Table of Contents

Запропоновано формальний, але інтуїтивно зрозумілий алгоритм, який автоматично формує дерево рівнів / класів, спираючись на (і) відстані між векторними представленнями класів, (іі) величину плутанини 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.

# Запропонований метод

**Вхідні дані**

| Позначення | Розмірність | Пояснення |
| --- | --- | --- |
|  | (зображення + мітка) | 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 | Вага між «просторовою відстанню» й «частотою помилкових сплутувань» |
|  | 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) |

## Крок 1: Витяг ознак для усіх ROI

1.1 Для кожного зображення виконуємо первинне детектування (будь‑який SOTA‑детектор) та отримуємо множину ROI .

1.2 Для кожного ROI обчислюємо вектор ознак

## Крок 2: Побудова «профілю» класу

2.1 Для кожного атомарного класу збираємо множину ознак .

2.2 Обчислюємо **центроїд**

2.3 Одноразово навчаємо одноетапний класифікатор без ієрархії (наприклад, простий MLP) на всіх та формуємо **матрицю плутанини**

де — кількість неправильних присвоєнь .

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

## Крок 3: Визначення міри близькості між класами

Комбінуємо геометричну й емпіричну інформацію:

*Чим більшим є — тим легше класи розділити.*

## Крок 4: Ієрархічне агломеративне групування

4.1 Будуємо повнозважений граф з вагами та застосовуємо **agglomerative clustering (average linkage)**, отримуючи дендрограму .

4.2 Переходимо зверху донизу: для кожної внутрішньої вершини перевіряємо:

* 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.

## Крок 5. Призначення моделей на кожен рівень ієрархії

**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.

### 5.1 Вхідні параметри (усі легко вимірювані)

| Змінна | Розмірність / Тип | Як отримати | Що означає |  |  |
| --- | --- | --- | --- | --- | --- |
| ( | L\_k | ) | скаляр (ціле) | Злічити 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 |  |  |

### 5.2 Дерево рішень (детермінований алгоритм)

python if |L\_k| ≤ 2 and {s}\_k ≥ 48: D\_k = YOLOv11-l C\_k = MLP (1×256) elif 3 ≤ |L\_k| ≤ 6 and {s}\_k ≥ 32: D\_k = YOLOv11-m C\_k = FT-Transformer (d=512, h=8, 2 blocks) elif |L\_k| > 6 or {s}\_k < 32: D\_k = RT-DETR-Tiny C\_k = FT-Transformer (d=768, h=12, 4 blocks) # ресурсна деградація if C\_GPU < 5: D\_k = YOLO-Nano C\_k = MobileNet-MLP

### 5.3 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.

### 5.4 Відтворюваність

* **Порогові значення** (48 px, 32 px, 16 px, TFLOPS = 5) обрано на базі емпіричних меж з розд. 2.5 рукопису; 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 два рядки 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.

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

6.1 Навчаємо на всьому датасеті.

6.2 Для кожного наступного рівня : використовуємо тільки ті ROI, які попередній рівень відніс до надкласу, та навчаємо .

6.3 Для побудови **конкатенованого вектора ознак** шукаємо підмножину шарів максимізуючи

аналогічно вибору у табл. 1 і рис. 5 рукопису .

## Крок 7: Калібрування порогів та перевірка ресурсних обмежень

7.1 Для кожного калібруємо поріг довіри за критерієм Youden J.

7.2 Обчислюємо сумарну латентність і FPS; якщо не виконує — збільшуємо (отримаємо менше рівнів) та/або замінюємо моделі згідно (5)–(6).

## Крок 8: Експорт структури

Формуємо JSON / YAML‑файл

yaml 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-s 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).

## 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 (3) |  |  |
|  | скаляр | Кінцева кількість ієрархічних рівнів |  |  |
|  | множина | Підмножина класів, що розрізняється на рівні |  |  |
| ( | L\_k | ) | скаляр | Поточна кількість підкласів на рівні |
|  | індекс | Номер рівня в ієрархії |  |  |

## Коротке обґрунтування вигод

| Критерій | Ручна схема (2.4–2.5) | Автоматичний алгоритм |
| --- | --- | --- |
| Вибір рівнів / класів | емпірично | формула (4) |
| Підбір моделей | фіксований | дерево рішень (5–6) |
| Пороги довіри | не описано | калібрування 7.1 |
| Підтримка FPS | декларативно | перевірка 7.2 |
| Масштабування на нові класи | ручне | повторити кроки 1–4 |

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

## Крок 5. Призначення моделей кожному рівню (удосконалене дерево рішень)

Нехай

* — кількість підкласів, які треба розрізнити на рівні ;
* — медіанний розмір ROI (у пікселях) між усіма підкласами рівня;
* — доступна обчислювальна потужність (TFLOPS).

| Вхідні умови | Детектор | Витяг | Класифікатор |  |  |
| --- | --- | --- | --- | --- | --- |
| ( | L\_k | !!2) та px | **YOLOv11-l** (≈130 FPS на RTX 3090) | Стандартний CSPDarknet-53 | **Linear MLP** (1 прих. шар, 256 нейронів) |
| (3!! | L\_k | !!6) & px | **YOLOv11-m** | CSPDarknet-53 + SPP-FPN | **FT-Transformer** (d = 512, h = 8, 2 блоки) |
| ( | L\_k | !>!6) 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** |  |  |