val_ids   = sel[split:]
print(f"Train {len(train_ids)} | Val {len(val_ids)}")
```

```
names = {int(k):v["name"] for k,v in coco.cats.items()}
cat_id2idx = {cid:i for i,cid in enumerate(sorted(names))}
def coco2yolo(b,w,h):
    x,y,bw,bh = b
    return (x+bw/2)/w, (y+bh/2)/h, bw/w, bh/h

def convert(ids, split):
    for img_id in tqdm(ids, desc=split):
        info = coco.loadImgs(img_id)[0]
        fn_png = Path(info["file_name"]).with_suffix(".png").name
        src = ROOT_PNG / fn_png
        dst_img = OUT_DIR / f"images/{split}/{fn_png}"
        os.symlink(src.resolve(), dst_img)
        h, w = info["height"], info["width"]
        lines = []
        for a in coco.loadAnns(coco.getAnnIds(imgIds=img_id)):
            cls = cat_id2idx[a["category_id"]]
            lines.append("{} {:.6f} {:.6f} {:.6f} {:.6f}".format(cls, *coco2yolo(a["bbox"],w,h)))
        (OUT_DIR / f"labels/{split}/{fn_png[:-4]}.txt").write_text("\n".join(lines))
convert(train_ids,"train")
convert(val_ids,"val")

data_yaml = {
    "path": str(OUT_DIR),
    "train": "images/train",
    "val":   "images/val",
    "nc": len(names),
    "names":[names[k] for k in sorted(names)]
}
yaml_path = OUT_DIR / "data.yaml"
yaml_path.write_text(yaml.dump(data_yaml))

device = 0 if torch.cuda.is_available() else "cpu"
model  = YOLO("yolov8n.pt")
results = model.train(
    data=str(yaml_path),
    epochs=EPOCHS,
    imgsz=IMGSZ,
    batch=BATCH,
    workers=0,
    device=device,
    name="thermal_sub"
)

best = Path(results.save_dir) / "weights/best.pt"
dest = Path("yolo_thermal_sub.pt")
shutil.copy2(best, dest)
print(f" Weights saved → {dest}")


ROOT_PNG = Path("thermal_sdr_png")
ANN_FILE = Path("thermal_annotations.json")
```

```
WEIGHTS  = Path("yolo_thermal_sub.pt")
LIMIT    = 400
CONF_TH  = 0.5

coco = COCO(str(ANN_FILE))
img_ids = []
img_paths = []
for img_id in coco.getImgIds():
    fn = coco.loadImgs(img_id)[0]["file_name"]
    png = ROOT_PNG / Path(fn).with_suffix(".png").name
    if png.is_file():
        img_ids.append(img_id)
        img_paths.append(png)
    if len(img_ids) == LIMIT:
        break
print(f"Evaluating {len(img_ids)} images")

device = 0 if torch.cuda.is_available() else "cpu"
model  = YOLO(str(WEIGHTS))
names  = model.names
idx2cat= {idx: int(cid) for idx, cid in enumerate(sorted(coco.cats))}

results = []
for img_id, path in tqdm(list(zip(img_ids, img_paths)), desc="Infer", ncols=80):
    out = model.predict(source=str(path), conf=0.001, iou=0.7, device=device, verbose=False)[0]

    if not len(out):
        continue
    xyxy   = out.boxes.xyxy.cpu().numpy()
    scores = out.boxes.conf.cpu().numpy()
    labels = out.boxes.cls.cpu().numpy().astype(int)

    xywh = xyxy.copy()
    xywh[:,2:] -= xywh[:,:2]
    for bb, sc, lb in zip(xywh, scores, labels):
        results.append({
            "image_id":   int(img_id),
            "category_id": idx2cat[int(lb)],
            "bbox":  [float(x) for x in bb],
            "score": float(sc),
        })

Path("yolo_val_results.json").write_text(json.dumps(results))

coco_dt = coco.loadRes("yolo_val_results.json")
ev = COCOeval(coco, coco_dt, iouType='bbox')
ev.evaluate()
ev.accumulate()
ev.summarize()

print(f"\nCOCO mAP@[.50:.95] = {ev.stats[0]:.4f}")
print(f"COCO mAP@0.50      = {ev.stats[1]:.4f}")
```

```
print(f"Average Recall     = {ev.stats[8]:.4f}")

ious = []
for img_id in coco.getImgIds():
    gt_ids = coco.getAnnIds(imgIds=img_id)
    dt_ids = coco_dt.getAnnIds(imgIds=img_id)
    if not gt_ids or not dt_ids:
        continue
    gt = np.array([coco.anns[i]["bbox"] for i in gt_ids])
    dt = np.array([coco_dt.anns[i]["bbox"] for i in dt_ids])
    gt[:,2:]+=gt[:,:2]
    dt[:,2:]+=dt[:,:2]
    for d in dt:
        inter = (np.maximum(0,np.minimum(gt[:,2],d[2])-np.maximum(gt[:,0],d[0])) *
                 np.maximum(0,np.minimum(gt[:,3],d[3])-np.maximum(gt[:,1],d[1])))
        union = ((d[2]-d[0])*(d[3]-d[1]) +
                 (gt[:,2]-gt[:,0])*(gt[:,3]-gt[:,1]) – inter)
        if union.size:
            ious.append((inter/union).max())
mean_iou = np.mean(ious)
print(f"Mean IoU (best match) = {mean_iou:.4f}")

show_ids = img_ids[:12]
fig, axes = plt.subplots(2,3, figsize=(15,8))

for ax, cid in zip(axes.ravel(), show_ids):
    info = coco.loadImgs(cid)[0]
    path = ROOT_PNG / Path(info["file_name"]).with_suffix(".png").name
    g    = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)
    vis  = cv2.cvtColor(g, cv2.COLOR_GRAY2BGR)

    for x,y,w,h in [coco.anns[i]["bbox"] for i in coco.getAnnIds(imgIds=cid)]:
        cv2.rectangle(vis,(int(x),int(y)),(int(x+w),int(y+h)),(0,0,255),2)

    pred = model.predict(source=str(path), conf=CONF_TH, iou=0.7, device=device,
verbose=False)[0]

    if len(pred.boxes):
        for (x1,y1,x2,y2), sc in zip(pred.boxes.xyxy.cpu().numpy(),
pred.boxes.conf.cpu().numpy()):
            cv2.rectangle(vis,(int(x1),int(y1)),(int(x2),int(y2)),(0,255,0),2)
            cv2.putText(vis,f"{sc:.2f}",(int(x1),int(y1)-2), cv2.FONT_HERSHEY_SIMPLEX, .4,
(0,255,0),1)
    else:
        cv2.putText(vis,"no preds",(5,15), cv2.FONT_HERSHEY_SIMPLEX, .5, (0,255,255),1)

    ax.imshow(cv2.cvtColor(vis, cv2.COLOR_BGR2RGB))
    ax.set_title(Path(path).name)
    ax.axis('off')

plt.tight_layout()
plt.show()
```