

AI VIET NAM – AI COURSE 2024

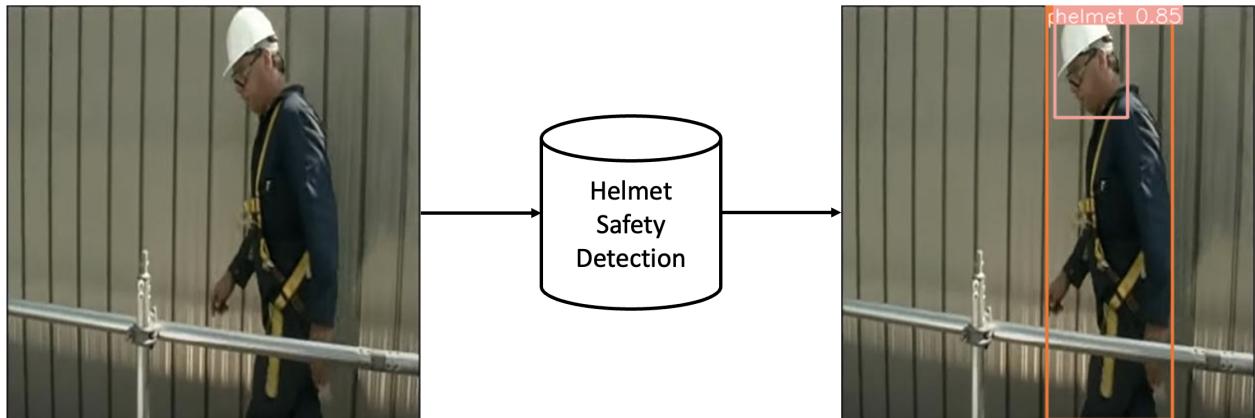
# Kiểm tra tuân thủ đội mũ bảo vệ với YOLOv10

Dinh-Thang Duong, Quang-Vinh Dinh

Ngày 22 tháng 6 năm 2024

## I. Giới thiệu

**Object Detection (Tạm dịch: Phát hiện đối tượng)** là một bài toán cổ điển thuộc lĩnh vực Computer Vision. Mục tiêu của bài toán này là tự động xác định vị trí của các đối tượng trong một tấm ảnh. Đây là một trong những bài toán quan trọng và phức tạp trong Computer Vision, với ứng dụng rộng rãi từ nhận diện khuôn mặt, nhận dạng biển số xe, đến theo dõi đối tượng trong video và tự động lái xe.

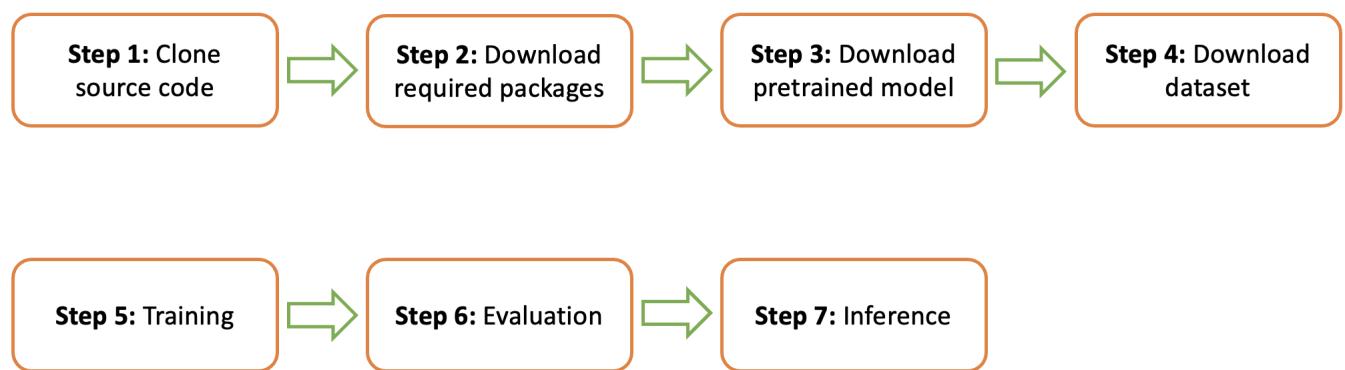


Hình 1: Chương trình phát hiện đối tượng có mang mũ bảo hiểm.

Trong project này, chúng ta sẽ xây dựng một chương trình phát hiện các nhân viên có đeo mũ bảo vệ trong công trường hay không? Mô hình mà chúng ta sử dụng sẽ là mô hình YOLOv10. Theo đó, Input và Output của chương trình là:

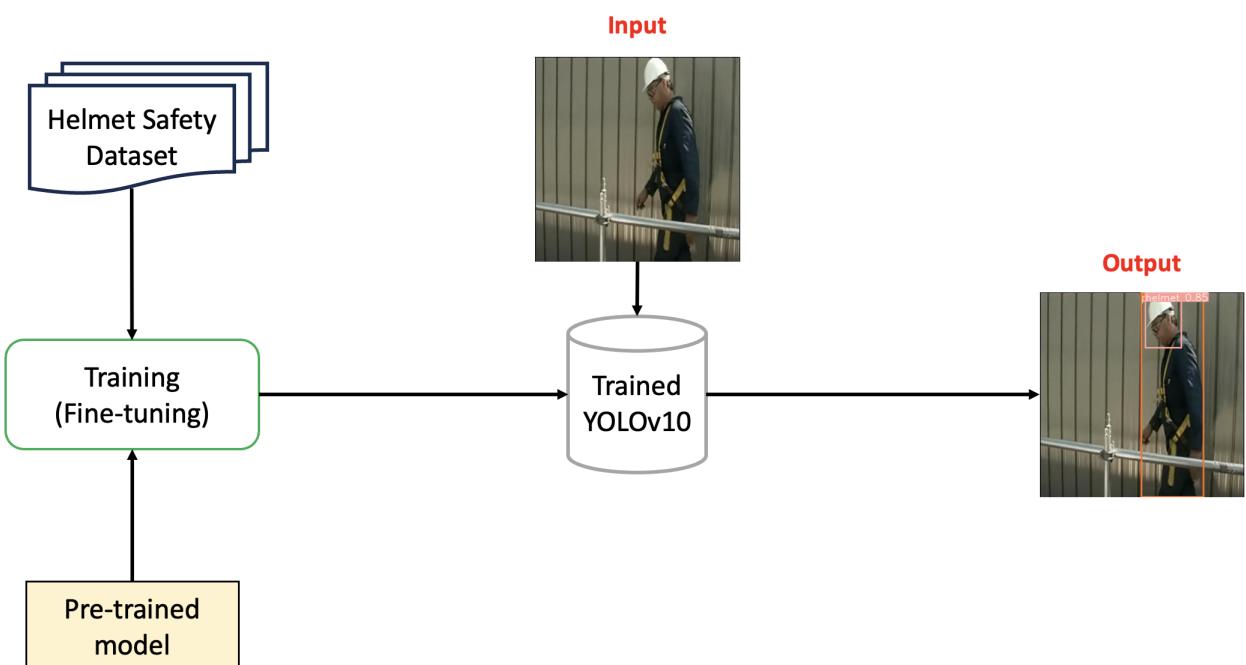
- **Input:** Một tấm ảnh.
- **Output:** Tọa độ (bounding box) của các nhân viên và phần mũ bảo hiểm.

Tổng quan, các bước thực hiện trong project của chúng để hoàn thiện hệ thống Helmet Safety Detection bao gồm:



Hình 2: Tổng quan các bước thực hiện trong project.

Luồng xử lý (pipeline) của chương trình Helmet Safety Detection mà chúng ta sẽ xây dựng có dạng như sau:



Hình 3: Pipeline của chương trình.

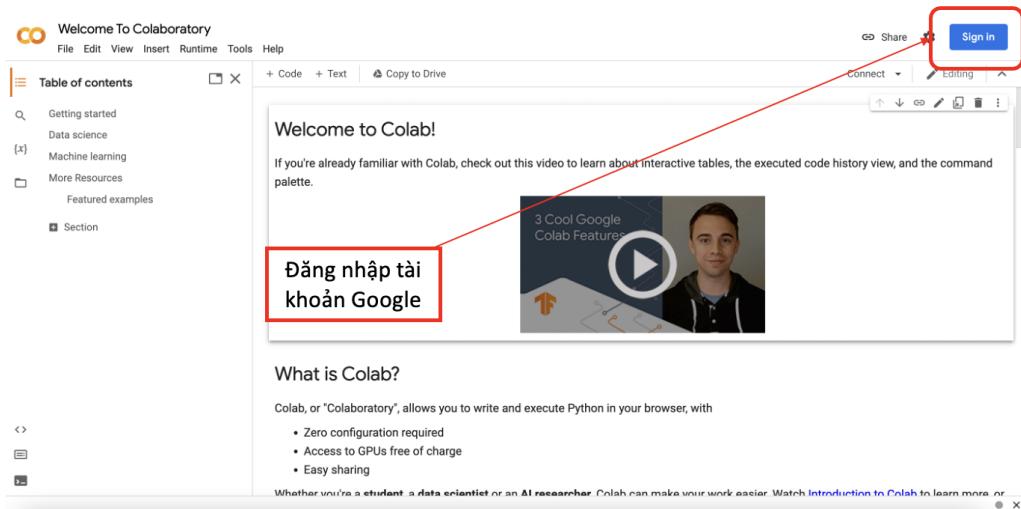
## II. Cài đặt chương trình

Trong phần này, chúng ta sẽ tìm hiểu cách sử dụng YOLOv10 bao gồm việc sử dụng pre-trained model và huấn luyện (fine-tuning) YOLOv10 trên bộ dữ liệu Helmet Safety Detection.

### II.I. Chuẩn bị môi trường lập trình

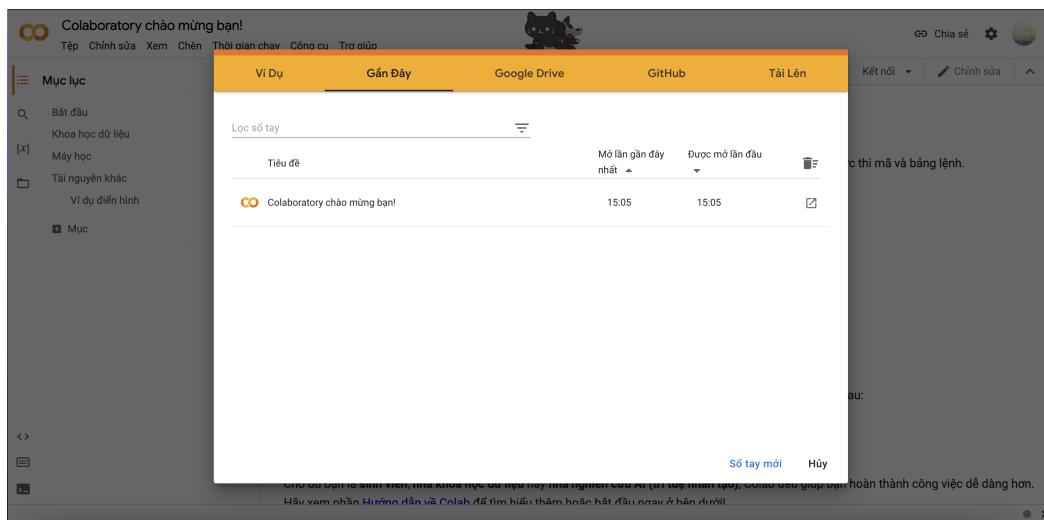
Để thuận tiện trong việc sử dụng YOLOv10, chúng ta sẽ dùng Google Colab làm môi trường cài đặt và thực thi code. Các bước sử dụng Google Colab được thực hiện như sau (**nếu các bạn đã biết cách kích hoạt GPU cho Google Colab thì có thể bỏ qua phần này**):

- **Bước 1:** Truy cập vào đường dẫn sau: [link](#). Nếu truy cập thành công, các bạn sẽ thấy giao diện như hình dưới đây:



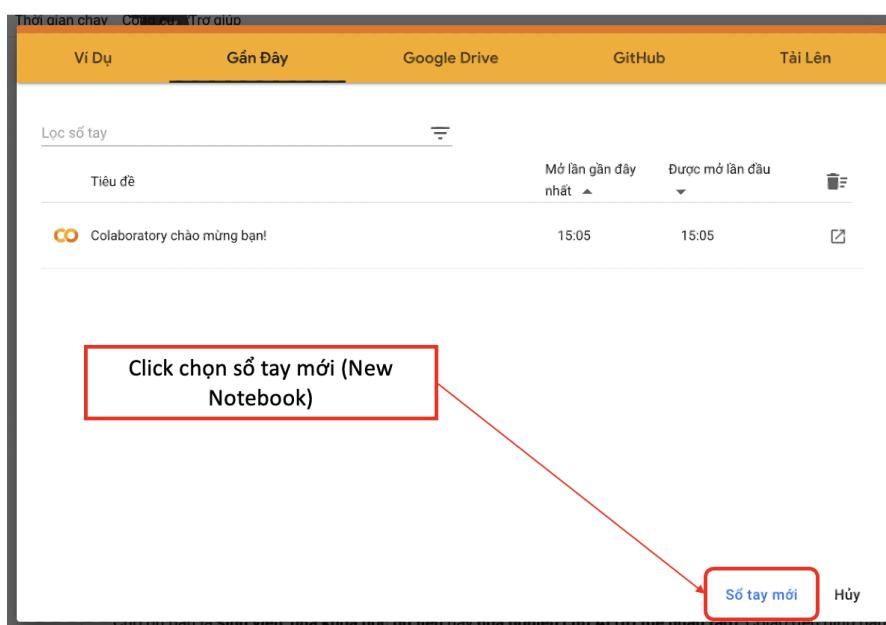
Hình 4: Giao diện chính của Google Colab

Sau đó, các bạn hãy đăng nhập bằng tài khoản Google của mình. Nếu đăng nhập thành công, một cửa sổ mới sẽ hiện lên như hình dưới đây:



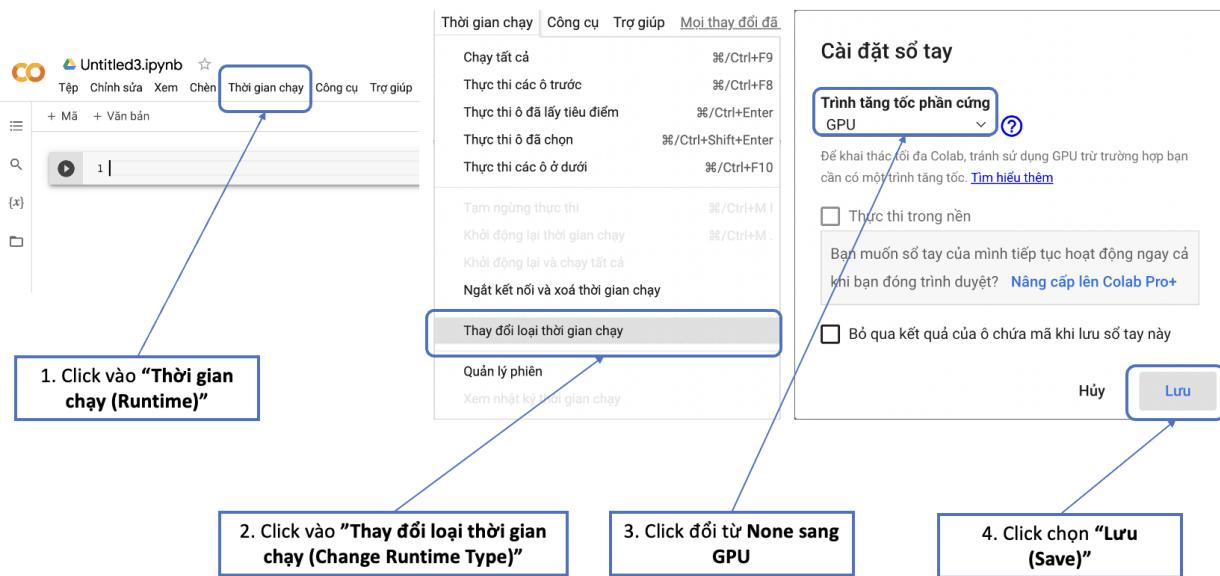
Hình 5: Giao diện Google Colab sau khi đăng nhập thành công

- **Bước 2:** Khởi tạo notebook mới. Notebook sẽ là giao diện để ta có thể viết các dòng lệnh Python. Để tạo được notebook, các bạn thực hiện thao tác theo hình dưới đây:



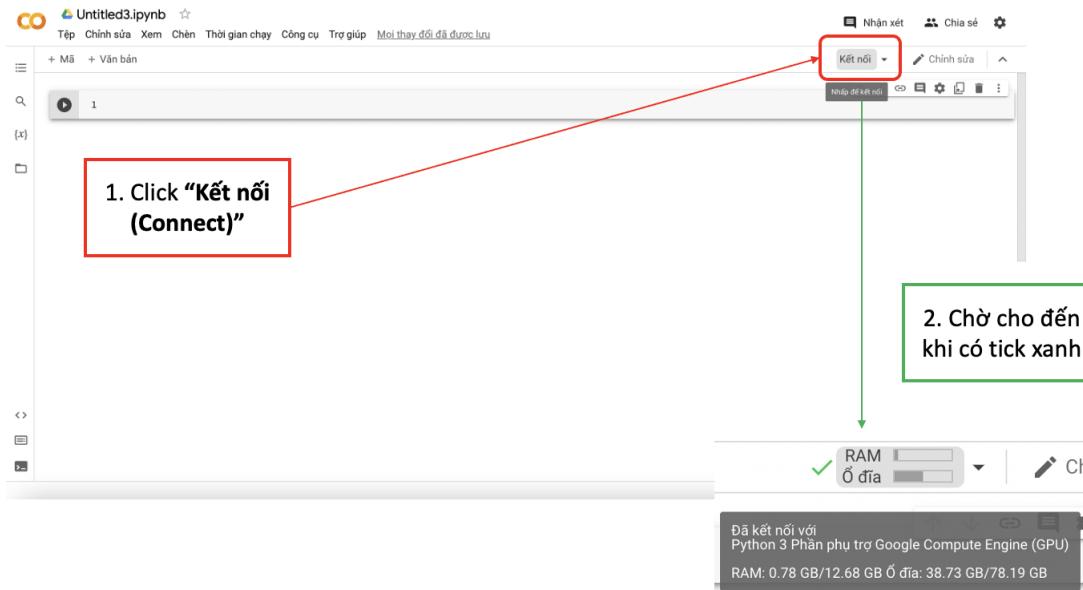
Hình 6: Khởi tạo notebook mới

- **Bước 3:** Thay đổi runtime của notebook từ CPU thành GPU.



Hình 7: Các bước kích hoạt GPU cho notebook mới trên Google Colab

- **Bước 4:** Cuối cùng, để có thể thực thi các dòng lệnh Python, ta cần khởi động notebook. Các thao tác khởi động một notebook trong Google Colab sẽ như hình sau:



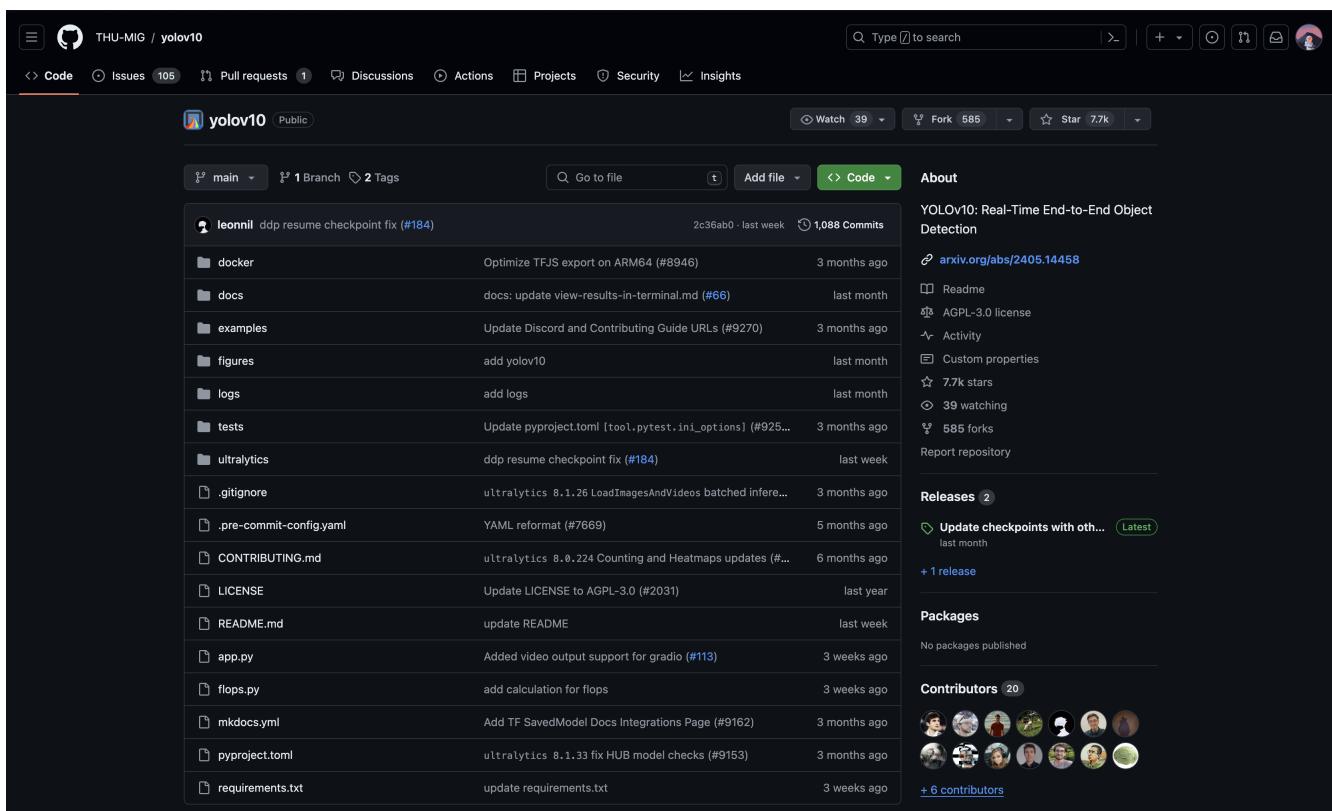
Hình 8: Khởi động một notebook có GPU trong Google Colab

Sau khi thực hiện các bước trên, các bạn đã có một môi trường code Python với GPU miễn phí từ Google.

## II.II. Cài đặt và sử dụng pre-trained model

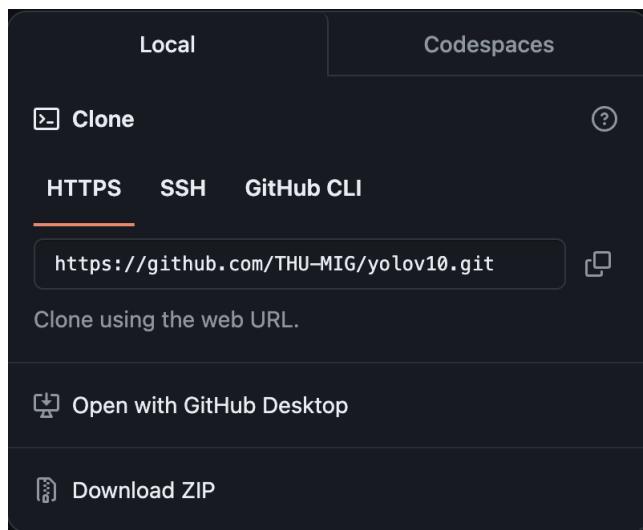
Một cách nhanh chóng để sử dụng được YOLOv10 đó là sử dụng pre-trained model (mô hình đã được huấn luyện sẵn trên bộ dữ liệu COCO - một bộ dữ liệu rất lớn). Để sử dụng pre-trained model, các bạn làm như sau:

1. **Tải mã nguồn YOLOv10 từ GitHub:** Để sử dụng YOLOv10, chúng ta cần tải mã nguồn (source code) của YOLOv10 về môi trường cài đặt code, mã nguồn của YOLOv10 được công khai trên GitHub. Như vậy, các bạn sẽ thực hiện theo các bước sau:
  - **Bước 1:** Các bạn truy cập vào đường dẫn GitHub của YOLOv10 tại [đây](#). Nếu truy cập thành công, các bạn sẽ thấy giao diện như hình dưới:



Hình 9: Giao diện GitHub của YOLOv10

- **Bước 2:** Tại trang GitHub của YOLOv10, các bạn chọn mục **Code** (có màu nền xanh dương như ảnh dưới) và chọn nút copy đường dẫn như ảnh minh họa dưới đây:



Hình 10: Copy đường dẫn để tải mã nguồn YOLOv10

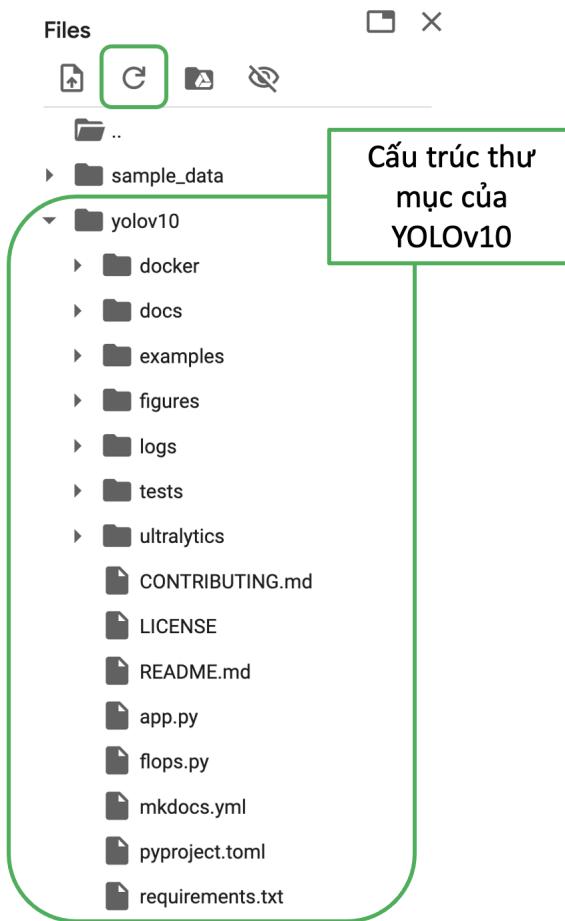
- **Bước 3:** Quay lại Google Colab, các bạn khởi tạo một code cell sử dụng lệnh: `!git clone <link>` (trong đó `<link>` là đường dẫn GitHub đã copy ở **bước 2**). Nếu code cell thực thi thành công, ta kết quả sẽ hiển thị như hình sau:

```
1 !git clone https://github.com/THU-MIG/yolov10.git
```

```
Cloning into 'yolov10'...
remote: Enumerating objects: 20299, done.
remote: Counting objects: 100% (1342/1342), done.
remote: Compressing objects: 100% (122/122), done.
remote: Total 20299 (delta 1299), reused 1220 (delta 1220), pack-reused 18957
Receiving objects: 100% (20299/20299), 11.18 MiB | 20.58 MiB/s, done.
Resolving deltas: 100% (14302/14302), done.
```

Hình 11: Tải mã nguồn YOLOv10 sử dụng lệnh git clone

Để kiểm tra, các bạn có thể refresh lại phần **Files** của Google Colab để xem thư mục YOLOv10 đã xuất hiện hay chưa.



Hình 12: Thư mục YOLOv10

2. **Cài đặt các thư viện cần thiết:** Mã nguồn YOLOv10 được xây dựng bằng rất nhiều các thư viện Python khác nhau. Vì vậy, để có thể chạy được YOLOv10, ta cần tải các gói thư viện cần thiết mà YOLOv10 yêu cầu bằng cách sử dụng file setup có sẵn trong mã nguồn, các bạn có thể làm như sau:

```

1 %cd yolov10
2 !pip install -q -r requirements.txt
3 !pip install -e .

```

3. **Tải trọng số của pre-trained models:** Các bạn tải file pretrained model tại [đây](#) và đặt file đã tải vào thư mục `./yolov10`. Trên Google Colab, việc này có thể được thực hiện thông qua lệnh `wget` như hình dưới đây:

```

1 # Nano version: yolov10n.pt
2 !wget https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10n.pt

--2024-06-21 18:55:52-- https://github.com/THU-MIG/yolov10/releases/download/v1.1/yolov10n.pt
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/804788522/411e0d4f-1023-4
--2024-06-21 18:55:52-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/804788522/
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11448431 (11M) [application/octet-stream]
Saving to: 'yolov10n.pt'

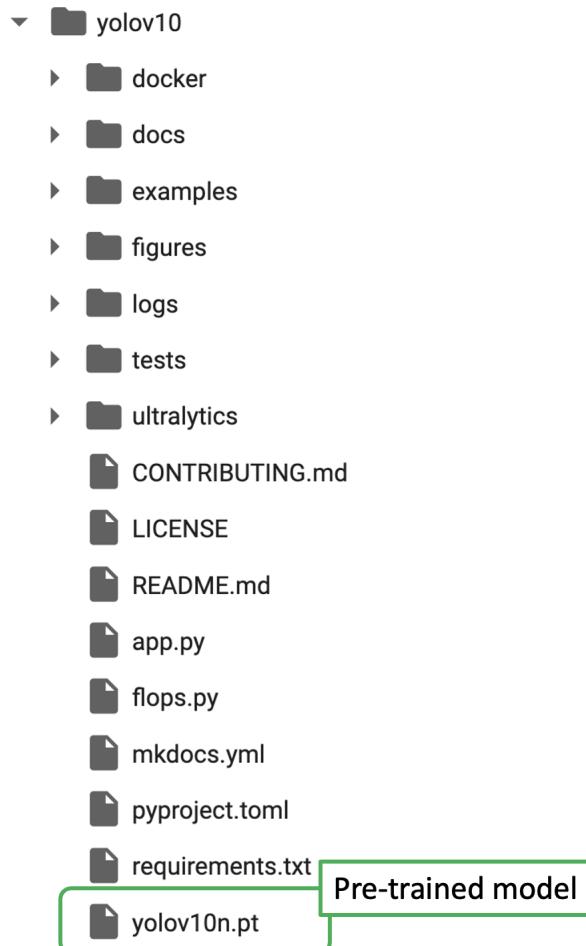
yolov10n.pt      100%[=====] 10.92M  --.-KB/s    in 0.04s

2024-06-21 18:55:52 (275 MB/s) - 'yolov10n.pt' saved [11448431/11448431]

```

Hình 13: Tải pretrained model sử dụng lệnh wget

Để kiểm tra xem file pretrained model đã được tải về thành công hay chưa, các bạn refresh phần **Files** của Google Colab và tìm kiếm file có tên **YOLOv10n.pt**:



Hình 14: File pretrained model

4. **Khởi tạo mô hình:** Để khởi tạo mô hình với trọng số vừa tải về, các bạn chạy đoạn code sau:

```
1 from ultralytics import YOLOv10  
2  
3 MODEL_PATH = 'yolov10n.pt'  
4 model = YOLOv10(MODEL_PATH)
```

5. **Tải ảnh cần dự đoán:** Chúng ta sẽ test mô hình trên một ảnh bất kì. Các bạn có thể tự chọn ảnh của riêng mình hoặc sử dụng ảnh tại [đây](#). Các bạn có thể chạy đoạn code sau để tải ảnh này vào colab tự động:

```
1 !gdown '1tr9PSRRdlC2pNir7jsYugpSMG-7v32VJ' -O './images/'
```

6. **Dự đoán:** Để chạy dự đoán cho ảnh đã tải về, các bạn truyền đường dẫn ảnh vào mô hình như đoạn code sau:

```
1 IMG_PATH = './images/HCMC_Street.jpg'  
2 result = model(source=IMG_PATH)[0]
```



Hình 15: Ảnh cần dự đoán.

7. **Lưu kết quả dự đoán:** Để lưu lại ảnh đã được dự đoán, các bạn chạy đoạn code sau:

```
1 result.save('./images/HCMC_Street_predict.png')
```



Hình 16: Kết quả dự đoán của mô hình YOLOv10 phiên bản **nano** (**yolov10n.pt**).

8. **Dự đoán youtube video:** Bên cạnh source về ảnh, các bạn cũng có thể input source với các tham số khác nhau, đại diện cho các dữ liệu đầu vào khác nhau:

Source	Argument	Type	Notes
image	'image.jpg'	str or Path	Single image file.
URL	'https://ultralytics.com/images/bus.jpg'	str	URL to an image.
screenshot	'screen'	str	Capture a screenshot.
PIL	Image.open('im.jpg')	PIL.Image	HWC format with RGB channels.
OpenCV	cv2.imread('im.jpg')	np.ndarray	HWC format with BGR channels uint8 (0-255).
numpy	np.zeros((640, 1280, 3))	np.ndarray	HWC format with BGR channels uint8 (0-255).
torch	torch.zeros(16, 3, 320, 640)	torch.Tensor	BCHW format with RGB channels float32 (0.0-1.0).
CSV	'sources.csv'	str or Path	CSV file containing paths to images, videos, or directories.
video	'video.mp4'	str or Path	Video file in formats like MP4, AVI, etc.
directory	'path/'	str or Path	Path to a directory containing images or videos.
glob	'path/*.jpg'	str	Glob pattern to match multiple files. Use the * character as a wildcard.
YouTube	'https://youtu.be/LNw0DJXcvt4'	str	URL to a YouTube video.
stream	'rtsp://example.com/media.mp4'	str	URL for streaming protocols such as RTSP, RTMP, TCP, or an IP address.
multi-stream	'list.streams'	str or Path	*.streams text file with one stream URL per row, i.e. 8 streams will run at batch-size 8.

Hình 17: Tổng hợp các kiểu dữ liệu đầu vào YOLOv10 hỗ trợ trong việc thực hiện predict.

Ví dụ, để dự đoán với input là youtube video, các bạn chỉ cần thay thế **IMG\_PATH**

bằng đường dẫn youtube video như đoạn code sau:

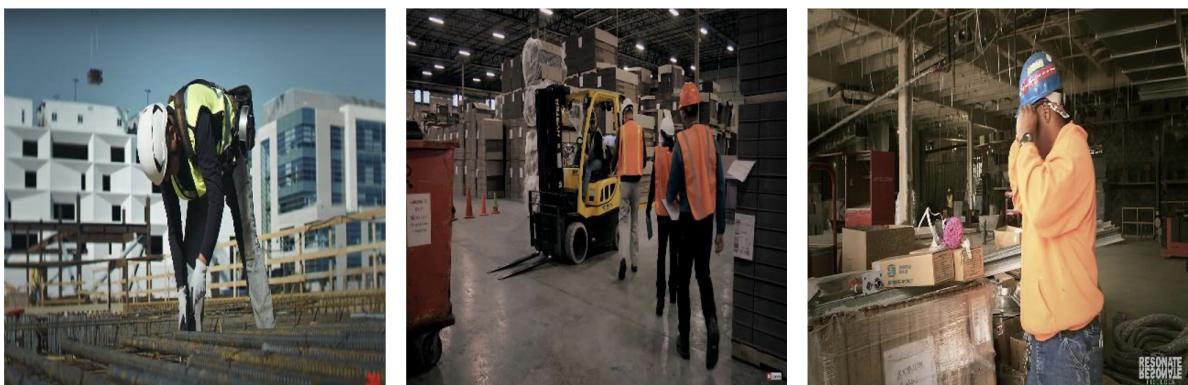
```
1 YOUTUBE_VIDEO_PATH = 'https://youtu.be/wqPSsu7XQ74'
2 video_result = model(source=YOUTUBE_VIDEO_PATH)
```

Kết quả dự đoán sẽ là một video được lưu dưới dạng .avi trong thư mục: /content/yolov10/runs/detect/predict

## II.III. Huấn luyện YOLOv10 trên tập dữ liệu mới

Trong phần này, chúng ta sẽ thực hiện huấn luyện mô hình YOLOv10 (fine-tuning) trên bộ dữ liệu Helmet Safety Detection. Để tránh sự nhầm lẫn, phần này sẽ được thực hiện ở một file colab khác so với phần trước. Các bước thực hiện như sau:

1. **Tải bộ dữ liệu:** Chúng ta sẽ giải quyết bài toán phát hiện các công nhân. Bộ dữ liệu được sử dụng trong bài toán này là Helmet Safety. Để dễ hình dung, các bạn có thể quan sát ảnh minh họa sau:



Hình 18: Một vài mẫu dữ liệu trong bộ dữ liệu về Helmet Safety Detection.

Để tải bộ dữ liệu trên, các bạn hãy chạy đoạn code sau (link tải bộ dữ liệu tại [đây](#)):

```
1 !gdown '1twdtZEfcw4ghSZIiPDypJurZnNXzM07R'
```

Giải nén bộ dữ liệu vào folder **datasets**. Các bạn thực thi đoạn code sau:

```
1 !gdown '1twdtZEfcw4ghSZIiPDypJurZnNXzM07R'
2 !mkdir safety_helmet_dataset
3 !unzip -q '/content/Safety_Helmet_Dataset.zip' -d '/content/safety_helmet_dataset'
```



Hình 19: Thư mục chứa bộ dữ liệu

Quan sát thư mục, có thể thấy bộ dữ liệu này đã được gán nhãn và đưa vào format cấu trúc dữ liệu training theo yêu cầu của YOLO. Vì vậy, chúng ta sẽ không cần thực hiện bước chuẩn bị dữ liệu ở bài này.

- Cài đặt và import các thư viện cần thiết:** Tương tự như phần trước, các bạn chạy các đoạn code sau để cài đặt các gói thư viện để sử dụng được YOLOv10:

```

1 !git clone https://github.com/THU-MIG/yolov10.git
2 %cd yolov10
3 !pip install -q -r requirements.txt
4 !pip install -e .
  
```

- Khởi tạo mô hình YOLOv10:** Chúng ta sẽ khởi tạo mô hình YOLOv10 với phiên bản **nano (n)** từ trọng số đã được huấn luyện trên bộ dữ liệu COCO. Để tải trọng số yolov10n.pt, các bạn chạy đoạn code sau:

```

1 !wget https://github.com/THU-MIG/yolov10/releases/download/v1.1/
yolov10n.pt
  
```

Sau đó, để khởi tạo mô hình từ trọng số đã tải về, các bạn chạy đoạn code sau:

```

1 from ultralytics import YOLOv10
2
3 MODEL_PATH = 'yolov10n.pt'
4 model = YOLOv10(MODEL_PATH)

```

- 4. Huấn luyện mô hình:** Chúng ta tiến hành huấn luyện YOLOv10 trên bộ dữ liệu Helmet Safety Detection với 50 epochs và kích thước ảnh là 640. Các bạn chạy đoạn code sau:

```

1 YAML_PATH = '../safety_helmet_dataset/data.yaml'
2 EPOCHS = 50
3 IMG_SIZE = 640
4 BATCH_SIZE = 256
5
6 model.train(data=YAML_PATH,
7             epochs=EPOCHS,
8             batch=BATCH_SIZE,
9             imgsz=IMG_SIZE)

```

```

Plotting labels to /content/yolov10/runs/detect/train/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: AdamW(lr=0.001429, momentum=0.9) with parameter groups 95 weight(decay=0.0), 108 weight(decay=0.0005), 107 bias(decay=0.0)
TensorBoard: model graph visualization added ✅
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to /content/yolov10/runs/detect/train
Starting training for 50 epochs...

```

Epoch	GPU_mem	box_om	cls_om	dfl_om	box_oo	cls_oo	dfl_oo	Instances	Size
1/50	3.19G	1.653	2.542	1.638	1.462	4.566	1.481	25	640: 100% [██████] 48/48 [00:27<00:00, 1.75it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:	100%	all
2/50	3.24G	1.698	1.891	1.627	1.544	3.881	1.495	50	640: 100% [██████] 48/48 [00:19<00:00, 2.51it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:	100%	all
3/50	3.24G	1.701	1.784	1.606	1.594	3.358	1.496	53	640: 100% [██████] 48/48 [00:18<00:00, 2.60it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:	100%	all
4/50	3.24G	1.693	1.779	1.621	1.627	3.013	1.528	31	640: 100% [██████] 48/48 [00:17<00:00, 2.73it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:	100%	all
5/50	3.27G	1.724	1.729	1.657	1.65	2.727	1.564	35	640: 100% [██████] 48/48 [00:16<00:00, 2.86it/s]
	Class	Images	Instances	Box(P)	R	mAP50	mAP50-95:	100%	all

Hình 20: Quá trình huấn luyện mô hình YOLOv10.

- 5. Đánh giá mô hình:** Để thực hiện đánh giá mô hình trên tập test, các bạn chạy đoạn code sau:

```

1 TRAINED_MODEL_PATH = 'runs/detect/train/weights/best.pt'
2 model = YOLOv10(TRAINED_MODEL_PATH)
3
4 model.val(data=YAML_PATH,
5             imgsz=IMG_SIZE,
6             split='test')

```

```

Ultralytics YOLOv8.1.34 🚀 Python-3.10.12 torch-2.0.1+cu117 CUDA:0 (Tesla T4, 15102MiB)
YOLOv10n summary (fused): 285 layers, 2695586 parameters, 0 gradients, 8.2 GFLOPs
val: Scanning /content/safety_helmet_dataset/test/labels... 109 images, 0 backgrounds, 0

/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was calle
    self.pid = os.fork()
      Class      Images   Instances     Box(P       R       mAP50     mAP50-95):
        all       109       320       0.768     0.738     0.816     0.417
        head      109        16       0.736     0.625     0.775     0.336
        helmet    109       162       0.818     0.864     0.893     0.465
        person    109       142       0.75      0.725     0.78      0.451
Speed: 5.1ms preprocess, 12.7ms inference, 0.0ms loss, 1.0ms postprocess per image
Results saved to /content/yolov10/runs/detect/val
ultralytics.utils.metrics.DetMetrics object with attributes:

```

Hình 21: Đánh giá mô hình sau khi huấn luyện trên tập test.

6. **OPTIONAL:** Phần này sẽ bàn luận thêm một vài vấn đề trong YOLOv10 bao gồm một vài các tham số trong lệnh training, lệnh đánh giá mô hình và gán nhãn dữ liệu.

- **Các tham số trong lệnh training:** Dòng lệnh training tại bước 4 có mặc định sẵn một vài tham số, các tham số này các bạn có thể tùy chỉnh theo ý muốn của mình, với một số tham số có giá trị khác nhau sẽ cho ra hiệu suất mô hình khác nhau. Sau đây là ý nghĩa cơ bản của một vài tham số trên:
  - **img:** Kích thước ảnh training, các ảnh train và test sẽ được resize lại về kích thước bạn gán, **mặc định là 640**. Các bạn hoàn toàn có thể thử nghiệm trên các kích thước ảnh khác nhau.
  - **batch:** Khi thực hiện tính toán trong quá trình training, các mô hình có thể đọc **một** lúc toàn bộ dữ liệu train **hoặc chia ra đọc theo từng batch**. Với mặc định là 64, bộ dữ liệu train sẽ được chia ra thành các batch có 64 mẫu dữ liệu. Các bạn có thể cài đặt các giá trị khác theo  $2^n (n \geq 0)$ .
  - **epochs:** Số lần lặp qua bộ dữ liệu trong quá trình huấn luyện.
  - **data:** Thông tin bộ dữ liệu (**file .yaml**) mà bạn mong muốn training.
  - **weights:** File pretrained model sử dụng. Các bạn có thể tải và sử dụng các file pretrained model khác nhau trong danh sách [này](#).

- **Gán nhãn dữ liệu:** Vì YOLOv10 là học có giám sát (supervised learning), tức các mẫu trong bộ dữ liệu training cần phải có labels tương ứng với từng mẫu. Vì vậy, để có thể cung cấp thêm dữ liệu hoặc tạo ra một bộ dữ liệu mới, với các class mới. Ta cần phải thực hiện gán nhãn dữ liệu, việc gán nhãn này sẽ phải thực hiện thủ công. Trong bài hướng dẫn này, chúng ta sẽ tìm hiểu cách sử dụng [labelImg](#). Để cài đặt labelImg

về máy tính, có rất nhiều cách khác nhau (các bạn có thể đọc file README.md trên trang GitHub của labelImg), ở đây chúng ta sẽ chọn cách cài đặt với Anaconda:

- **Bước 1:** Tải mã nguồn của labelImg tại trang GitHub sử dụng dòng lệnh sau (các dòng lệnh trong các bước ở đây sẽ được thực hiện trong cmd/terminal):

```
1 $ git clone https://github.com/heartexlabs/labelImg.git
```

- **Bước 2:** Cài đặt Anaconda tại [đây](#).
- **Bước 3:** Sau khi đã cài đặt Anaconda, các bạn sẽ khởi tạo môi trường Anaconda bằng dòng lệnh sau:

```
1 $ conda create -n labelimg_env python=3.9 -y
```

- **Bước 4:** Kích hoạt môi trường ảo đã tạo ở bước 3 sử dụng lệnh:

```
1 $ conda activate labelimg_env
```

- **Bước 5:** Di chuyển vào thư mục mã nguồn của labelImg sử dụng lệnh sau:

```
1 $ cd <path_to_labelImg_folder>
```

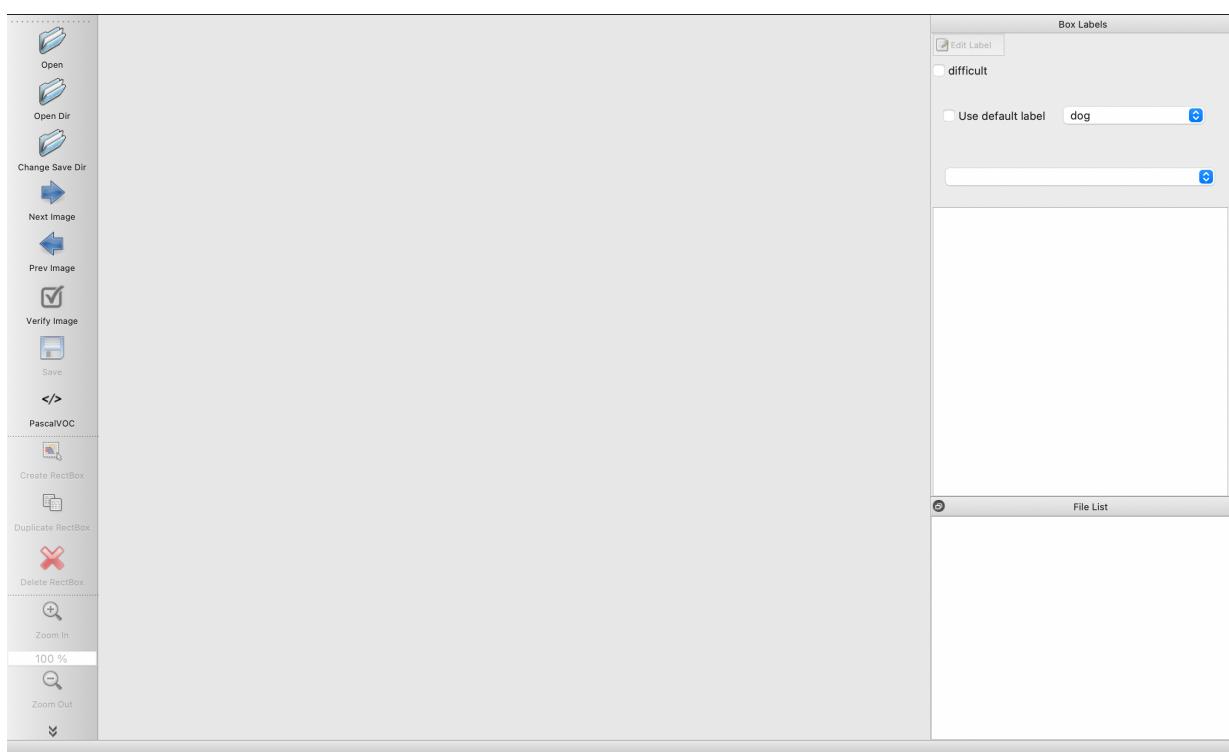
- **Bước 6:** Thực hiện lần lượt các dòng lệnh:

```
1 $ conda install pyqt=5
2 $ conda install -c anaconda lxml
3 $ pyrcc5 -o libs/resources.py resources.qrc
```

- **Bước 7:** Khởi động giao diện của labelImg sử dụng lệnh:

```
1 $ python labelImg.py
```

Nếu thành công, các bạn sẽ thấy một giao diện màn hình như sau:



Về sau, mỗi lần cần sử dụng labelImg, ta chỉ cần kích hoạt môi trường đã tạo này và khởi động labelImg (tức chỉ thực hiện bước 4 và bước 7).

Bây giờ, giả sử ta muốn gán nhãn một tập ảnh người (class person). Chúng ta sẽ thực hiện các bước như sau (chuẩn bị sẵn một thư mục hình ảnh):

- **Bước 1:** Truy cập vào thư mục data trong mã nguồn labelImg, sửa lại nội dung file **predefined\_classes.txt** bằng tên của các class trong bộ dữ liệu của bạn, mỗi tên class sẽ là một hàng trong file:

```

dog
person
cat
tv
car
meatballs
marinara sauce
tomato soup
chicken noodle soup
french onion soup
chicken breast
ribs
pulled pork
hamburger
cavity

```

Hình 22: Nội dung trong file **predefined\_classes.txt**. Mỗi hàng trong file tương ứng với mỗi tên class trong bộ dữ liệu.

Các bạn sẽ sửa lại nội dung file bằng tên của class trong bộ dữ liệu của mình, giả sử chỉ có class là person, vậy ta chỉ sửa lại file thành như sau:

```

person

```

Hình 23: Nội dung trong file **predefined\_classes.txt** sau khi cập nhật.

Đối với bộ dữ liệu Helmet Safety Dataset, ta sẽ có 3 class gồm ('head', 'helmet', 'person'), các bạn có thể coi ở trong file data.yaml ở thư mục bộ dữ liệu:

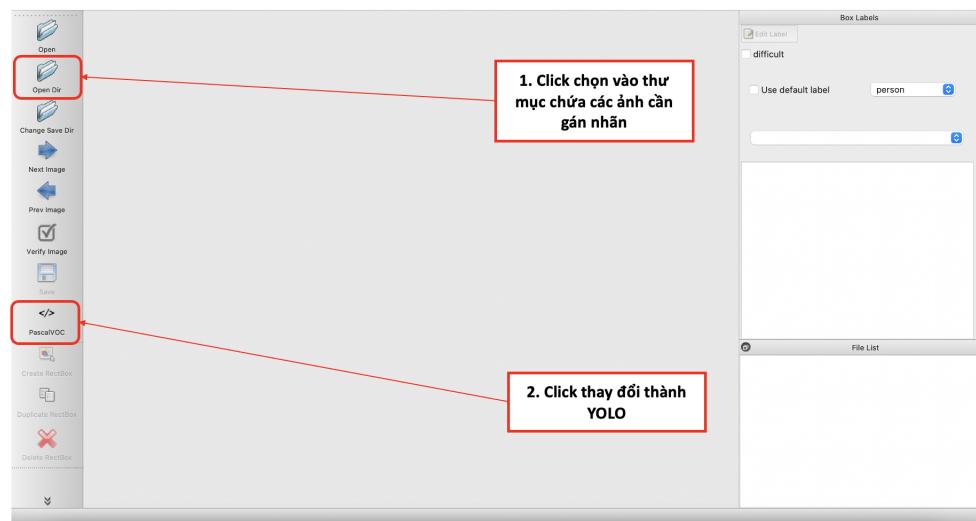
```

data.yaml ×
1 train: ../train/images
2 val: ../valid/images
3 test: ../test/images
4
5 nc: 3
6 names: ['head', 'helmet', 'person']

```

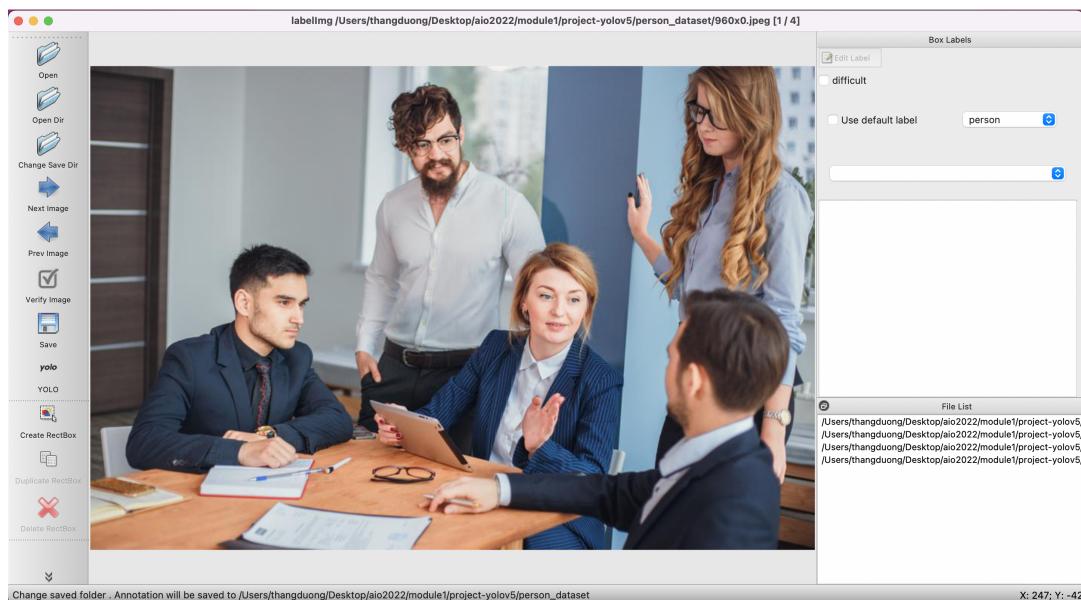
Hình 24: Nội dung file .yaml trong data Helmet Safety Detection.

- **Bước 2:** Truy cập vào giao diện labelImg, các bạn hãy làm theo các bước theo hình sau:



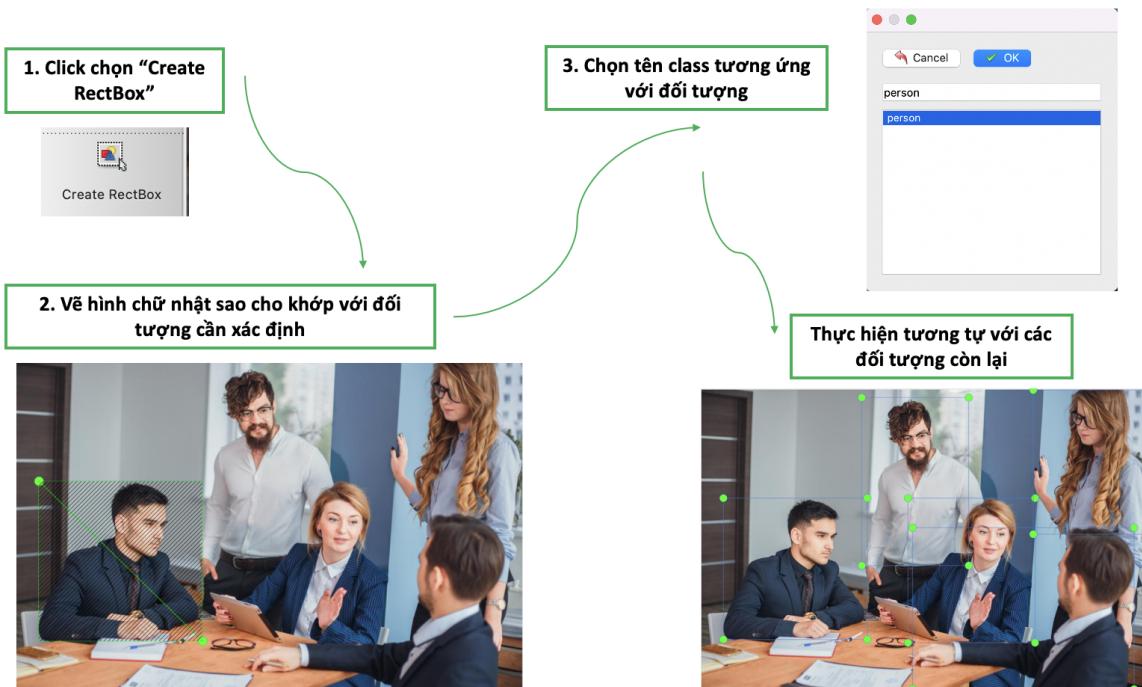
Hình 25: Mở thư mục chứa ảnh cần gán nhãn và chuyển format nhãn thành YOLO

– **Bước 3:** Nếu thành công, các bạn sẽ thấy giao diện như sau:



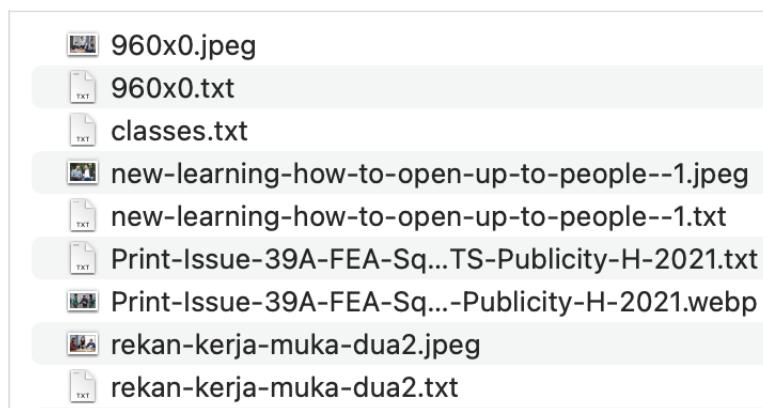
Hình 26: Giao diện sau khi chọn thư mục ảnh cần gán nhãn

– **Bước 4:** Thực hiện gán nhãn. Các bạn làm theo hình dưới đây:



Hình 27: Quá trình gán nhãn

Sau khi kết thúc gán nhãn cho một ảnh, các bạn save lại (CTRL + S) và di chuyển đi hình tiếp theo (phím D (Next Image) hoặc phím A (Prev Image)). Khi đã gán nhãn hết tất cả các ảnh, các bạn có thể kiểm tra tại thư mục chứa ảnh của mình, nếu thấy các file .txt cho từng ảnh bạn đã gán nhãn, bạn đã gán nhãn thành công:



Hình 28: Thư mục chứa ảnh sau khi gán nhãn

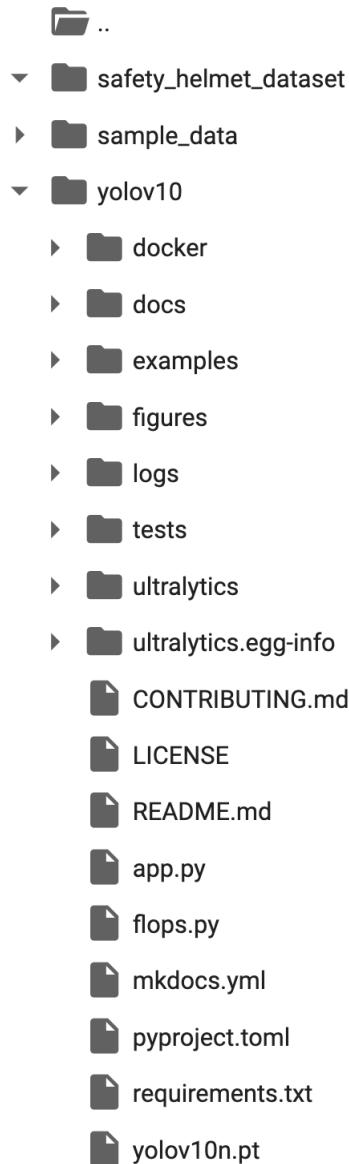
Đây là công việc tưởng chừng nhẹ nhàng nhưng lại hết sức quan trọng, vì nó sẽ ảnh hưởng rất lớn đến kết quả từ mô hình của bạn, vì vậy cần thật sự tập trung và cẩn trọng trong việc gán nhãn dữ liệu.

### III. Câu hỏi trắc nghiệm

1. Trong một chương trình **Helmet Safety Detection** với chức năng dùng để **phát hiện người có đội mũ bảo vệ hay không** trong một ảnh, Input của chương trình là gì?
  - (a) Mô hình Object Detection.
  - (b) Bounding Box.
  - (c) Ảnh bất kì.
  - (d) **Ảnh chứa người.**
2. Trong một chương trình Object Detection, Output của mô hình Object Detection là gì?
  - (a) Ảnh chứa người.
  - (b) Tọa độ bounding box nếu có.
  - (c) **Tọa độ bounding box và tên class nếu có.**
  - (d) Ảnh chứa bounding box nếu có.
3. Trong các bước cài đặt và thực hiện huấn luyện YOLOv10, bước nào sau đây không nằm trong quy trình?
  - (a) Tải pretrained model.
  - (b) **Cài đặt hàm huấn luyện.**
  - (c) Cài đặt thư viện ultralytics.
  - (d) Tải bộ dữ liệu.
4. Trong các bước sử dụng **labelImg** để gán nhãn dữ liệu huấn luyện YOLOv10, bước nào sau đây không nằm trong thao tác chính?
  - (a) **Đổi format label sang PascalVOC.**
  - (b) Chọn thư mục chứa ảnh.
  - (c) Chọn thư mục lưu label.
  - (d) Sửa tên class trong `predefine_classes.txt`
5. Trong file cấu hình của bộ dữ liệu (file `.yaml`), trường thông tin **nc** có ý nghĩa gì?
  - (a) Số mẫu dữ liệu trong dataset.
  - (b) Số lượng dataset.
  - (c) Số thứ tự của dataset.
  - (d) **Số lượng class.**
6. Lệnh nào sau đây dùng để kích hoạt môi trường ảo tên **yolo\_env** trong conda?
  - (a) `conda deactivate yolo_env`
  - (b) `conda env deactivate yolo_env`

- (c) conda env activate yolo\_env
- (d) **conda activate yolo\_env**

7. Cho hình ảnh cấu trúc cây thư mục (trong Google Colab) và đoạn lệnh sau:



```
1 YAML_PATH = './safety_helmet_dataset/data.yaml'
2 EPOCHS = 50
3 IMG_SIZE = 640
4
5 model.train(data=YAML_PATH,
6             epochs=EPOCHS,
7             imgsz=IMG_SIZE)
```

Giả sử các bạn đang ở thư mục **/content/yolov10**, khi thực thi đoạn lệnh trên có xảy ra lỗi không?

- (a) Có.  
(b) Không.
8. Trong câu lệnh huấn luyện mô hình, tham số **EPOCHS** có ý nghĩa là?
- (a) Tham số mô hình quan trọng trong quá trình huấn luyện.  
(b) Được sử dụng để chỉ định số lần lặp lại quá trình huấn luyện.  
(c) Một biến số ngẫu nhiên trong quá trình huấn luyện mô hình.  
(d) Số lần mà toàn bộ dữ liệu huấn luyện sẽ được sử dụng để cập nhật các trọng số của mô hình.
9. Trong câu lệnh huấn luyện mô hình, tham số **BATCH\_SIZE** có ý nghĩa là?
- (a) Số lần lặp lại quá trình huấn luyện mô hình.  
(b) Số lượng tham số của mô hình cần được cập nhật sau mỗi lượt huấn luyện.  
(c) Số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật gradient.  
(d) Thời gian tối đa cho mỗi lần huấn luyện mô hình.
10. Trong ngữ cảnh của project này, việc huấn luyện (training) mô hình còn được gọi với từ khóa là "**fine-tuning**", thuật ngữ này có thể được hiểu như thế nào?
- (a) Quá trình điều chỉnh lại mô hình đã được huấn luyện trước đó trên một tập dữ liệu mới để cải thiện hiệu suất hoặc thích nghi với một tác vụ cụ thể.  
(b) Quá trình huấn luyện mô hình từ đầu mà không sử dụng bất kỳ trọng số nào từ mô hình đã được huấn luyện trước đó.  
(c) Quá trình điều chỉnh siêu tham số của mô hình để tối ưu hóa độ chính xác.  
(d) Quá trình lấy mẫu một cách ngẫu nhiên từ tập dữ liệu huấn luyện để đảm bảo tính ngẫu nhiên và đại diện của dữ liệu.

- **Hết** -