PROJECT REPORT

January 20, 2026

Báo cáo cuối kỳ môn học: PYTHON CHO KHOA HỌC DỮ LIỆU

Lớp 23TTH, Khoa Toán - Tin học, Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM

Đề tài thực hiện:

USING DEEP LEARNING TO CLASSIFY ANIMAL AND HUMAN IMAGES

Giảng viên hướng dẫn: ThS. Hà Văn Thảo

Danh sách thành viên nhóm:

1. 23110114 - Nguyễn Thị Hồng Thắm
2. 23110123 - Lê Huỳnh Yến Vy
3. 23110132 - Trần Nhật Anh

1 GIỚI THIỆU

Object detection là một trong những chủ đề "nóng" trong deep learing bởi tính ứng dụng cao trong thực tiễn và nguồn dữ liệu dồi dào, dễ chuẩn bị. Một trong những thuật toán object detection nổi tiếng nhất là **YOLO**.

YOLO là mô hình mạng neuron tích chập (CNN) được sử dụng phổ biển để nhận dạng các đối tượng trong ảnh hoặc video. Điểm đặc biệt của mô hình này là có khả năng phát hiện tất cả các đối tượng trong một hình ảnh chỉ qua một lần lan truyền của CNN.

Các phương pháp truyền thống tách biệt bước đề xuất vùng và bước phân loại, YOLO xử lý toàn bộ hình ảnh trong một lần lan truyền duy nhất qua mạng CNN. Cơ chế này cho phép mô hình vừa định vị (localization), vừa phân loại (classification) đối tượng cùng lúc.

YOLO có nghĩa là "You only look once", thuật toán chỉ cần "nhìn" một lần để có thể đưa ra dự đoán, giúp đạt được tốc độ xử lý cực nhanh, tiệm cận thời gian thực (real-time) mà vẫn đảm bảo độ chính xác cao. Vì vậy, YOLO được ứng dụng rộng rãi trong đa dạng lĩnh vực: từ quản lý giao thông thông minh, giám sát dây chuyền sản xuất, nông nghiệp công nghệ cao (đếm vật nuôi,...), cho đến các hệ thống an ninh giám sát (phát hiện vũ khí, chấm công tự động,...),...

2 TẠO MÔI TRƯỜNG ẢO VÀ KERNEL CHẠY NOTEBOOK (LINUX)

Dự án Python cần **môi trường ảo (virtual environment)** để tự cách ly, tránh xung đột phiên bản thư viện giữa các dự án. `venv` là môi trường ảo mà chúng ta sẽ sử dụng trong dự án này. Sau khi cài đặt `venv`, chúng ta di chuyển đường dẫn đến folder chứa dự án trong terminal và sử dụng lệnh sau để cài đặt môi trường ảo cho dự án:

```
python -m venv .venv
```

Trong đó, `.venv` là tên của folder chứa môi trường ảo của dự án, đồng thời nó cũng sẽ "đóng băng" phiên bản Python, pip và các thư viện sẽ được dùng trong dự án.

Kích hoạt môi trường ảo:

```
source .venv/bin/activate
```

Lúc này, phiên bản Python và `pip` được dùng là của môi trường ảo, các thư viện cài bằng `pip install` cũng chỉ ảnh hưởng trong `.venv`. Cách nhận biết đang ở môi trường ảo là promt terminal thường đổi thành `(.venv) user_name@machine:~` (nếu đang sử dụng Linux). Khi đã kích hoạt môi trường ảo, đảm bảo phiên bản Python và `pip` đã "đóng băng" trong đó, sử dụng lệnh:

```
which python && which pip
```

Nếu output có dạng `.../<project_name>/.venv/...` thì môi trường ảo đã được kích hoạt thành công.

Tiếp theo, tạo một kernel để chạy Jupyter Notebook. Cài đặt `ipykernel` để tạo kernel:

```
python -m pip install ipykernel
```

Sau khi cài đặt thành công, tiến hành tạo kernel để chạy file `.ipynb`:

```
python -m ipykernel install --prefix .venv --name yolovenv --display-name
"this_project"
```

`--prefix .venv`: kernel mặc định không tự lưu vào `.venv`, thuộc tính này sẽ lưu kernel đã tạo vào `.venv`
`--name yolovenv`: tên folder chứa kernel, ở đây tên folder là `yolovenv`. Kernel sẽ được lưu tại `.venv/share/jupyter/kernels/yolovenv/`
`--display-name "this_project`: kernel sẽ hiển thị dưới tên `this_project` trong VS Code.

Khi đã tạo kernel, click vào biểu tượng kernel ở góc trên bên phải, chọn

Select Another Kernel → Jupyter Kernel... → this_project

3 CÁC THƯ VIỆN CẦN DÙNG

[1]:
```
from collections import Counter
import cv2
from itertools import product
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
from PIL import Image
import pandas as pd
from pathlib import Path
import seaborn as sns
from ultralytics import YOLO
```

4 CHUẨN BỊ VÀ TỔNG QUAN VỀ DỮ LIỆU

4.1 CHUẨN BỊ DỮ LIỆU

Dữ liệu được tải về tại các nguồn sau:
- https://www.kaggle.com/datasets/antoreepjana/animals-detection-images-dataset
- https://www.kaggle.com/datasets/biancaferreira/african-wildlife
- https://www.kaggle.com/datasets/wutheringwang/dog-face-detectionyolo-format
- https://www.kaggle.com/datasets/samuelayman/cat-dataset
- https://universe.roboflow.com/labo-yolo/age-and-gender-xlnfj/dataset/3

Dữ liệu sau khi được tải về sẽ được xử lý (gán lại class ID; phân loại thành các folder train, valid, test;…), sau đó được gộp thành một folder tập dữ liệu (dataset) duy nhất. Dataset nếu muốn được YOLO "hiểu" thì phải có file `.yaml` chứa các thông tin về dataset như vị trí của file train, valid, test; số class; tên của các class ứng với mỗi class_id;…

Dataset của dự án này nằm tại `dataset/complete_dataset`, bao gồm:
- folder `images` chứa các file ảnh, được chia thành ba folder train, valid, test.
- folder `labels` chứa các file **nhãn dữ liệu (label)** có đuôi `.txt`, cũng được chia thành ba folder train, valid, test và các file label có tên ứng với các file ảnh.
- file `.yaml` để cung cấp thông tin về dataset cho YOLO.

[2]:
```
DATASET_PATH = "dataset/complete_dataset/data.yaml"
BASE_DIR = Path("dataset/complete_dataset")
IMAGES_DIR = "dataset/complete_dataset/images"
LABELS_DIR = "dataset/complete_dataset/labels"
```

4.2 LÀM SẠCH VÀ CHUẨN HOÁ DỮ LIỆU

Dữ liệu hình ảnh được tổng hợp từ nhiều nguồn khác nhau thường tìm ẩn các vấn đề về tính toàn vẹn, chẳng hạn như sự không đồng nhất về định dạng (PNG, JPG, WEBP…), lỗi header của file ảnh hoặc file bị hỏng hoàn toàn (corrupt). Những vấn đề này có thể gây ra lỗi ngắt quãng (crash) đột ngột cho thư viện huấn luyện (như PyTorch/YOLO) khi mô hình cố gắng đọc dữ liệu, làm gián đoạn quá trình huấn luyện kéo dài.

Để khắc phục, cần làm sạch và chuẩn hóa dataset thông qua hàm `resave_images` để: - Chuẩn hóa định dạng: Mở từng file ảnh, chuyển đổi hệ màu về RGB và lưu lại (re-save) dưới định dạng JPEG để đảm bảo tính nhất quán.
- Xử lý lỗi tự động: Nếu phát hiện file ảnh bị lỗi không thể đọc được (corrupt), tự động xóa file ảnh đó, đồng thời xóa luôn file nhãn (label) tương ứng để tránh dư thừa label.
- Tối ưu hóa quy trình: Để tránh việc phải xử lý lại toàn bộ dữ liệu (tốn nhiều thời gian) mỗi khi chạy lại file Jupyter Notebook, cần tạo một cơ chế đánh dấu bằng việc kiểm tra liệu có tồn tại file

`flag_file.txt` trong folder chứa dataset. Quá trình chuẩn hóa chỉ được thực hiện nếu file này chưa tồn tại.

Kiểm tra xem dataset đã từng được xử lý chưa thông qua hàm `check_flag_file`.

[3]:
```python
def check_flag_file(BASE_DIR):
    flag_file = BASE_DIR / "flag_file.txt"

    if not flag_file.exists():
        return False

    with open(flag_file, 'r') as file:
        return file.read().strip() == "This dataset has been re-saved already."
```

Hàm `create_flag_file` dùng để tạo flag file.

[4]:
```python
def create_flag_file(BASE_DIR):
    if check_flag_file(BASE_DIR):
        return

    flag_file = BASE_DIR / "flag_file.txt"
    content = "This dataset has been re-saved already."

    with open(flag_file, 'w') as file:
        file.write(content)
```

Thực hiện chuẩn hóa và loại bỏ ảnh lỗi. Hàm `resave` sẽ duyệt qua toàn bộ thư mục, thực hiện việc xác thực (verify) và lưu lại ảnh. Những ảnh bị lỗi Exception sẽ bị xóa cùng với file label của chúng.

[5]:
```python
def resave(IMAGES_DIR, LABELS_DIR):
    fixed = 0
    removed = 0

    for root, _, files in os.walk(IMAGES_DIR):
        for file in files:
            if not file.lower().endswith((".jpg", ".jpeg", ".png", ".webp")):
                continue

            # Ghép IMAGES_DIR với tên file ảnh thành path file ảnh
            image_path = os.path.join(root, file)

            try:
                # Mở ảnh để verify nhanh, nếu ảnh có vấn đề -> exception
                with Image.open(image_path) as image:
                    image.verify()

                # Mở lại để re-save
                with Image.open(image_path) as image:
```

```
                    image = image.convert("RGB")
                    image.save(image_path, "JPEG", quality=95, subsampling=0)

                fixed += 1

            except Exception:
                # Nếu ảnh hỏng -> xoá file ảnh cùng với file label tương ứng
                os.remove(image_path)
                removed += 1

                relative_path = os.path.relpath(root, IMAGES_DIR)

                label_path = os.path.join(
                    LABELS_DIR,
                    relative_path,
                    os.path.splitext(file)[0] + ".txt"
                )

                if os.path.exists(label_path):
                    os.remove(label_path)

    return fixed, removed
```

Hàm `resave_images` điều phối quá trình và thông báo số lượng ảnh đã sửa hoặc đã xóa.

[6]:
```
def resave_images(BASE_DIR, IMAGES_DIR, LABELS_DIR):
    if check_flag_file(BASE_DIR):
        return 0, 0
    else:
        fixed, removed = resave(IMAGES_DIR, LABELS_DIR)
        create_flag_file(BASE_DIR)

        return fixed, removed
```

[7]:
```
fixed, removed = resave_images(BASE_DIR, IMAGES_DIR, LABELS_DIR)

print("Fixed  :", fixed)
print("Removed:", removed)
```

```
Fixed  : 0
Removed: 0
```

4.3 KIỂM TRA FILE LABEL TƯƠNG ỨNG VỚI FILE ẢNH

Sau khi hoàn tất xử lý dataset, dùng hàm `count_missing` để kiểm tra xem với mỗi file ảnh thì có file label tương ứng hay không. Cần đảm bảo 100% ảnh đầu vào đều có nhãn tương ứng (0% missing labels).

Sau bước làm sạch và chuẩn hoá dataset, tiến hành kiểm tra tính toàn vẹn của dữ liệu thông qua

hàm `count_missing`. Mục tiêu của bước này là rà soát đối sánh từng file ảnh với file label tương ứng trong các thư mục con (train, valid, test) để đảm bảo rằng mọi dữ liệu đưa vào mô hình đều hợp lệ và đầy đủ thông tin.

Hàm `count_missing` được xây dựng để duyệt qua từng file ảnh và kiểm tra sự tồn tại của file label `.txt` tương ứng. Cần đảm bảo toàn bộ ảnh đều có file label (0% missing labels).

[8]:
```python
def count_missing(BASE_DIR, image_exts, sub_folder):
    images_dir = BASE_DIR / "images" / sub_folder
    labels_dir = BASE_DIR / "labels" / sub_folder

    images_cnt = 0
    missing_cnt = 0

    for image in images_dir.iterdir():
        if image.suffix.lower() not in image_exts:
            continue

        images_cnt += 1
        """
        image.stem dùng để lấy tên của file image mà không có phần mở rộng
        Ví dụ: example123.png -> example123
        """
        label_file = labels_dir / f"{image.stem}.txt"
        if not label_file.exists():
            missing_cnt += 1

    return images_cnt, missing_cnt
```

Tiến hành chạy kiểm tra trên toàn bộ dataset và tổng hợp kết quả vào bảng để dễ dàng quan sát tỷ lệ lỗi.

[9]:
```python
image_exts = {".jpg", ".jpeg", ".png", ".webp"}
sub_folders = ["train", "valid", "test"]

print(f"Checking dataset: {str(BASE_DIR)}\n")

total_images = 0
total_missing = 0
images_cnt_arr = np.array([])
missing_cnt_arr = np.array([])
missing_percent_arr = np.array([])

for sub_folder in sub_folders:
    images_cnt, missing_cnt = count_missing(BASE_DIR, image_exts, sub_folder)
    total_images += images_cnt
    total_missing += missing_cnt
```

```
    missing_percent = (missing_cnt / images_cnt) * 100 if images_cnt > 0 else 0

    images_cnt_arr = np.append(images_cnt_arr, images_cnt)
    missing_cnt_arr = np.append(missing_cnt_arr, missing_cnt)
    missing_percent_arr = np.append(missing_percent_arr, missing_percent)

images_percent = np.array([(images_cnt / total_images) * 100 \
                           for images_cnt in images_cnt_arr])

np_table = np.array([images_cnt_arr, images_percent, \
                     missing_cnt_arr, missing_percent_arr])
table = pd.DataFrame(np_table).transpose()
table.columns = ["images", "images/total (%)",
                 "missing_labels", "missing_labels (%)"]
table.index = ["train", "valid", "test"]
table['images'] = table['images'].astype(int)
table['missing_labels'] = table['missing_labels'].astype(int)

if total_images > 0:
    missing_percent = (total_missing / total_images) * 100
else:
    missing_percent = 0

print(table)
print("")
print(f"-> Total images:       {total_images}")
print(f"-> Total missing:      {total_missing}")
print(f"-> Total missing (%):  {missing_percent}")
```

```
Checking dataset: dataset/complete_dataset

       images  images/total (%)  missing_labels  missing_labels (%)
train   20451         65.619585               0                 0.0
valid    6078         19.502021               0                 0.0
test     4637         14.878393               0                 0.0

-> Total images:       31166
-> Total missing:      0
-> Total missing (%):  0.0
```

Kết quả cho thấy toàn bộ 31166 ảnh đều có đủ file label tương ứng (`Total missing: 0`), dữ liệu đảm bảo điều kiện để huấn luyện mô hình.

4.4 PHÂN BỐ CÁC CLASS

Việc thống kê số object của mỗi class nhằm xác định mức độ mất cân bằng dữ liệu giữa 85 class của dataset, nhận diện lớp chiếm ưu thế và các lớp thiểu số hoặc các lớp rỗng để lường trước những nhược điểm, hạn chế của mô hình.

[10]:
```
CLASS_NUM = 85
```

Ngoài ra cũng có thể có nhiều object trong mỗi ảnh nên tổng số object luôn nhiều hơn hoặc bằng số ảnh. Cụ thể, dataset này có tổng cộng 36243 objects so với 31166 ảnh.

[11]:
```
LABELS_DIR = BASE_DIR / "labels"

counter = Counter()

for label_file in LABELS_DIR.rglob("*.txt"):
    with open(label_file, "r") as file:
        for line in file:
            if line.strip():
                class_id = int(line.split()[0])
                counter[class_id] += 1

df = pd.DataFrame(
    sorted(counter.items()),
    columns=["class_id", "object_count"]
)

df = df.sort_values("object_count", ascending=False, ignore_index=True)
total_objects = df["object_count"].sum()
df["object_count (%)"] = (df["object_count"] * 100 / total_objects).round(5)
print(f"Total objects: {total_objects}")
```

```
Total objects: 36243
```

Có 85 class nhưng bảng `df` chỉ có 79 dòng, nghĩa là có 6 class không có object.

[12]:
```
df
```

[12]:

	class_id	object_count	object_count (%)
0	83	6340	17.49303
1	3	2774	7.65389
2	17	1647	4.54433
3	48	1603	4.42292
4	36	1246	3.43791
..	…	…	…
74	24	33	0.09105
75	75	26	0.07174
76	67	16	0.04415
77	1	8	0.02207
78	59	7	0.01931

[79 rows x 3 columns]

Các `class_id` không có object nào là 25, 40, 50, 54, 57, 58.

[13]:
```
all_classes = set(range(CLASS_NUM))
found_classes = set(counter.keys())

missing_classes = sorted(all_classes - found_classes)
print("Missing class_ids:", missing_classes)
print("Number of missing classes:", len(missing_classes))
```

```
Missing class_ids: [25, 40, 50, 54, 57, 58]
Number of missing classes: 6
```

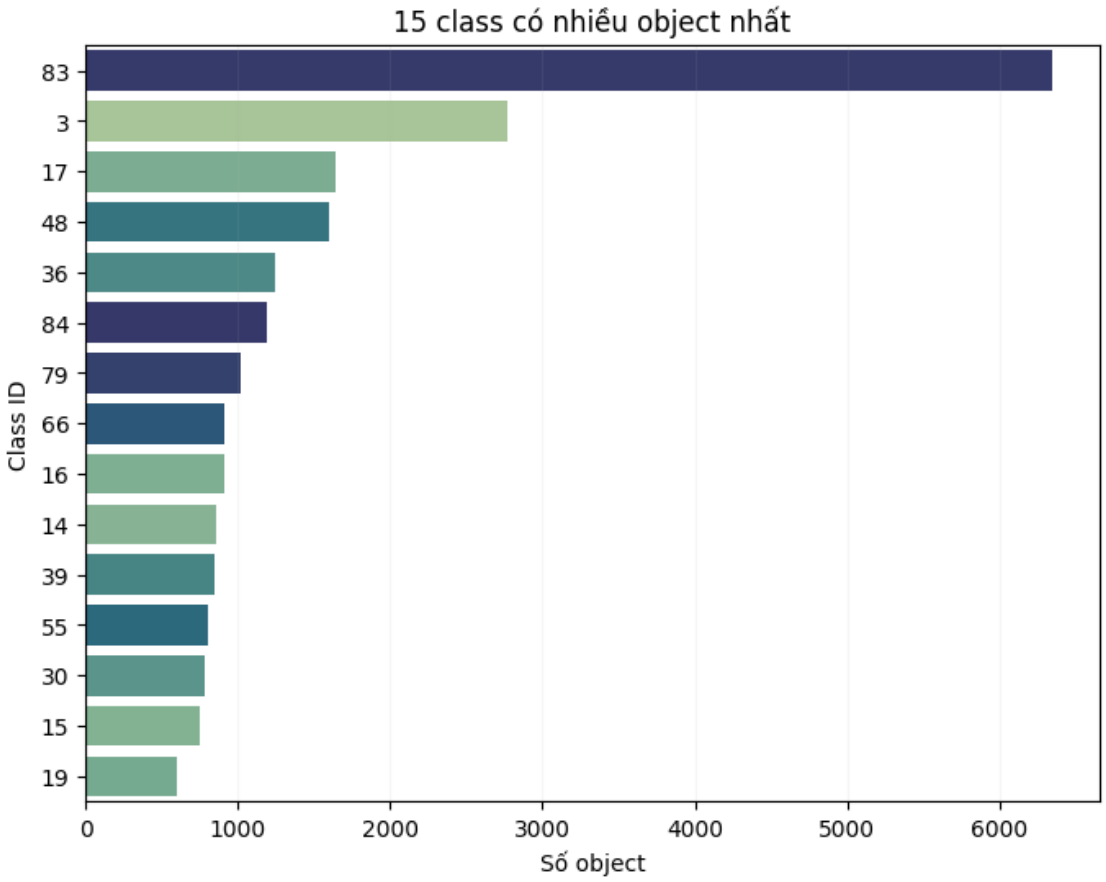
[14]:
```
# Sắp xếp df theo thứ tự giảm dần của cột object_count rồi lấy 15 dòng đầu tiên
df_top15 = df.sort_values(by="object_count",
                          ascending=False,
                          ignore_index=True).head(15)

df_top15
```

[14]:

	class_id	object_count	object_count (%)
0	83	6340	17.49303
1	3	2774	7.65389
2	17	1647	4.54433
3	48	1603	4.42292
4	36	1246	3.43791
5	84	1188	3.27787
6	79	1021	2.81710
7	66	915	2.52463
8	16	913	2.51911
9	14	864	2.38391
10	39	849	2.34252
11	55	809	2.23216
12	30	782	2.15766
13	15	746	2.05833
14	19	598	1.64997

[15]:
```
plt.figure(figsize=(8, 6))

order = (
    df_top15
    .sort_values("object_count", ascending=False)["class_id"]
    .astype(str)
)

sns.barplot(
    data=df_top15,
    x="object_count",
    y=df_top15["class_id"].astype(str),
    order=order,
    hue="class_id",
```

```
    palette="crest",
    legend=False,
    errorbar=None
)

plt.xlabel("Số object")
plt.ylabel("Class ID")
plt.title("15 class có nhiều object nhất")
plt.grid(axis='x', alpha=0.1)
plt.show()
```

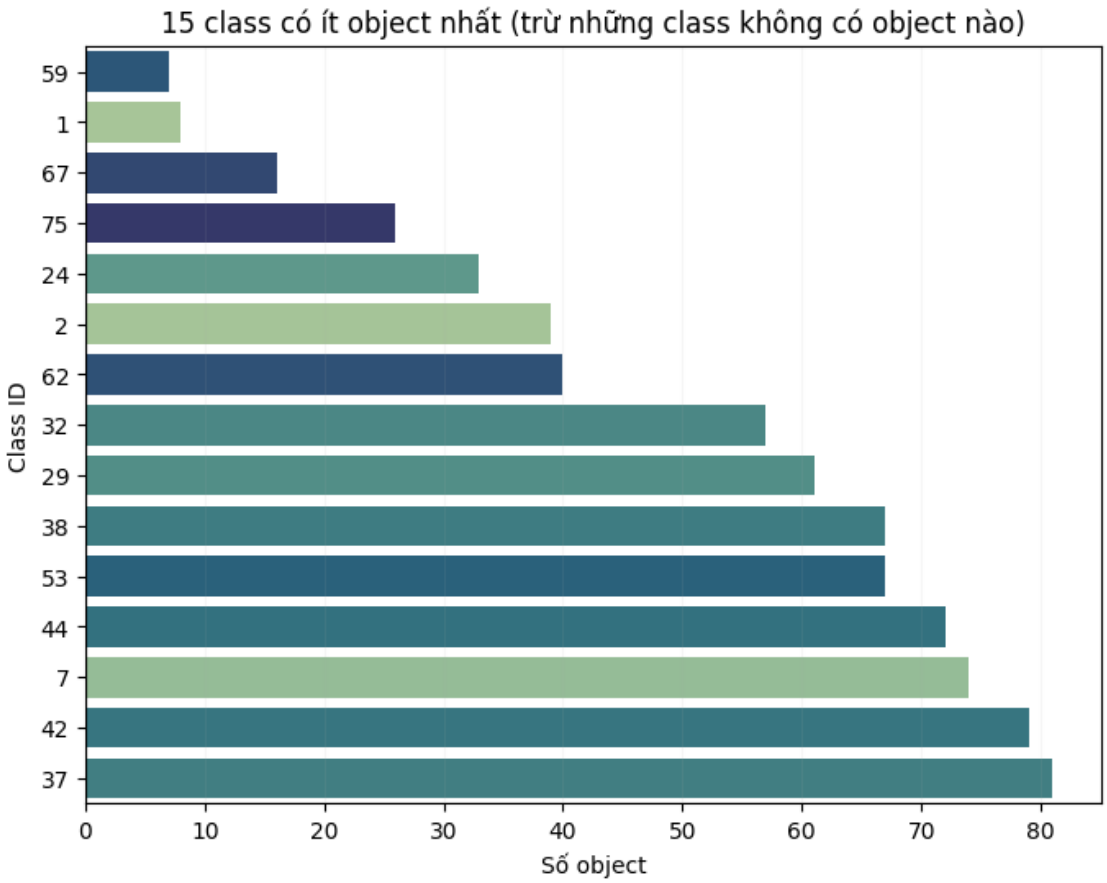


[16]:
```
# Sắp xếp df theo thứ tự tăng dần của cột object_count rồi lấy 15 dòng đầu tiên
df_bottom15 = df.sort_values(by="object_count",
                             ascending=True,
                             ignore_index=True).head(15)

df_bottom15
```

[16]:

	class_id	object_count	object_count (%)
0	59	7	0.01931
1	1	8	0.02207
2	67	16	0.04415
3	75	26	0.07174
4	24	33	0.09105
5	2	39	0.10761
6	62	40	0.11037
7	32	57	0.15727
8	29	61	0.16831
9	38	67	0.18486
10	53	67	0.18486
11	44	72	0.19866
12	7	74	0.20418
13	42	79	0.21797
14	37	81	0.22349

[17]:
```
plt.figure(figsize=(8, 6))

order = (
    df_bottom15
    .sort_values("object_count", ascending=True)["class_id"]
    .astype(str)
)

sns.barplot(
    data=df_bottom15,
    x="object_count",
    y=df_bottom15["class_id"].astype(str),
    order=order,
    hue="class_id",
    palette="crest",
    legend=False,
    errorbar=None
)

plt.xlabel("Số object")
plt.ylabel("Class ID")
plt.title("15 class có ít object nhất (trừ những class không có object nào)")
plt.grid(axis='x', alpha=0.1)
plt.show()
```



Phân bố số lượng đối tượng (object) giữa các class là không đồng đều. Quan sát hai biểu đồ trên, có thể thấy sự mất cân bằng nghiêm trọng giữa các class. Cụ thể: - Class có ID 83 chiếm số lượng áp đảo với 6340 objects (khoảng 17.5%), gấp hơn 2 lần so với class đứng thứ hai là 3 (2774 đối tượng, chiếm khoảng 7.65%). - Ngược lại, có những lớp có dữ liệu cực ít như class 59 (7 objects), class 1 (8 objects). Thậm chí, có 6 class (25, 40, 50, 54, 57, 58) hoàn toàn không có object nào.

Điều này sẽ khiến mô hình có xu hướng học thiên lệch (bias), nhận diện tốt các class có nhiều object nhưng dễ bỏ qua các class chiếm thiểu số, `recall` (độ nhạy) thấp cho các class thiểu số, `precision` (độ chính xác) có vẻ cao nhưng `mAP` giữa các class không đồng đều,…

Sự mất cân bằng nghiêm trọng về số lượng mẫu giữa các lớp (class imbalance) tạo ra rào cản lớn cho quá trình tối ưu hóa: hàm mất mát (loss function) sẽ bị chi phối bởi các lớp đa số, khiến mạng neuron có xu hướng học thiên lệch (bias) để giảm thiểu sai số cục bộ. Hệ quả là mô hình có thể đạt độ chính xác tổng thể cao nhờ đoán đúng các lớp phổ biến, nhưng lại thất bại hoặc có `recall` rất thấp khi nhận diện các lớp thiểu số do không đủ dữ liệu để học các đặc trưng tổng quát.

4.5 PHÂN BỐ KÍCH THƯỚC ẢNH

Bên cạnh vấn đề mất cân bằng về số lượng object giữa các class (như đã phân tích ở trên) thì độ phân giải hình ảnh (image resolution) cũng là một yếu tố then chốt ảnh hưởng đến hiệu năng của

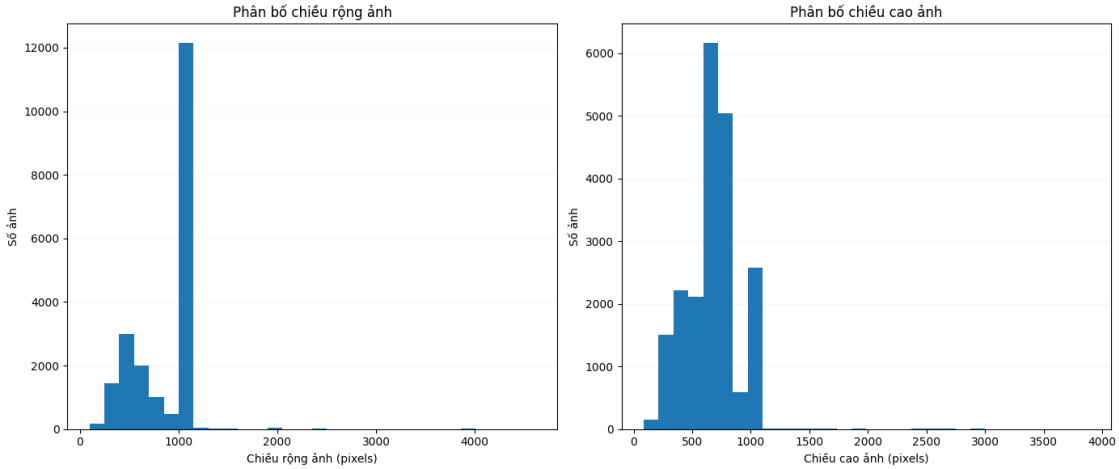
mô hình. Do bộ dữ liệu được tổng hợp từ nhiều nguồn khác nhau (Kaggle, Roboflow…), sự không đồng nhất về kích thước ảnh (chiều rộng và chiều cao) là điều không thể tránh khỏi.

Vì vậy, cần thực hiện phân tích thống kê chiều rộng và chiều cao của ảnh thuộc file train. Kết quả phân bố này là cơ sở thực tiễn để chúng ta đưa ra quyết định lựa chọn `imgsz` phù hợp nhất cho các thực nghiệm ở phần sau.

[18]:
```
# Chỉ cần kiểm tra phân bố kích thước ảnh của file train
TRAIN_IMAGES_DIR = Path("dataset/complete_dataset/images/train")

widths = np.array([])
heights = np.array([])

image_paths = [
    p for p in TRAIN_IMAGES_DIR.iterdir()
    if p.suffix.lower() in {".jpg", ".png", ".jpeg", ".webp"}
]

for image_path in image_paths:
    image = cv2.imread(str(image_path))
    if image is None:
        continue

    height, width, _ = image.shape
    widths = np.append(widths, width)
    heights = np.append(heights, height)
```

Biểu đồ phân bố kích thước ảnh.

[19]:
```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(widths, bins=30)
plt.xlabel("Chiều rộng ảnh (pixels)")
plt.ylabel("Số ảnh")
plt.title("Phân bố chiều rộng ảnh")
plt.grid(axis='y', alpha=0.1)
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.hist(heights, bins=30)
plt.xlabel("Chiều cao ảnh (pixels)")
plt.ylabel("Số ảnh")
plt.title("Phân bố chiều cao ảnh")
plt.grid(axis='y', alpha=0.1)
plt.tight_layout()

plt.show()
```



Quan sát biểu đồ tần suất (histogram) của chiều rộng ảnh, ta thấy sự tập trung mật độ cao nhất nằm ở khoảng 1000 pixels. Tuy nhiên, phổ kích thước trải rất rộng, xuất hiện cả những ảnh có chiều ngang rất nhỏ (dưới 500 pixels) và cực lớn (lên tới 4000 pixels).

Tương tự, chiều cao của ảnh cũng không đồng nhất. Đa số các ảnh có chiều cao tập trung trong khoảng 600 đến 800 pixels (thể hiện qua cột cao nhất trên biểu đồ). Bên cạnh đó, vẫn tồn tại một lượng đáng kể các ảnh có chiều cao vượt trội (trên 2000 - 3000 pixels) hoặc rất thấp.

Các biểu đồ phân bố trên đã chỉ ra sự không đồng nhất lớn về kích thước của các ảnh dữ liệu đầu vào, với độ phân giải trải dài từ thấp đến cao. Điều này sẽ ảnh hưởng đến việc lựa chọn `imgsz` phù hợp trong khi huấn luyện mô hình. Mặc dù `imgsz` cao giúp bảo toàn thông tin chi tiết của ảnh gốc, nâng cao hiệu quả nhận diện, đặc biệt là với những ảnh có object nhỏ. Tuy nhiên, đổi lại là thời gian và tài nguyên tính toán.

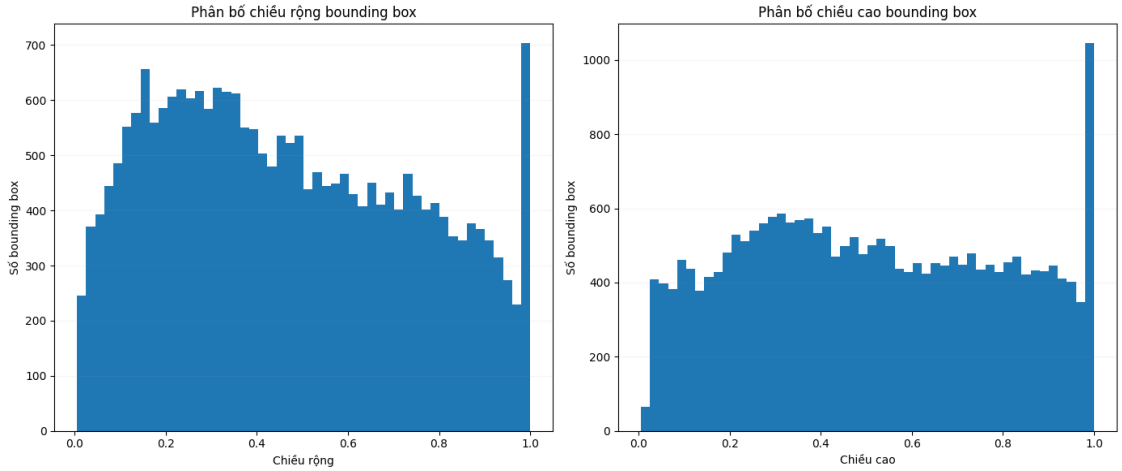
4.6 PHÂN BỐ BOUNDING BOX

[20]:
```
TRAIN_LABELS_DIR = Path("dataset/complete_dataset/labels/train")

bbox_widths = np.array([])
bbox_heights = np.array([])
bbox_areas = np.array([])

label_paths = [
    p for p in TRAIN_LABELS_DIR.iterdir()
    if p.suffix == ".txt"
]

for label_path in label_paths:
    with open(label_path, 'r') as label:
        lines = label.readlines()

    for line in lines:
```

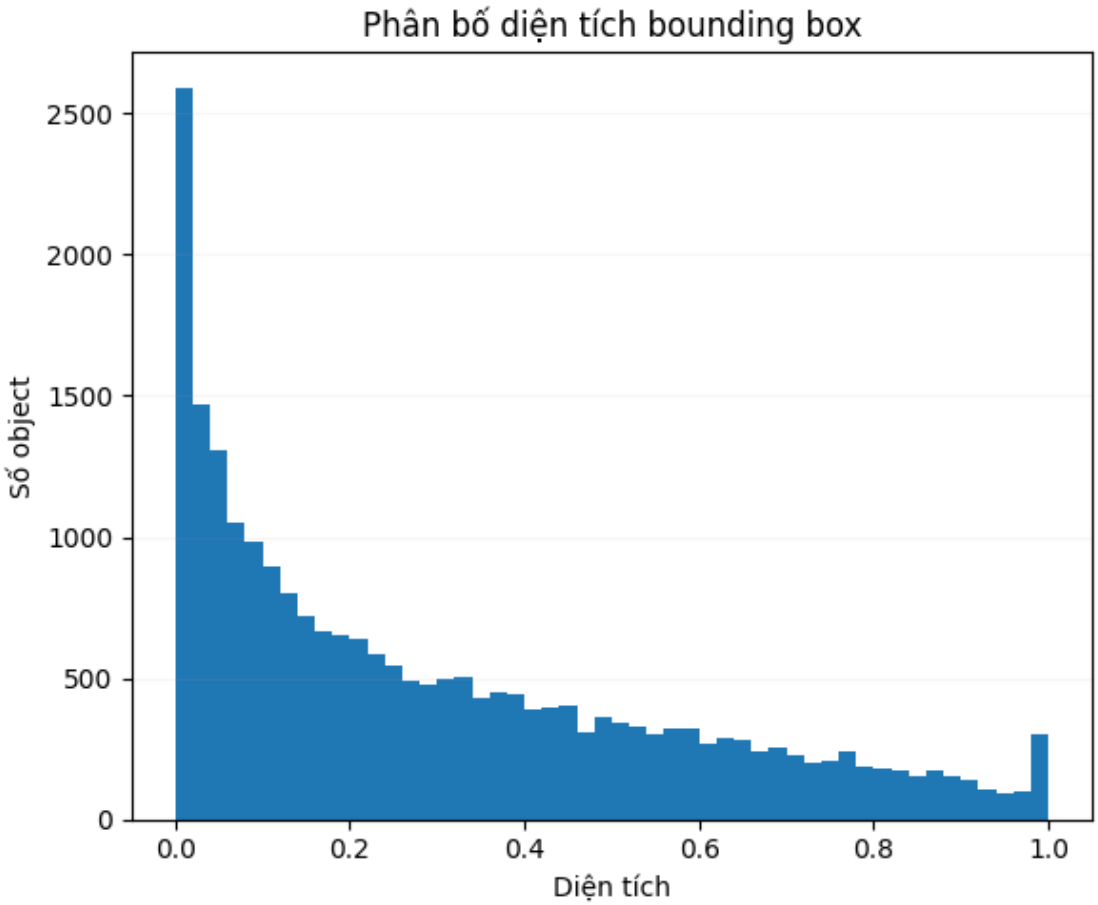
```
        parts = line.strip().split()
        if len(parts) != 5:
            continue

        _, x_center, y_center, width, height = map(float, parts)

        bbox_widths = np.append(bbox_widths, width)
        bbox_heights = np.append(bbox_heights, height)
        bbox_areas = np.append(bbox_areas, width * height)

print(f"Total bounding boxes: {len(bbox_areas)}")
print(f"Mean bbox width     : {bbox_widths.mean():.4f}")
print(f"Mean bbox height    : {bbox_heights.mean():.4f}")
print(f"Mean bbox area      : {bbox_areas.mean():.4f}")
```

```
Total bounding boxes: 23628
Mean bbox width     : 0.4705
Mean bbox height    : 0.5141
Mean bbox area      : 0.3012
```

[21]:
```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(bbox_widths, bins=50)
plt.title("Phân bố chiều rộng bounding box")
plt.xlabel("Chiều rộng")
plt.ylabel("Số bounding box")
plt.grid(axis='y', alpha=0.1)
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.hist(bbox_heights, bins=50)
plt.title("Phân bố chiều cao bounding box")
plt.xlabel("Chiều cao")
plt.ylabel("Số bounding box")
plt.grid(axis='y', alpha=0.1)
plt.tight_layout()

plt.show()
```



[22]:
```
plt.figure(figsize=(6, 5))
plt.hist(bbox_areas, bins=50)
plt.title("Phân bố diện tích bounding box")
plt.xlabel("Diện tích")
plt.ylabel("Số object")
plt.grid(axis='y', alpha=0.1)
plt.tight_layout()
plt.show()
```



Phân bố bounding box cho thấy ở các cột bên trái của mỗi biểu đồ thường cao hơn, nghĩa là có nhiều object có kích thước nhỏ. Nhưng nhìn chung, kích thước của object trải dài rất nhỏ đến rất lớn so với toàn bộ ảnh, thể hiện qua số object tập trung nhiều ở vùng kích thước nhỏ.

Vì vậy, việc sử dụng kích thước ảnh đầu vào lớn hơn `imgsz=640` có thể giúp mô hình học được đặc trưng tốt hơn cho các object nhỏ, trong khi `imgsz=448` vẫn được thử nghiệm để đánh giá sự cân bằng giữa độ chính xác và chi phí tính toán.

5 KHAI BÁO, HUẤN LUYỆN VÀ LƯU MÔ HÌNH

Phiên bản các thư viện dùng để huấn luyện mô hình:
- Ultralytics 8.3.243
- Python-3.12.8 torch-2.9.1+cu128 CUDA:0 (NVIDIA RTX A6000, 48541MiB)

Mô hình sẽ được huấn luyện với hai hàm `train_model_with_SGD` và `train_model_with_AdamW`. Hàm `train_model_with_AdamW` có thêm các tham số tối ưu hoá, huấn luyện mô hình để cho ra các kết quả dùng để so sánh ở các phần sau.

Optimizer mặc định của YOLO11s/YOLO11m là `SGD`, mô hình sẽ được huấn luyện để so sánh với optimizer `AdamW`.

Ngoài ra, dùng `seed=42` là giá trị hạt giống ngẫu nhiên (`random seed`), dùng để cố định tính ngẫu nhiên trong quá trình huấn luyện. Giữa các lần chạy hàm `train`, cách chia dữ liệu train/val, thứ tự ảnh huấn luyện, trọng số ban đầu,... sẽ giống nhau. Điều này nhằm đảm bảo công bằng khi so sánh nhiều mô hình, nhiều tổ hợp tham số khác nhau. Tuy nhiên, không có công bằng "tuyệt đối" vì có tồn tại thứ tự tính toán khác nhau, sai số ở một vài chữ số thập phân,...

[23]:
```python
def train_model_with_SGD(model, epochs, imgsz, dataset_path, project_path,
                         project_name):
    model.train(
        exist_ok=True,
        data=dataset_path,
        project=project_path,
        name=project_name,

        epochs=epochs,
        imgsz=imgsz,
        batch=32,
        workers=12,
        seed=42,
        device=0,  # sử dụng GPU

        cos_lr=True,

        mosaic=0.1,
        mixup=0.0,
        copy_paste=0.0,

        optimizer="SGD",
        lr0=0.01,
        momentum=0.937,
        weight_decay=5e-4,
    )


def train_model_with_AdamW(model, epochs, imgsz, dataset_path, project_path,
                           project_name):
    model.train(
        exist_ok=True,
        data=dataset_path,
        project=project_path,
        name=project_name,

        epochs=epochs,
        imgsz=imgsz,
        batch=32,
        workers=12,
        seed=42,
```

```
        device=0,  # sử dụng GPU

        cos_lr=True,

        mosaic=0.1,
        mixup=0.0,
        copy_paste=0.0,

        optimizer="AdamW",
        lr0=0.001,
        weight_decay=1e-2,
    )
```

Thay vì chọn một cấu hình ngẫu nhiên, áp dụng phương pháp tìm kiếm lưới để xác định kiến trúc tối ưu nhất: - So sánh kiến trúc mạng: Đánh giá hiệu năng giữa YOLO11s (Small - tối ưu tốc độ) và YOLO11m (Medium - cân bằng độ chính xác). - Khảo sát siêu tham số: huấn luyện 16 mô hình với 16 tổ hợp tham số: số vòng lặp (`epochs`); kích thước ảnh (`imgsz`); thuật toán tối ưu (`optimizer`): So sánh giữa `SGD` (truyền thống) và `AdamW` (hiện đại) xem thuật toán nào hội tụ tốt hơn, cho ra kết quả tốt hơn.

[24]:
```
BASE_MODELS = ["yolo11s", "yolo11m"]

epochs_list = [30, 80]
imgsz_list = [448, 640]
optimizer_list = ["SGD", "AdamW"]

# Huấn luyện mô hình với các thông số khác nhau
parameters = list(product(epochs_list, imgsz_list, optimizer_list))
```

Huấn luyện mô hình với các tham số khác nhau dựa trên YOLO11s và YOLO11m.

[25]:
```
for base_model in BASE_MODELS:
    for parameter in parameters:
        epochs, imgsz, optimizer = parameter

        # Lưu model tại "runs/{...}/epochs{...}_imgsz{...}_{...}" sau khi train
        best_path = (
            Path(f"runs/{base_model}")
            / f"epochs{epochs}_imgsz{imgsz}_{optimizer}"
            / "weights"
            / "best.pt"
        )
        model_path = f"runs/{base_model}"
        model_name = f"epochs{epochs}_imgsz{imgsz}_{optimizer}"

        # Huấn luyện mô hình với dataset
        if best_path.exists():
            print(str(best_path))
```

```
            print(f"Model has been trained already. It is being loaded \
                  again...")
            model = YOLO(str(best_path))
        else:
            print(str(best_path))
            print("Model hasn't been trained. Start training...")

            # Huấn luyện mô hình dựa trên mô hình gốc
            model = YOLO(base_model)
            if optimizer == "AdamW":
                train_model_with_AdamW(
                    model=model,
                    epochs=epochs,
                    imgsz=imgsz,
                    dataset_path=DATASET_PATH,
                    project_path=model_path,
                    project_name=model_name
                )
            else:
                train_model_with_SGD(
                    model=model,
                    epochs=epochs,
                    imgsz=imgsz,
                    dataset_path=DATASET_PATH,
                    project_path=model_path,
                    project_name=model_name
                )

            # Load lại best.pt sau khi train, nếu không tìm thấy thì in ra lỗi
            assert best_path.exists(), "File best.pt not found after training"
            model = YOLO(str(best_path))

            print("Training finished")
```

```
runs/yolo11s/epochs30_imgsz448_SGD/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs30_imgsz448_AdamW/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs30_imgsz640_SGD/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs30_imgsz640_AdamW/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs80_imgsz448_SGD/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs80_imgsz448_AdamW/weights/best.pt
Model has been trained already. It is being loaded                  again...
runs/yolo11s/epochs80_imgsz640_SGD/weights/best.pt
Model has been trained already. It is being loaded                  again...
```

```
runs/yolo11s/epochs80_imgsz640_AdamW/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs30_imgsz448_SGD/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs30_imgsz448_AdamW/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs30_imgsz640_SGD/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs30_imgsz640_AdamW/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs80_imgsz448_SGD/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs80_imgsz448_AdamW/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs80_imgsz640_SGD/weights/best.pt
Model has been trained already. It is being loaded                    again…
runs/yolo11m/epochs80_imgsz640_AdamW/weights/best.pt
Model has been trained already. It is being loaded                    again…
```

6 KẾT QUẢ SAU HUẤN LUYỆN

6.1 TỔNG HỢP KẾT QUẢ

Tổng hợp kết quả từ 16 tổ hợp tham số vào bảng dữ liệu thống nhất, dựa trên các chỉ số `mAP50`, `mAP50-95`, `precision` và `recall` để chọn ra bộ trọng số (`best.pt`) tốt nhất.

[26]:
```python
# Hàm ghép dòng vào dataframe
def df_concat(last_results, best_results, base_model, parameter):
    epochs, imgsz, optimizer = parameter

    row = pd.DataFrame([{
        "model": base_model,
        "optimizer": optimizer,
        "epochs": int(epochs),
        "best_epoch": int(best_results["epoch"].iloc[0]),
        "imgsz": int(imgsz),
        "mAP50": float(best_results["metrics/mAP50(B)"].iloc[0]),
        "mAP50-95": float(best_results["metrics/mAP50-95(B)"].iloc[0]),
        "precision": float(best_results["metrics/precision(B)"].iloc[0]),
        "recall": float(best_results["metrics/recall(B)"].iloc[0])
    }])

    last_results = pd.concat([last_results, row], ignore_index=True)

    return last_results
```

[27]:
```
printed_results_columns = [
    "epoch",
    "metrics/mAP50(B)",
    "metrics/mAP50-95(B)",
    "metrics/precision(B)",
    "metrics/recall(B)"
]
df_columns = {
    "model": "string",
    "optimizer": "string",
    "epochs": "int64",
    "best_epoch": "int64",
    "imgsz": "int64",
    "mAP50": "float64",
    "mAP50-95": "float64",
    "precision": "float64",
    "recall": "float64",
}
# last_results là tổng hợp 16 best_results của 16 tham số
last_results = pd.DataFrame({
    columns: pd.Series(dtype=dtype) for columns, dtype in df_columns.items()
})

for base_model in BASE_MODELS:
    for parameter in parameters:
        epochs, imgsz, optimizer = parameter

        results_path = (
            Path(f"runs/{base_model}")
            / f"epochs{epochs}_imgsz{imgsz}_{optimizer}"
            / "results.csv"
        )

        results = pd.read_csv(results_path)
        # best_results là dòng trong results.csv có metrics/mAP50-95 cao nhất cột
        total_training_time = results["time"].iloc[0]
        best_results = (
            results.nlargest(1, "metrics/mAP50-95(B)")[printed_results_columns]
        )
        last_results = df_concat(last_results, best_results, base_model,
                                 parameter)

last_results
```

[27]:

	model	optimizer	epochs	best_epoch	imgsz	mAP50	mAP50-95	\
0	yolo11s	SGD	30	28	448	0.63171	0.52254	

1	yolo11s	AdamW	30	29	448	0.62606	0.51541
2	yolo11s	SGD	30	30	640	0.64837	0.53774
3	yolo11s	AdamW	30	27	640	0.65413	0.53899
4	yolo11s	SGD	80	59	448	0.68338	0.56863
5	yolo11s	AdamW	80	51	448	0.66939	0.55505
6	yolo11s	SGD	80	74	640	0.69285	0.58045
7	yolo11s	AdamW	80	61	640	0.67469	0.56426
8	yolo11m	SGD	30	30	448	0.62506	0.51984
9	yolo11m	AdamW	30	30	448	0.58683	0.48617
10	yolo11m	SGD	30	29	640	0.64561	0.53681
11	yolo11m	AdamW	30	30	640	0.58599	0.48635
12	yolo11m	SGD	80	75	448	0.68433	0.57586
13	yolo11m	AdamW	80	80	448	0.65613	0.55038
14	yolo11m	SGD	80	77	640	0.69895	0.58760
15	yolo11m	AdamW	80	66	640	0.66513	0.55854

	precision	recall
0	0.67112	0.60531
1	0.69223	0.57907
2	0.72044	0.60040
3	0.69437	0.61965
4	0.70953	0.64419
5	0.71050	0.63714
6	0.71261	0.65929
7	0.72661	0.62902
8	0.66062	0.61128
9	0.61359	0.56119
10	0.67240	0.61436
11	0.60895	0.57126
12	0.70530	0.64449
13	0.70220	0.59994
14	0.73820	0.64068
15	0.67832	0.65270

6.2 SO SÁNH

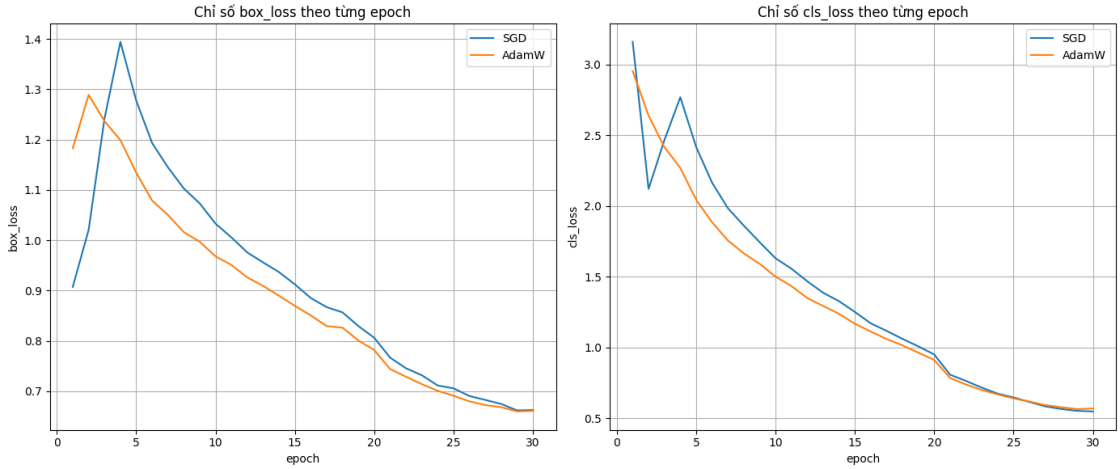
6.2.1 Loss theo epochs

[28]:
```python
def plot_loss_comparison_by_epochs(
    base_model: str,
    epochs: int,
    imgsz: int,
):
    print(f"{base_model} | epochs={epochs} | imgsz={imgsz}")

    results_paths = {
        "SGD": (
```

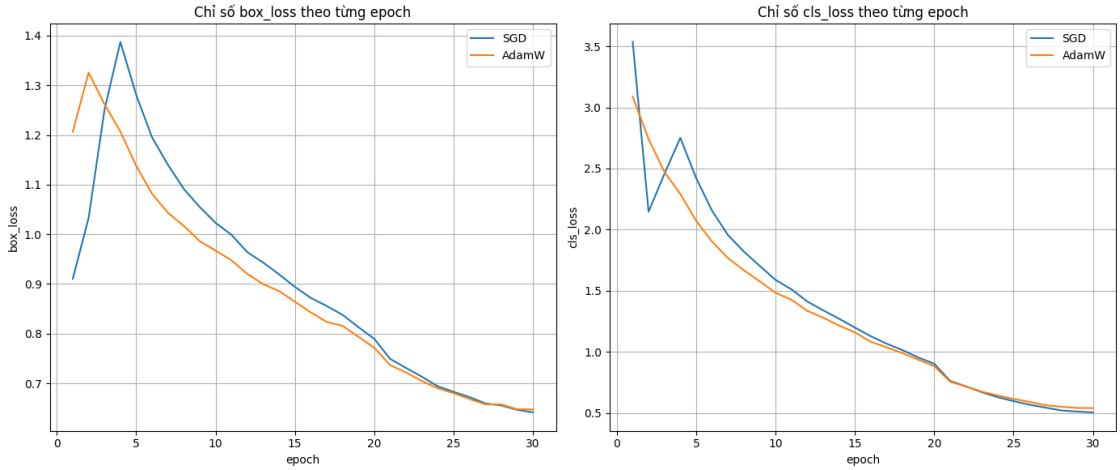
```
            Path(f"runs/{base_model}/epochs{epochs}_imgsz{imgsz}_SGD")
            / "results.csv"
        ),
        "AdamW": (
            Path(f"runs/{base_model}/epochs{epochs}_imgsz{imgsz}_AdamW")
            / "results.csv"
        ),
    }

    fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharex=True)

    loss_types = ["box", "cls"]

    for optimizer, results_path in results_paths.items():
        if not results_path.exists():
            print(f"{results_path} not found")
            continue

        df = pd.read_csv(results_path)

        for ax, loss_type in zip(axes, loss_types):
            loss_column = f"train/{loss_type}_loss"
            ax.plot(
                df["epoch"],
                df[loss_column],
                label=optimizer
            )
            ax.set_title(f"Chỉ số {loss_type}_loss theo từng epoch")
            ax.set_xlabel("epoch")
            ax.set_ylabel(f"{loss_type}_loss")
            ax.grid(True)
            ax.legend()

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

[29]:
```
for base_model in BASE_MODELS:
    for epochs in epochs_list:
        for imgsz in imgsz_list:
            plot_loss_comparison_by_epochs(
                base_model=base_model,
                epochs=epochs,
                imgsz=imgsz
            )
```

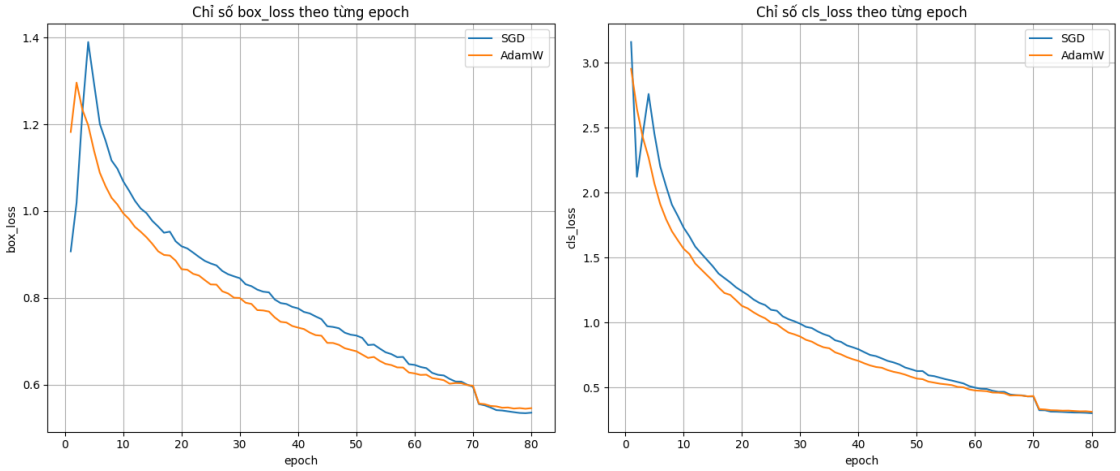
```
yolo11s | epochs=30 | imgsz=448
```



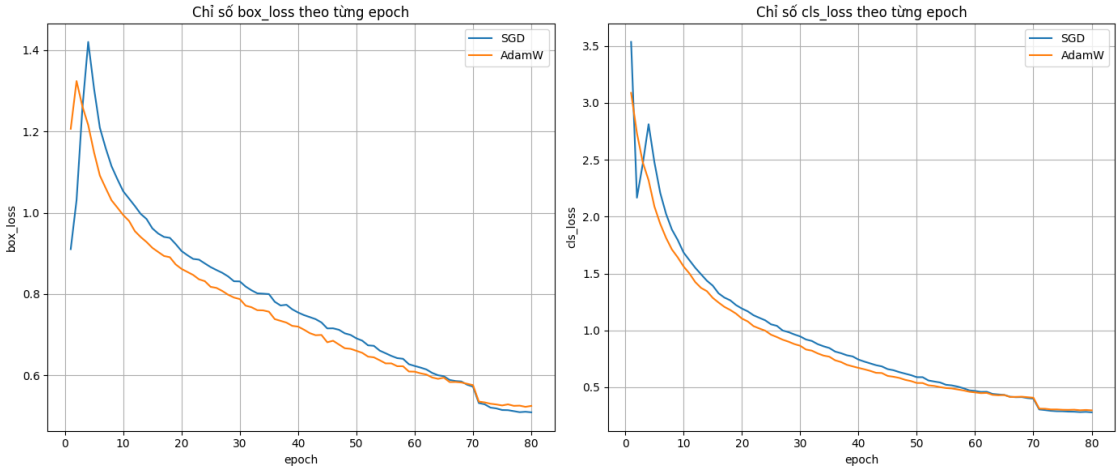
`yolo11s | epochs=30 | imgsz=640`



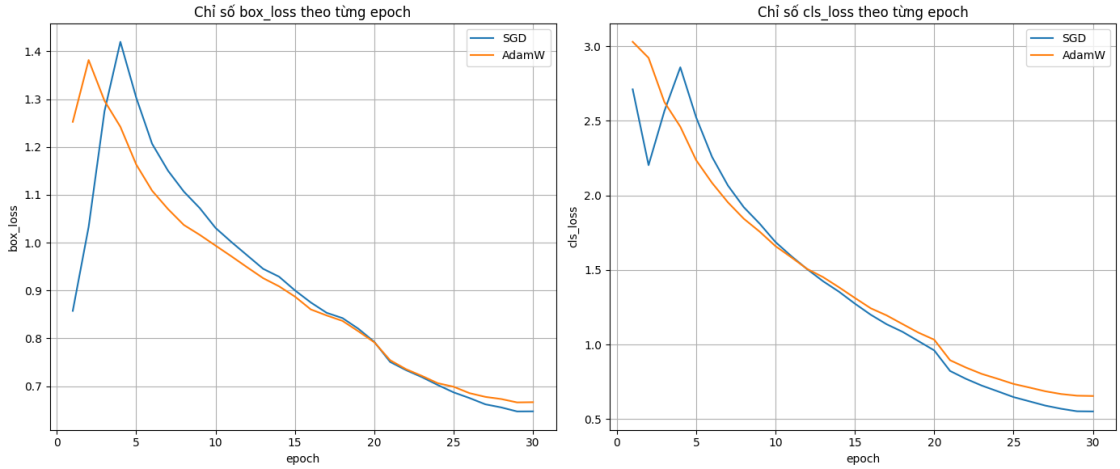
`yolo11s | epochs=80 | imgsz=448`



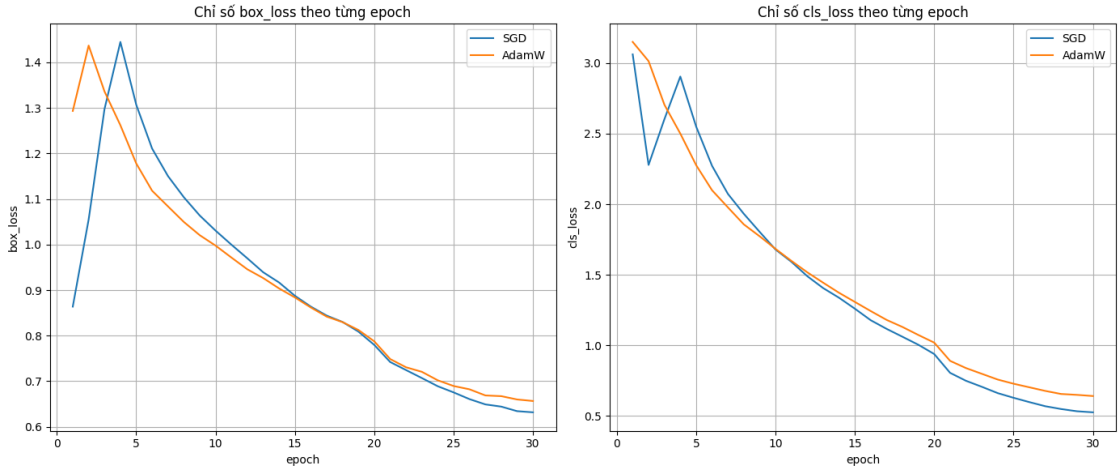
`yolo11s | epochs=80 | imgsz=640`



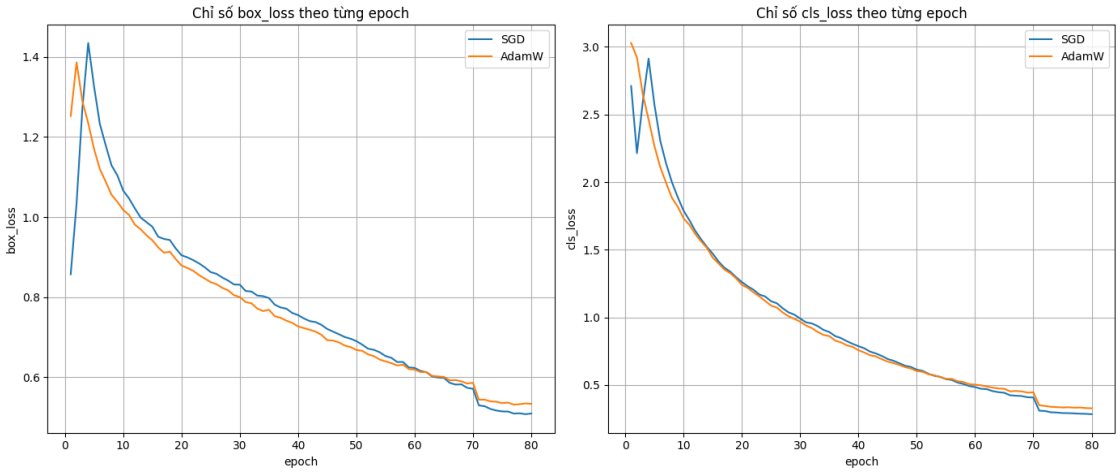
`yolo11m | epochs=30 | imgsz=448`



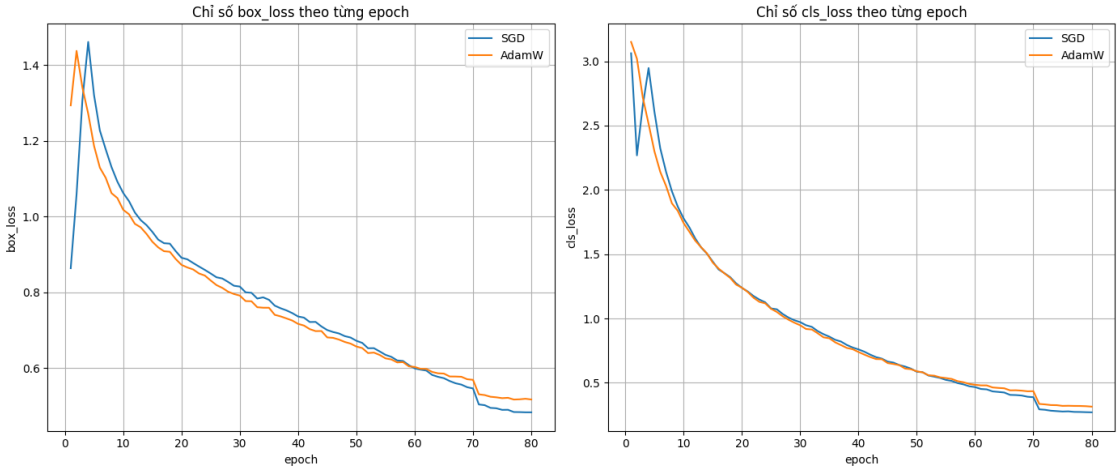
`yolo11m | epochs=30 | imgsz=640`



`yolo11m | epochs=80 | imgsz=448`



`yolo11m | epochs=80 | imgsz=640`



Điểm chung của các biểu đồ ở trên là ở những `epoch` đầu, đường `loss` của optimizer `AdamW` nằm dưới `SGD`, nghĩa là `loss` của `AdamW` thấp hơn `SGD` vào những `epoch` đầu, optimizer `AdamW` hội tụ nhanh hơn so với `SGD`. Tuy nhiên, ở những `epoch` sau, đường `loss` của `SGD` tiếp tục giảm dần và thấp hơn `AdamW` vào những epoch cuối. Điều này có nghĩa là `AdamW` học nhanh hơn `SGD` ở giai đoạn đầu nhưng `SGD` lại giúp quá trình huấn luyện ổn định hơn khi số `epoch` tăng. Vì vậy, khi lựa chọn optimizer, cần cân nhắc giữa tốc độ hội tụ ban đầu và khả năng tối ưu ở các `epoch` cuối, và một số yếu tố khác.

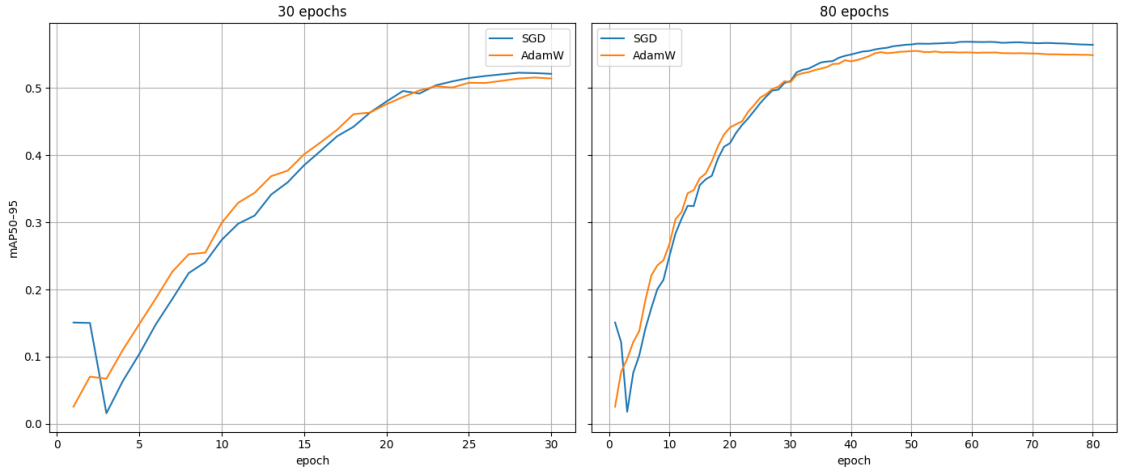
Mặc dù quan trọng, `loss` không phải chỉ số để đánh giá một mô hình có tốt hay không. Chất lượng của mô hình cuối cùng được đánh giá dựa trên các chỉ số `mAP50`, `mAP50-95` trên tập validation.

6.2.2 mAP50-95 theo epochs

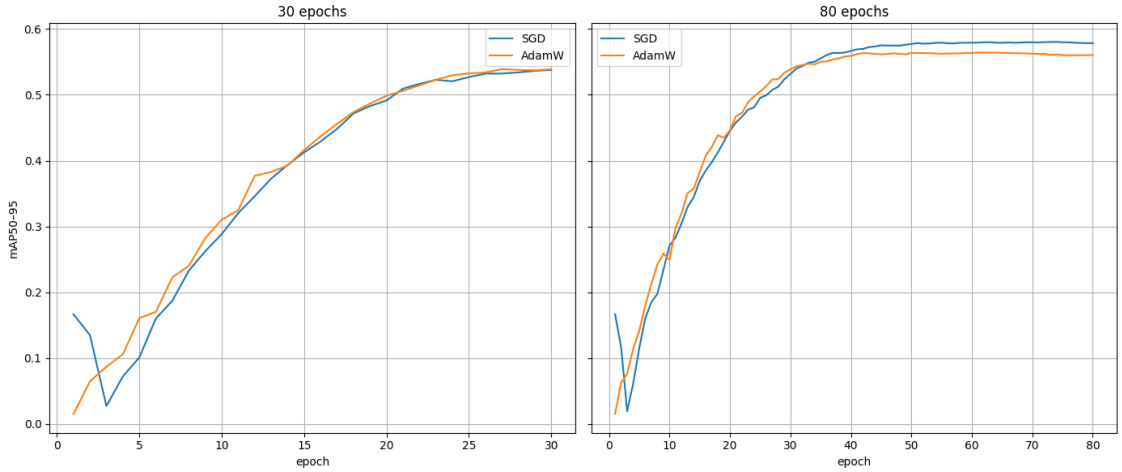
[30]:
```python
def plot_map_comparison_by_epochs(
    base_model: str,
    imgsz: int
):
    print(f"{base_model} | imgsz={imgsz}")

    fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

    for ax, epochs in zip(axes, epochs_list):
        results_paths = {
            "SGD": (
                Path(f"runs/{base_model}/epochs{epochs}_imgsz{imgsz}_SGD")
                / "results.csv"
            ),
            "AdamW": (
                Path(f"runs/{base_model}/epochs{epochs}_imgsz{imgsz}_AdamW")
                / "results.csv"
            ),
        }

        for optimizer, path in results_paths.items():
            if not path.exists():
                print(f"{path} not found")
                continue

            df = pd.read_csv(path)
            ax.plot(
                df["epoch"],
                df["metrics/mAP50-95(B)"],
                label=optimizer
            )

        ax.set_title(f"{epochs} epochs")
        ax.set_xlabel("epoch")
        ax.grid(True)
        ax.legend()

    axes[0].set_ylabel("mAP50-95")

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

[31]:
```python
for base_model in BASE_MODELS:
    for imgsz in imgsz_list:
        plot_map_comparison_by_epochs(base_model, imgsz)
```

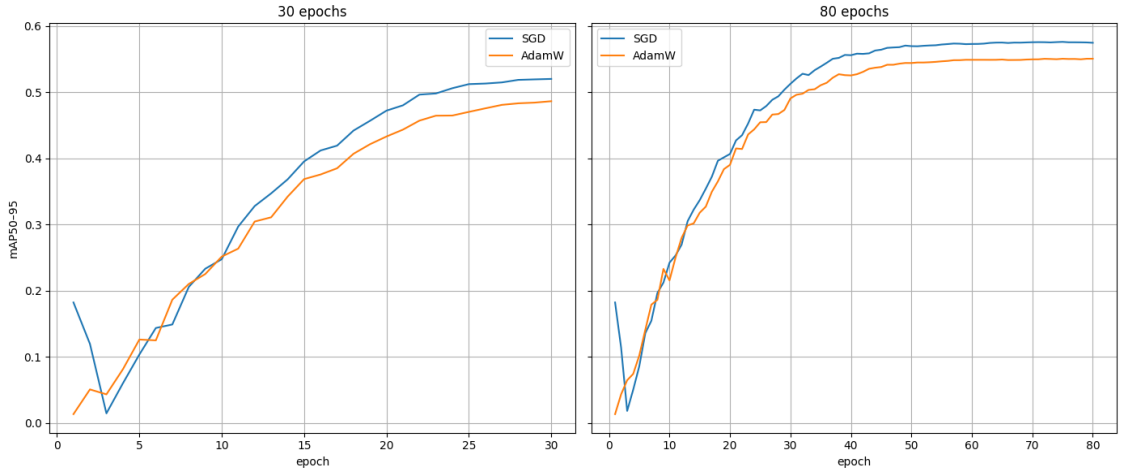
`yolo11s | imgsz=448`



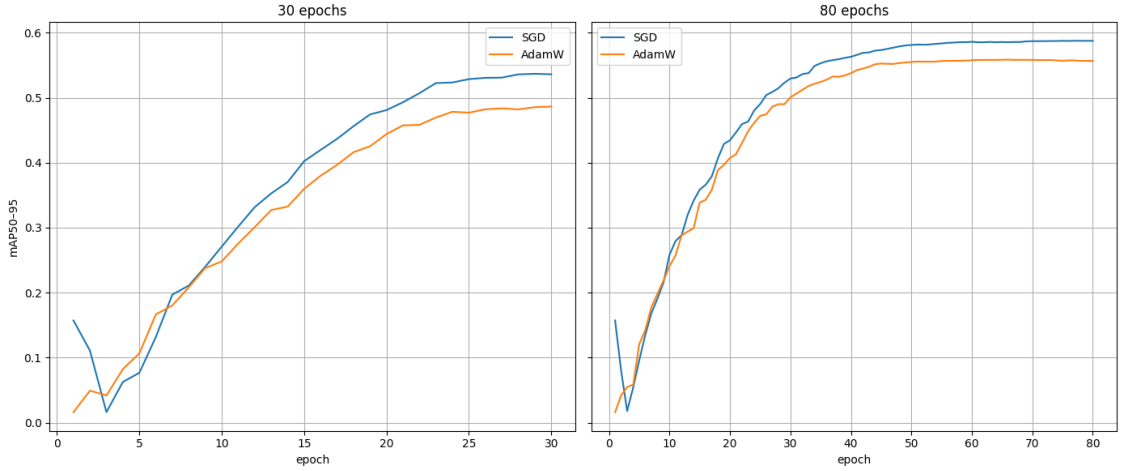
`yolo11s | imgsz=640`



`yolo11m | imgsz=448`



`yolo11m | imgsz=640`



Có thể thấy rằng với cùng một tổ hợp tham số nhưng nếu `epochs` cao hơn (80 so với 30) thì chỉ số `mAP50-95` được cải thiện.

Các biểu đồ cho thấy `AdamW` thường đạt `mAP50-95` cao hơn trong các `epoch` đầu. Tuy nhiên, ở các `epoch` sau, chỉ số `map50-95` của optimizer `SGD` có cải thiện và trong đa số trường hợp đạt `mAP50-95` tương đương hoặc cao hơn `AdamW`.

`AdamW` thường hội tụ nhanh hơn `SGD`, trong khi `SGD` lại thể hiện ưu thế hơn khi huấn luyện mô hình trong thời gian dài, cho phép quá trình tối ưu diễn ra ổn định và hiệu quả hơn.

Ngoài ra, mô hình gốc cũng tác động đến chỉ số này. Ở hai biểu đồ đầu tiên, khi mô hình gốc là YOLO11s, có thể thấy đường chỉ số `mAP50-95` của cả hai optimizer đều gần như trùng nhau hoặc cách nhau một đoạn rất ngắn, tức là chỉ số `mAP50-95` gần bằng nhau; còn với YOLO11m, khi mô hình được huấn luyện đến những `epoch` cuối, chỉ số `mAP50-95` cách nhau một đoạn khá xa, có sự khác biệt rõ rệt hơn so với YOLO11s.

Ở những biểu đồ thể hiện việc mô hình được huấn luyện với 80 `epochs`, dễ dàng nhận thấy rằng bắt đầu từ `epoch` 30 trở đi là `SGD` tỏ ra ưu thế hơn so với `AdamW`, khi đường `mAP50-95` của cả hai optimizer đều cắt nhau trong khoảng `epoch` 10 đến 15.

Như vậy, việc lựa chọn optimizer nào còn tuỳ thuộc vào thời gian và tài nguyên tính toán, bài toán có cần hội tụ nhanh hay không.

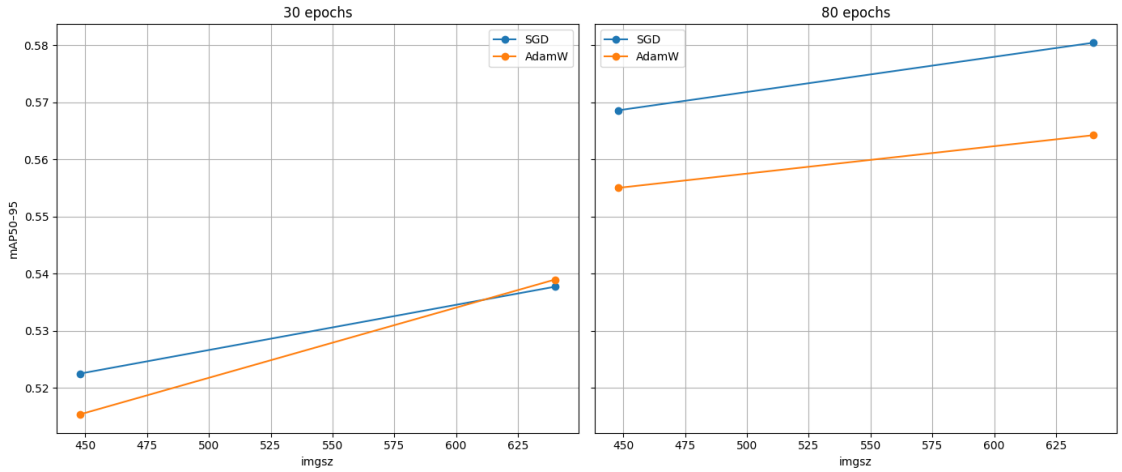
6.2.3 mAP50-95 theo imgsz

[32]:
```python
def plot_map_comparison_by_imgsz(
    base_model: str,
):
    print(f"{base_model}")

    fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

    for ax, epochs in zip(axes, epochs_list):
        for optimizer in ["SGD", "AdamW"]:
            maps = []

            for imgsz in imgsz_list:
                results_path = (
                    Path(f"runs/{base_model}")
                    / f"epochs{epochs}_imgsz{imgsz}_{optimizer}"
                    / "results.csv"
                )

                if not results_path.exists():
                    print(f"{results_path} not found")
                    maps.append(None)
                    continue

                df = pd.read_csv(results_path)
                # Lấy giá trị mAP50-95(B) cao nhất trong cột
                best_map = df["metrics/mAP50-95(B)"].max()
                maps.append(best_map)

            ax.plot(
                imgsz_list,
                maps,
                marker="o",
                label=optimizer
            )

        ax.set_title(f"{epochs} epochs")
        ax.set_xlabel("imgsz")
        ax.grid(True)
        ax.legend()
```

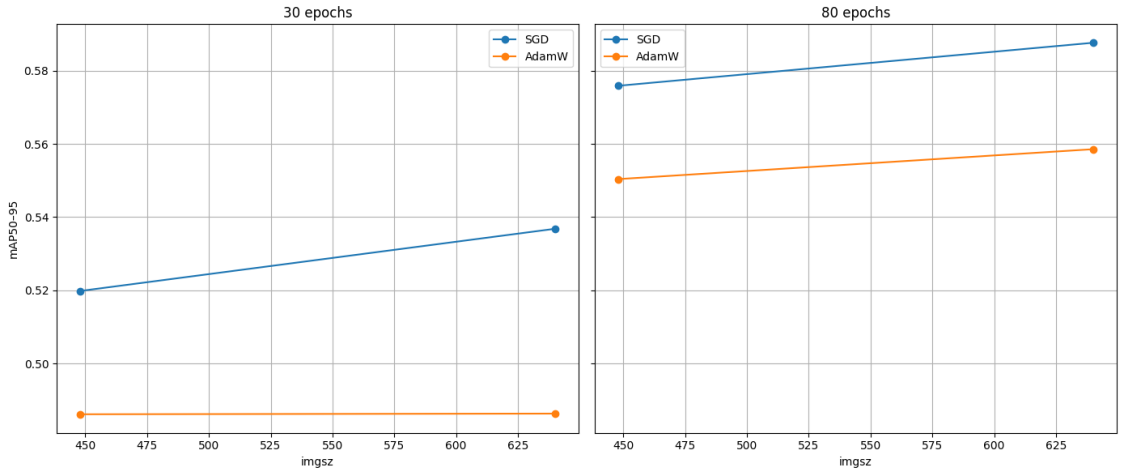
```
    axes[0].set_ylabel("mAP50-95")

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

[33]:
```
for base_model in BASE_MODELS:
    plot_map_comparison_by_imgsz(base_model)
```

yolo11s



yolo11m



Khi so sánh `mAP50-95` theo `imgsz`, cần lấy kết quả `mAP50-95` cao nhất để so sánh.

Khi so sánh cố định `imgsz`, `SGD` cho kết quả vượt trội hơn hẳn so với `AdamW`, trừ các trường hợp nếu

mô hình là YOLO11s và 30 `epochs`. Giải thích cho điều này là vì mô hình YOLO11s ít tham số hơn so với YOLO11m, và `AdamW` hội tụ nhanh hơn `SGD` nên nếu `epochs` thấp thì kết quả `mAP50-95` của `AdamW` có thể cao hơn `SGD`. Nhưng nhìn chung trong đa số trường hợp, `SGD` cho giá trị `mAP50-95` cao hơn.

Khi cố định số `epochs`, việc tăng `imgsz` từ 448 lên 640 giúp cải thiện `mAP50-95` trong hầu hết các trường hợp, đặc biệt rõ rệt khi sử dụng SGD ở 80 epochs. `imgsz` cao cũng có ích trong việc phát hiện và nhận diện object nhỏ. Tuy nhiên, khi tăng `imgsz`, do có chi tiết cao nên cần nhiều thời gian và tài nguyên tính toán hơn so với `imgsz` nhỏ.

7 ƯU VÀ NHƯỢC ĐIỂM CỦA YOLO

7.1 ƯU ĐIỂM

- **Tốc độ xử lý nhanh**: YOLO nổi bật với khả năng xử lý ảnh cực nhanh nhờ thiết kế một bước duy nhất. Thay vì quét từng vùng nhỏ trong ảnh như nhiều phương pháp khác, YOLO xử lý toàn bộ hình ảnh cùng lúc, giúp phát hiện object gần như tức thì. Ưu điểm này phù hợp với các ứng dụng thời gian thực như giám sát an ninh, xe tự hành, drone,...

- **Độ chính xác cao**: Mặc dù ưu tiên tốc độ, các phiên bản mới của YOLO vẫn đạt độ chính xác cao nhờ cải tiến kiến trúc mạng, chiến thuật huấn luyện và cơ chế phát hiện đa tỷ lệ, giúp nhận diện tốt cả các đối tượng nhỏ, mờ hoặc bị che khuất. Tuy vậy, trong một số bài toán yêu cầu độ chính xác cực cao, đặc biệt là với các vật thể rất nhỏ hoặc cần định vị chính xác từng pixel, YOLO có thể chưa vượt trội bằng một số mô hình two-stage hoặc transformer-based, dù sự khác biệt này đã được thu hẹp đáng kể ở các phiên bản gần đây.

- **Khả năng tổng quát tốt**: Khác với các mô hình chia nhỏ ảnh thành nhiều vùng để xử lý, YOLO phân tích toàn bộ hình ảnh một cách tổng thể, giúp mô hình hiểu ngữ cảnh và mối liên hệ giữa các object. Nhờ đó, mô hình có khả năng tổng quát tốt hơn và có thể nhận diện được các objet trong những tình huống chưa từng gặp khi huấn luyện.

- **Mã nguồn mở**: Mô hình YOLO được phát triển và duy trì dưới dạng mã nguồn mở, giúp cộng đồng nghiên cứu và phát triển dễ dàng truy cập, sử dụng và tùy chỉnh, khắc phục nhanh chóng nếu có lỗi hoặc có vấn đề về hiệu năng.

7.2 HẠN CHẾ

- YOLO thường gặp khó khăn trong việc phát hiện các vật thể nhỏ trong ảnh. Điều này là do nó chia ảnh thành một lưới (grid), dự đoán các bounding box và xác suất lớp cho mỗi ô lưới (grid cell), điều này có thể không đủ đối với các vật thể nhỏ trải rộng trên nhiều ô. Ví dụ: một bức ảnh về một đàn chim nhỏ, mô hình có thể bỏ sót một vài con hoặc dự đoán sai vị trí của chúng.

- YOLO sử dụng các feature tương đối thô để prediet bounding box, do model sử dụng nhiều lớp downsampling từ các bức ảnh đầu vào. Bởi các hạn chế này của model khi huấn luyện để predict bounding box từ data, dẫn đến YOLO không thực sự tốt trong việc nhận diện các object với tỉ lệ hình khối mới hoặc bất thường so với tập data. YOLO đã khắc phục phần nào vấn đề này, nhưng vẫn thua kém nhiều so với FRCNN.

- Ngoài ra, trong quá trình training, loss function không có sự đánh giá riêng biệt giữa error của bounding box kích thước nhỏ so với error của bounding box kích thước lớn. Việc coi chúng như cùng loại và tổng hợp lại làm ảnh hưởng đến độ chính xác toàn cục của mạng. Error nhỏ trên box lớn nhìn chung ít tác hại, nhưng error nhỏ với box rất nhỏ sẽ đặc biệt ảnh hưởng đến giá trị IoU. Ở YOLO, hầu hết sai số đến từ việc định vị vật thể không chính xác.

8 ỨNG DỤNG

- **Chăm sóc sức khỏe**: YOLO được ứng dụng để phát hiện nhanh các dấu hiệu bất thường trên hình ảnh y khoa như X-quang, CT hay MRI. Mô hình này hỗ trợ bác sĩ trong việc chẩn đoán chính xác và kịp thời các bệnh lý như ung thư, tổn thương mô hoặc các khối u nhỏ khó nhận biết bằng mắt thường.

- **Robot và tự động hóa**: Nhờ YOLO, robot có thể nhận diện và phân loại các vật thể trong môi trường xung quanh một cách nhanh chóng và chính xác. Nhờ đó, chúng có thể thực hiện các nhiệm vụ như phân loại hàng hóa trong kho, nhận biết chướng ngại vật để di chuyển an toàn, hoặc tương tác hiệu quả với con người.

- **Giám sát an ninh**: YOLO giúp nhận diện và theo dõi người, xe cộ hay vật thể khả nghi qua hệ thống camera. Nhờ khả năng phát hiện thời gian thực, YOLO hỗ trợ cảnh báo sớm các tình huống bất thường, giúp lực lượng an ninh phản ứng kịp thời và nâng cao hiệu quả bảo vệ an toàn cho các khu vực quan trọng.

- *Mô hình xe tự lái**: YOLO đóng vai trò quan trọng trong hệ thống điều khiển của xe tự hành, giúp phát hiện người đi bộ, xe cộ, biển báo giao thông và vật cản trên đường. Nhờ khả năng nhận diện thời gian thực, xe có thể phản ứng kịp thời trước các tình huống bất ngờ, đảm bảo an toàn khi vận hành.

9 KẾT LUẬN

- Nếu dựa vào giá trị `mAP50-95` tốt nhất để xem thử mô hình nào là tốt nhất thì đó chính là YOLO11m với optimizer `SGD`, `epochs=80` và `imgsz=640`. Tuy nhiên, không phải cứ `mAP50-95` cao thì mô hình gọi là tốt, cần phải tối ưu giữa giá trị này và yêu cầu thực tế của vấn đề, cân bằng với thời gian và chi phí tính toán,...

[34]:
```
last_results.nlargest(1, "mAP50-95")
```

[34]:
```
      model optimizer  epochs  best_epoch  imgsz    mAP50  mAP50-95  \
14  yolo11m       SGD      80          77    640  0.69895    0.5876

    precision   recall
14     0.7382  0.64068
```

Trong đề tài này, nhóm đã nghiên cứu và triển khai mô hình YOLO nhằm giải quyết bài toán phân loại hình ảnh giữa con người và động vật. Thông qua việc tìm hiểu cơ sở lý thuyết về học sâu và mô hình YOLO, nhóm đã xây dựng được quy trình xử lý dữ liệu hoàn chỉnh bao gồm thu thập dữ liệu, tiền xử lý ảnh, huấn luyện mô hình và đánh giá kết quả.

Kết quả cho thấy mô hình YOLO có khả năng nhận diện và phân loại đối tượng người và động vật trong ảnh một cách hiệu quả, đồng thời đáp ứng yêu cầu về độ chính xác và tốc độ xử lý .. Việc mô hình vừa phân loại vừa xác định vị trí đối tượng đã minh họa rõ cách thuật toán được sử dụng để giải quyết một bài toán phức tạp từ dữ liệu đầu vào đến kết quả đầu ra.

Thông qua đồ án, nhóm đã củng cố kiến thức nền tảng về học sâu và quy trình phân tích dữ liệu, đặc biệt là các bước làm việc với dữ liệu hình ảnh, huấn luyện mô hình và đánh giá hiệu suất. Quá trình thực hiện đồ án cũng giúp nhóm nâng cao kỹ năng tiếp cận và giải quyết các bài toán dữ liệu mang tính thực tiễn.

Trong định hướng phát triển tiếp theo, đề tài có thể được mở rộng bằng cách cải thiện chất lượng và độ đa dạng của tập dữ liệu, đồng thời thử nghiệm các cấu hình mô hình khác nhau nhằm nâng cao khả năng tổng quát hóa. Bên cạnh đó, việc phân tích sâu hơn các kết quả đầu ra và triển khai hệ thống vào các bài toán phân tích dữ liệu thực tế sẽ góp phần đánh giá rõ hơn hiệu quả và tính ứng dụng của mô hình.