Table of Contents

# “Метод автоматичного формування ієрархічної структури класів для розпізнавання об’єктів”

**author: “Науково-технічний звіт”** **date: “23.07.2025”** **lang: uk-UA**

У технічному звіті запропоновано формальний, інтуїтивно зрозумілий метод, який автоматично формує дерево рівнів/класів, спираючись на (і) відстані між векторними поданнями класів, (іі) величину невідповідності між ними та (ііі) прості порогові правила. Алгоритм узгоджує емпіричні спостереження авторів щодо трирівневої схеми «H/V → T/B/O → G/M/Z» й узагальнює їх до автоматично масштабованої процедури.

## 1. Запропонований метод

Метод складається з восьми послідовних кроків, що охоплюють процеси від вилучення ознак до експорту готової ієрархічної структури. На вхід подаються дані, зазначені в таблиці 1.

**Таблиця 1.** Вхідні дані для алгоритму.

| Позначення | Розмірність | Пояснення |
| --- | --- | --- |
|  |  | Навчальний набір кадрів БПЛА з їхніми початковими мітками. |
|  |  | **Базові класи** (тобто найдрібніші, що вже не діляться). |
|  | скаляр | Кількість базових класів. |
|  | функція | Back-bone, який перетворює ROI у вектор ознак розмірності . |
|  | скаляр | Коефіцієнт, що задає частку просторової відстані відносно емпіричної невідповідності у комбінованій мірі . |
|  | скаляр | Мінімальна відстань між кластерами, за якої їх ще розділяємо. |
|  | скаляр | Максимально допустима частка помилкових сплутувань усередині кластера. |
|  | вектор | Мінімальна швидкість (кадрів/с) та максимальна латентність (мс). |

### 1.1. Крок 1: Вилучення ознак для усіх ROI

Перший етап є підготовчим. Його мета — перетворити кожну область інтересу (ROI) на зображенні у числовий вектор (ембединг), з яким можна виконувати математичні операції.

1. Для кожного зображення виконуємо первинне детектування (будь-який SOTA-детектор) та отримуємо множину ROI .
2. Для кожного ROI обчислюємо вектор ознак:

**Примітка 1.** Для забезпечення відтворюваності використовується однаковий базовий детектор ROI на всіх рівнях — YOLOv11-m з конфігурацією confidence=0.25, IoU=0.5. Усі подальші моделі працюють з уже обрізаними ROI і не виконують повторного пошуку об’єктів. **Примітка 2.** Дані діляться на train/val/test у пропорції 70 / 15 / 15 %, що жорстко фіксується через random seed = 2025.

### 1.2. Крок 2: Побудова «профілю» класу

На цьому етапі для кожного базового класу ми обчислюємо дві ключові характеристики: його усереднене представлення у просторі ознак (центроїд) та те, як часто його плутають з іншими класами (матриця невідповідності). Ці два показники стануть основою для вимірювання «близькості» між класами.

1. Для кожного атомарного класу збираємо множину ознак .
2. Обчислюємо **центроїд**:
3. Одноразово навчаємо одноетапний класифікатор без ієрархії (наприклад, простий MLP) на всіх та формуємо **матрицю невідповідності**:

* де — кількість неправильних присвоєнь .

Тут під одноетапним класифікатором без ієрархії розуміємо модель, що одразу прогнозує один з усіх базових класів без ієрархії.

### 1.3. Крок 3: Визначення міри близькості між класами

Тут ми поєднуємо дві метрики з попереднього кроку в єдиний показник . Ця міра близькості враховує як геометричну відстань між центроїдами класів, так і емпіричні дані про те, наскільки часто класифікатор їх плутає. Параметр дозволяє регулювати важливість кожного з цих компонентів. Комбінація відбувається за формулою (3):

*Чим більшим є — тим легше класи розділити.*

### 1.4. Крок 4: Ієрархічне агломеративне групування

Це ядро алгоритму, де відбувається безпосередня побудова ієрархії. Використовуючи обчислену міру близькості , ми застосовуємо метод кластеризації, який послідовно об’єднує найближчі класи. Потім, рухаючись по отриманій дендрограмі, ми приймаємо рішення про поділ або об’єднання груп на основі заданих порогів, перевіряючи умову (4).

1. Будуємо повнозважений граф з вагами та застосовуємо agglomerative clustering (average linkage), отримуючи дендрограму .
2. Переходимо зверху донизу: для кожної внутрішньої вершини перевіряємо умову:

* Якщо **умова виконується** → **не розділяємо** (класи лишаються у цьому ж листі).
* Якщо **умова не виконується** → **ділимо** вершину на її дві дочірні кластери й повторюємо перевірку.

У результаті маємо рівнів з дедалі дрібнішими класами; алгоритм природно повторює логіку H/V → T/B/O → G/M/Z з рукопису, але без ручного втручання.

### 1.5. Крок 5: Призначення моделей на кожен рівень ієрархії

Після того, як структура ієрархії визначена, необхідно для кожного її рівня підібрати оптимальні моделі детектора та класифікатора. Вибір відбувається автоматично на основі характеристик рівня (кількість класів, розмір об’єктів) та апаратних обмежень.

**Мета:** для кожного рівня вибрати пару **детектор**  і **класифікатор**  так, щоб:

1. виконувалися ресурсні обмеження ;
2. модель залишалася придатною до малого/великого числа класів та розміру об’єктів.

#### 1.5.1. Вхідні параметри (усі легко вимірювані)

**Таблиця 2.** Вхідні параметри для вибору моделей на рівні .

| Змінна | Розмірність / Тип | Як отримати | Що означає |
| --- | --- | --- | --- |
|  | скаляр (ціле) | Обчислити кількість унікальних підкласів, що залишилися на рівні . | Скільки категорій треба розрізняти на поточному рівні. |
|  | скаляр (пікселі) | Взяти медіану значень для всіх ROI рівня . | Середній лінійний розмір об’єктів, які обробляє рівень. |
|  | скаляр (кадри/с) | Задано технічним завданням. | Мінімально допустима швидкість системи в робочому режимі. |
|  | скаляр (мс) | Задано технічним завданням. | Гранична сумарна латентність повного каскаду. |
|  | скаляр (TFLOPS) | Паспортна потужність доступного GPU (або фактична, виміряна утилітою nvidia-smi). | Обчислювальний ресурс, на який спирається вибір «важкої» чи «легкої» моделі. |

##### 1.5.2. Дерево рішень (детермінований алгоритм)

Логіка вибору моделей базується на наборі if-elif-else правил, що враховують параметри з таблиці 2.

# ---- детермінований вибір моделей на рівні k ----  
if |L\_k| <= 2 and bar\_s\_k >= 48:  
 D\_k = "YOLOv11-l"; F\_k = "CSPDarknet-53"  
 C\_k = "MLP(256)"  
elif 3 <= |L\_k| <= 6 and bar\_s\_k >= 32:  
 D\_k = "YOLOv11-m"; F\_k = "CSPDarknet-53+SPP-FPN"  
 C\_k = "FT-Transformer(d=512,h=8,blocks=2)"  
else: # |L\_k| > 6 or bar\_s\_k < 32  
 D\_k = "RT-DETR-Tiny"; F\_k = "CSPDarknet-53+FPN+CARAFE"  
 C\_k = "FT-Transformer(d=768,h=12,blocks=4)"  
  
# ---- ресурсне зниження складності ----  
if C\_GPU < 5.0: # TFLOPS  
 D\_k = "YOLO-Nano"; F\_k = "Lite-FPN"  
 C\_k = "MobileNet-MLP(128)"

**Примітка 1.** Значення 5.0 TFLOPS відповідає орієнтовній межі між «десктопними» та «вбудованими» GPU‑платформами (Jetson AGX Orin ≈ 4 TFLOPS FP16).

#### 1.5.3. Правила корекції

Для забезпечення надійності системи застосовуються додаткові правила корекції.

1. **Перевірка латентності:** якщо сумарна латентність рівня , знижуємо варіант на один рівень у дереві.
2. **Малі об’єкти** ( px): додаємо попередню SR-підвищувальну мережу (наприклад, ESRGAN ×2) перед .
3. **FPS-тест:** якщо фактичний FPS , замінюємо на компактний MLP і повторюємо тест.
4. **Баланс витрат енергії:** якщо сумарне споживання енергії (W) перевищує 80 % паспортного TDP платформи, замінюємо RT‑DETR‑Tiny -> YOLOv11‑m та повторюємо FPS‑тест.

#### 1.5.4. Відтворюваність

Алгоритм забезпечує високу відтворюваність результатів.

* **Порогові значення** (48 px, 32 px, 16 px, TFLOPS = 5) обрано на базі емпіричних меж; у реплікації їх можна варіювати, але сам алгоритм (порівняння «> / ≤») залишається сталим.
* **Вхідні змінні** , , вимірюються однозначно; отже для тих самих даних і «заліза» вибір моделей повториться.
* Таблиця/псевдокод вище може бути реалізована в кілька рядків Python (if-elif-else), тому pipeline легко інтегрується у CI.

**Вихід:** для кожного рівня повертаємо пару (, ), гарантувавши, що вся ієрархія задовольняє і не потребує ручного тюнінгу.

#### 1.5.5. Дерево рішень (таблична форма)

Правила вибору моделей, описані в псевдокоді, можна представити у більш структурованій табличній формі (таблиця 3).

**Таблиця 3.** Дерево рішень для вибору моделей.

| Вхідні умови | Детектор | Вилучення | Класифікатор |
| --- | --- | --- | --- |
| та px | **YOLOv11-l** (≈130 FPS на RTX 3090) | Стандартний CSPDarknet-53 | **Linear MLP** (1 прих. шар, 256 нейронів) |
| та px | **YOLOv11-m** | CSPDarknet-53 + SPP-FPN | **FT-Transformer** (d=512, h=8, 2 блоки) |
| або px | **RT-DETR-Tiny** (ембединг 256) | CSPDarknet-53 + FPN + CARAFE | **FT-Transformer** (d=768, h=12, 4 блоки) |
| При дефіциті TFLOPS () | **YOLO-Nano** | Lightweight FPN | **MobileNet-MLP** |

### 1.6. Крок 6: Послідовне (ззовні → всередину) навчання

Навчання моделей відбувається послідовно, від найвищого (найзагальнішого) рівня ієрархії до найнижчого (найспецифічнішого). Моделі глибших рівнів навчаються лише на тих даних, які були правильно класифіковані на попередньому рівні, що дозволяє їм спеціалізуватися на розрізненні дедалі більш подібних класів.

1. Навчаємо (, , ) на всьому датасеті.
2. Для кожного наступного рівня : використовуємо тільки ті ROI, які попередній рівень відніс до надкласу, та навчаємо (, , ).
3. Для побудови **конкатенованого вектора ознак** шукаємо підмножину шарів максимізуючи вираз (5):

* аналогічно вибору у таблиці 1 і рис. 5 рукопису.

**Алгоритм вибору .** Використовуємо жадібну процедуру:

1. Стартуємо з найглибшого шару back‑bone.
2. На кожній ітерації додаємо той шар, що дає найбільший приріст (формула 5) на валідації.
3. Зупиняємося, коли приріст < 0.5 %, або кількість шарів > 3.

### 1.7. Крок 7: Калібрування порогів та перевірка ресурсних обмежень

Фінальний етап перед експортом — це тонке налаштування та верифікація. Для кожного класифікатора визначається оптимальний поріг впевненості, а вся система перевіряється на відповідність вимогам до швидкості та затримок. Якщо обмеження не виконуються, алгоритм може повернутися до попередніх кроків для спрощення ієрархії.

1. Для кожного калібруємо поріг довіри методом Youden‑J на ROC‑кривій, щоб мінімізувати error‑propagation униз по каскаду. Тут слугує оптимальним порогом довіри класифікатора .
2. Обчислюємо сумарну латентність і FPS; якщо не виконує — збільшуємо (отримаємо менше рівнів) та/або замінюємо моделі згідно з правилами з розділу 1.5.

**Застереження.** Після зміни у кроці 7 алгоритм **повторно** перераховує матрицю й запускає Крок 4, інакше отримана ієрархія буде несумісна з новим порогом.

### 1.8. Крок 8: Експорт структури

Результатом роботи алгоритму є готова до розгортання структура: конфігураційний файл, що описує ієрархію та призначені моделі, а також самі навчені моделі.

Формуємо JSON/YAML‑файл:

levels:  
 - id: 1  
 classes: ["H","V"]  
 detector: YOLOv11-l  
 classifier: FT-Transformer  
 threshold: 0.82  
 - id: 2  
 parent\_class: "V"  
 classes: ["T","B","O"]  
 detector: YOLOv11-m  
 classifier: FT-Transformer  
 threshold: 0.80  
 - id: 3  
 parent\_class: "O"  
 classes: ["G","M","Z"]  
 detector: YOLOv11-m  
 classifier: MLP  
 threshold: 0.78

* збережені ваги моделей.

**Вихідні дані:**

* Дерево ієрархії з моделями .
* Конфігураційний файл для розгортання на БПЛА чи наземній станції.
* Аналітичний звіт (таблиці метрик, граф FPS-vs-F1, дендрограма).

## 2. Глосарій

**Таблиця 4.** Опис усіх позначень.

| Позначення | Розмірність | Значення |
| --- | --- | --- |
|  | скаляр | Кількість зображень у датасеті. |
|  |  | -те зображення RGB. |
|  | скаляр з діапазону | Базова мітка з множини . |
|  | скаляр | Число базових (найдрібніших) класів. |
|  |  | Вектор ознак -го ROI на . |
|  | скаляр | Розмірність простору ознак після . |
|  |  | Центроїд ознак класу . |
|  |  | Матриця нормованих помилок між парами класів. |
|  | трикутна | Матриця комбінованої близькості (формула 3). |
|  | скаляр | Кінцева кількість ієрархічних рівнів. |
|  | множина | Підмножина класів, що розрізняється на рівні . |
|  | скаляр | Поточна кількість підкласів на рівні . |
|  | індекс | Номер рівня в ієрархії. |

## 3. Коротке обґрунтування автоматизованого підходу

**Таблиця 5.** Порівняння ручного та автоматичного підходів.

| Критерій | Ручна схема (2.4–2.5) | Автоматичний алгоритм |
| --- | --- | --- |
| Вибір рівнів/класів | емпірично | формула (4) |
| Підбір моделей | фіксований | дерево рішень (таблиця 3) |
| Пороги довіри | не описано | калібрування (розділ 1.7) |
| Підтримка FPS | декларативно | перевірка (розділ 1.7) |
| Масштабування на нові класи | ручне | повторити кроки 1.1–1.4 |

Алгоритм робить побудову структури **трасованою, відтворюваною** й адаптивною до нових наборів класів чи апаратних обмежень, зберігаючи концепцію, закладену авторами.