YOLO 熱力圖與辨識結果批次生成器

（GradCAM/YOLOv8 一鍵產生結果圖）

一、工具功能簡介

本工具支援利用**YOLOv8模型**與各式 CAM（如 GradCAM、EigenCAM、HiResCAM...）方法，批次對資料夾內所有圖片進行辨識與熱力圖產生，並自動儲存**可視化結果圖**。

- **支援任務**：Detect、Segment、Pose、OBB、Classify
- **支援CAM方法**：GradCAMPlusPlus、GradCAM、XGradCAM、EigenCAM、HiResCAM、LayerCAM、RandomCAM、EigenGradCAM
- **圖像批次處理**：自動批次讀取資料夾，逐張產生偵測結果和熱力圖
- **產生檔案**：每張圖建立一個資料夾，內含 `detection.png`（YOLO預測）與 `heatmap.png`（可選是否疊加預測框/只在框內顯示）

二、操作步驟

1. 選擇輸入與輸出資料夾

- **輸入圖像資料夾**：選擇要批次處理的圖片所在的資料夾
- **輸出結果資料夾**：產生的可視化圖檔會放在這個資料夾內（自動建立子資料夾）

2. 選擇 YOLO 權重

- 按【選擇權重檔案】，指定 `.pt` 格式的YOLOv8權重檔

3. 設定推論與熱力圖參數

- **熱力圖方法**：選擇一種CAM方法（推薦 GradCAMPlusPlus 或 GradCAM）
- **模型任務**：選擇權重對應的任務（detect/segment/pose/obb/classify）
- **計算裝置**：有GPU時可選 cuda:0，否則選 cpu
- **目標層（Layers）**：指定YOLO模型中的目標層（通常不用更動，預設可用）
- **信心度閾值（Conf Threshold）**：過濾低信心度預測（0.2建議值）
- **目標比例（Ratio）**：可用於選擇前多少比例的目標（0.02 通常等同前1-2個偵測物件）
- **圖像尺寸**：推論時resize到的大小，建議640
- **在熱力圖上顯示辨識結果**：可勾選，則熱力圖上會畫出預測框
- **將熱力圖限制在框內（Renormalize）**：只顯示偵測框內的熱力圖，利於分析目標區域

4. 執行批次處理

- 點擊【開始處理】，下方狀態列顯示進度
- 處理過程中可按【停止】隨時中斷
- 每張圖像會在輸出資料夾下建立以檔名為名的子資料夾，裡面有
  - `detection.png`（YOLO預測繪圖結果）
  - `heatmap.png`（CAM可視化結果，依設定可能有疊加預測框）

三、結果結構說明

bash 複製程式碼

```
output_dir/
├── 圖片A/
│   ├── detection.png      # YOLO 預測繪圖
│   └── heatmap.png        # 熱力圖（疊加可選）
├── 圖片B/
│   ├── detection.png
│   └── heatmap.png
...
```

四、注意事項

- 權重檔必須為**YOLOv8系列**，可用 Ultralytics 官網訓練/下載
- 預設的 layer 可依據不同權重調整（通常10,12,14,16,18適用v8n/v8s/v8m）
- 如果遇到 "CUDA out of memory" 請切換到 cpu 或調低圖像尺寸
- 權重和任務類型要相符（如分類權重只能選 classify）
- 圖片格式支援：jpg/png/bmp/webp
- 支援中文/長路徑
- 狀態列會即時回報進度與錯誤

五、常見應用情境

- YOLO模型可解釋性分析、可視化每個預測物件的注意力區域
- 學術研究論文附圖
- 對比不同模型/增強前後資料的可視化差異

YOLOv8 架構:

```yaml
# YOLOv8.0n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]]  # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]]  # 1-P2/4
  - [-1, 3, C2f, [128, True]]
  - [-1, 1, Conv, [256, 3, 2]]  # 3-P3/8
  - [-1, 6, C2f, [256, True]]
  - [-1, 1, Conv, [512, 3, 2]]  # 5-P4/16
  - [-1, 6, C2f, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]]  # 7-P5/32
  - [-1, 3, C2f, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]]  # 9

# YOLOv8.0n head
head:
  - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
  - [[-1, 6], 1, Concat, [1]]  # cat backbone P4
  - [-1, 3, C2f, [512]]  # 12

  - [-1, 1, nn.Upsample, [None, 2, 'nearest']]
  - [[-1, 4], 1, Concat, [1]]  # cat backbone P3
  - [-1, 3, C2f, [256]]  # 15 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 12], 1, Concat, [1]]  # cat head P4
  - [-1, 3, C2f, [512]]  # 18 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 9], 1, Concat, [1]]  # cat head P5
  - [-1, 3, C2f, [1024]]  # 21 (P5/32-large)

  - [[15, 18, 21], 1, Detect, [nc]]  # Detect(P3, P4, P5)
```

YOLOv11 架構:

```yaml
# YOLO11n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 2, C3k2, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 2, C3k2, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
  - [-1, 2, C2PSA, [1024]] # 10

# YOLO11n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 2, C3k2, [512, False]] # 13

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 2, C3k2, [256, False]] # 16 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 13], 1, Concat, [1]] # cat head P4
  - [-1, 2, C3k2, [512, False]] # 19 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 10], 1, Concat, [1]] # cat head P5
  - [-1, 2, C3k2, [1024, True]] # 22 (P5/32-large)

  - [[16, 19, 22], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

1. YOLOv8/YOLOv11 的 backbone/head 結構快速解析

YOLOv8 主要組件

- **Backbone（特徵提取）：**
  - 0~9層，最後是 SPPF
- **Head（多尺度檢測/融合）：**
  - 10 之後，Upsample、Concat、C2f、Conv
  - 15(P3/8)、18(P4/16)、21(P5/32)：多尺度輸出
  - **Detect 層：最後一層，結合15/18/21作多尺度檢測**

YOLOv11 主要組件

- **Backbone：**
  - 0~10 (注意 C3k2、C2PSA、SPPF...)
- **Head：**
  - 11 之後，Upsample、Concat、C3k2、Conv
  - 16(P3/8)、19(P4/16)、22(P5/32)：多尺度
  - **Detect 層：最後一層，結合16/19/22**

2. 實際工具層編號對應說明

- **「目標層（layers）」參數**，即要輸入給 CAM/heatmap 工具的層索引清單，例如「10,12,14,16,18」。
- 這些「層」其實指的是**YOLO模型 .model 的 Sequential list 裡的模組序號**（即 YAML 配置檔每個[]的索引）。

如何選？

- **最常見做法：**
  取**backbone 最後幾層或head 中間幾層**（融合層、C2f/C3/C3k2/SPPF/C2PSA這類特徵提取豐富的層），這樣 CAM/heatmap 最有意義。
- **過於前面的層（如0,1）通常不建議**，因為只抓到低階邊緣特徵。

3. YOLOv8 / YOLOv11 實戰層索引舉例

YOLOv8

- **Backbone最後特徵層**：通常是 7,8,9（SPPF），對應深層語意特徵
- **Head中融合層**：像 12, 15, 18, 21
- **常用組合：** `10,12,14,16,18`
  - 例如：10(upsample), 12(C2f融合), 14(upsample), 16(C2f融合), 18(C2f融合)...
  - 這些層包含了多尺度和融合特徵

實際範例（v8n.yaml）：

yaml 複製程式碼

```yaml
9  # SPPF (backbone最後一層)
12 # C2f (head P4融合)
15 # C2f (head P3/8-small)
18 # C2f (head P4/16-medium)
21 # C2f (head P5/32-large)
```

建議 layers 參數填：

`"9,12,15,18,21"`

YOLOv11

- Backbone最後特徵層：10 (C2PSA, SPPF), 9
- Head融合層：13, 16, 19, 22
- 常用組合：`10,13,16,19,22`

實際範例（yolo11n.yaml）：

yaml 複製程式碼

```yaml
10 # C2PSA (backbone)
13 # C3k2 (head P4融合)
16 # C3k2 (head P3/8-small)
19 # C3k2 (head P4/16-medium)
22 # C3k2 (head P5/32-large)
```

建議 layers 參數填：

`"10,13,16,19,22"`

---

4. 如何實際操作你這個工具

步驟

1. 先查你的 YAML（或對應.py），確定想分析的層數字（如上）
2. 工具啟動後，"目標層(Layers)" 填入索引
   - 例如 `10,12,14,16,18` 或 `10,13,16,19,22`
3. 其它參數可用預設或依需求調整（method, conf, img_size...）
4. 開始批次分析即可，產生的 heatmap 會依你選的層疊加

---

5. 補充說明

- 如果你不確定用哪幾層，推薦先用 backbone 最後1層+head幾個融合層
- 輸出的 heatmap 會明顯區分不同層捕捉到的目標語意（越後面越聚焦於大區域目標）
- 不同模型結構，層數和命名不同，記得根據實際 YAML 修改 layers
- 若自訂模型有新模組，也用這種方法：查索引 → 輸入