Human Object Interaction

Phân tích bài toán

1. Yêu cầu

Phát hiện các hành động giúp cửa hàng biết được sự tương tác giữa khách hàng và sản phẩm như: tương tác với nhân viên, xem quảng cáo, xem menu… sử dụng ảnh tĩnh.

2. Hướng giải quyết

Để phát hiện các hành động tương tác với các sản phẩm, ta không chỉ dựa vào bounding box xung quanh khách hàng mà ta cần bounding box xung quanh các object mà khách hàng đang tương tác. Do đó ta sẽ lựa chọn các model Human Object Interaction (HOI).

- Input: ảnh tĩnh.
- Output: list <*human, object, interaction*>, trong đó *human* và *object* là các bounding box xung quanh người và object mà người đang tương tác, và *interaction* là hành động mà người đang tương tác với object.

Có 2 dạng model HOI chính:

- Model **two-stage**: các model dạng này sẽ bao gồm 2 mạng neural nối tiếp nhau: 1 mạng detector dùng để detect tất cả human và object có trong ảnh, sau đó các human và object này sẽ được bắt cặp với nhau và các đặc trưng của các cặp này sẽ được truyền qua 1 mạng downstream để phân loại hành động. Các model dạng này dễ hiểu và dễ debug, tuy nhiên chúng phải lặp qua tất cả các cặp human-object nên sẽ gây hao phí tính toán. Hơn nữa, quá trình train và optimize có thể dẫn đến mất thông tin ngữ cảnh.
- Model **one-stage**: các model dạng này sẽ trực tiếp dự đoán các cặp **<human, object, interaction>** thay vì thông qua 2 stage như trên. Tuy nhiên, các

model này thường áp dụng cấu trúc học cộng tác giữa các task dẫn đến sự can thiệp lẫn nhau giữa các task.

Ta sẽ sử dụng model **two-stage** do ta có thể sử dụng các bounding box từ detector trong model để tái sử dụng cho các module khác như counting, tracking… thay vì phải tốn thêm một model detector độc lập. Hơn nữa, sử dụng model two-stage giúp ta có thể dễ dàng debug model: ta có thể train model detector một cách độc lập, sau đó đưa vào model HOI, nếu cần cải thiện chất lượng detect ta chỉ cần train lại model detector thay vì phải train lại toàn bộ model.

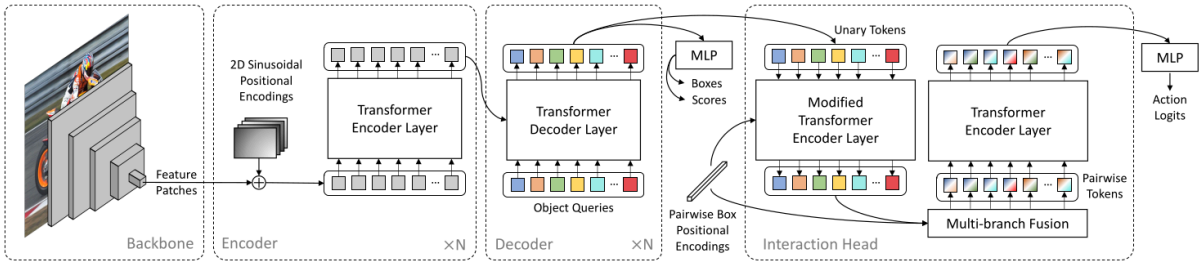
Mô hình sử dụng

Mô hình mà ta lựa chọn sử dụng sẽ là UPT (Efficient Two-Stage Detection of Human–Object Interactions with a Novel Unary–Pairwise Transformer). Source code: https://github.com/fredzzhang/upt?tab=readme-ov-file do đây là mô hình two-stage SOTA cho đến thời điểm hiện tại (hình 1) cũng như UPT có khả năng inference ở mức real-time (Tốc độ inference trên nVIDIA RTX 3090 khoảng 40ms).

Methods	Backbone	Source	Default (**mAP**↑)			Know Object (**mAP**↑)		
			Full	Rare	None-Rare	Full	Rare	None-Rare
Two-Stage Methods								
HO-RCNN [6]	CaffeNet	WACV 2018	7.81	5.37	8.54	10.41	8.94	10.85
InteractNet [2]	ResNet-50-FPN	CVPR 2018	9.94	7.16	10.77	-	-	-
GPNN [7]	ResNet-101	ECCV 2018	13.11	9.34	14.23	-	-	-
iCAN [1]	ResNet-50	BMCV 2018	14.84	10.45	16.15	16.26	11.33	17.73
PMFNet [8]	ResNet-50-FPN	ICCV 2019	17.46	15.56	18.00	20.34	17.47	21.20
No-Frills [9]	ResNet-152	ICCV 2019	17.18	12.17	18.68	-	-	-
VSGNet [10]	ResNet-152	CVPR2020	19.80	16.05	20.91	-	-	-
FCMNet [11]	ResNet-50	ECCV2020	20.41	17.34	21.56	22.04	18.97	23.12
ACP [12]	Res-DCN-152	ECCV 2020	20.59	15.92	21.98	-	-	-
PD-Net [13]	ResNet-152-FPN	ECCV 2020	20.81	15.90	22.28	24.78	18.88	26.54
SG2HOI [14]	ResNet-50	ICCV 2021	20.93	18.24	21.78	24.83	20.52	25.32
DJ-RN [15]	ResNet-50	CVPR 2020	21.34	18.53	22.18	23.69	20.64	24.60
SCG [20]	ResNet-50-FPN	ICCV 2021	21.85	18.11	22.97	-	-	-
IDN [16]	ResNet-50	NIPS 2020	23.36	22.47	23.63	26.43	25.01	26.85
ATL [17]	ResNet-50	CVPR 2021	23.81	17.43	25.72	27.38	22.09	28.96
UPT [18]	ResNet-101-DC5	CVPR 2022	**32.62**	**28.62**	**33.81**	**36.08**	**31.41**	**37.47**
One-Stage Methods								

Hình 1. Hiệu suất của các model HOI two-stage trên bộ dữ liệu HICO-DET.

1. Tổng quan kiến trúc model:



Hình 2. Tổng quan kiến trúc UPT.

Hình ảnh đầu vào sẽ được truyền qua một mạng CNN để rút trích đặc trưng, sau đó sẽ được chia thành các patch và tăng cường với positional encoding dạng hình sin. Những token này sẽ được truyền qua DETR với encoder-decoder tạo ra các đặc trưng kích thước cố định tương ứng với số lượng object tối đa. Những đặc trưng này sau đó tiếp tục được decode bởi một MLP thành classification score và bounding box, sau đó được đưa vào interaction head như *unary* token. Interaction head cũng nhận *pairwise positional encoding* được tính từ tọa độ bouding box. Một transformer sẽ tinh chỉnh *unary* token dùng *pairwise positional encoding*. Output token sẽ được bắt cặp và kết hợp với cùng *positional encoding* để tạo ra *pairwise token*, sau đó các token này sẽ được truyền qua một layer transformer và MLP để decode đặc trưng thành action score.

2. Cấu trúc dataset sử dụng

Thông thường các model HOI sẽ sử dụng 2 dạng dataset: HICO-DET và VCOCO. Trong ví dụ này, ta sẽ sử dụng VCOCO, định dạng annotation của VCOCO là 1 file json chứa các field sau đây:

- **annotations**: list chứa các thông tin gán nhãn của VCOCO như box của human, box của object, interaction giữa human-object, nhãn của object và path của hình.
- **classes**: chứa tên của các action hợp lệ.
- **objects:** chứa tên các object hợp lệ.
- **images:** id của các hình.
- **action_to_object**: index của các object hợp lệ tương ứng với index từng action. Ví dụ: objects: [“background", “person", “mouse", keyboard"), actions: [“hold", “tap"]. Ta có 2 action hợp lệ là “hold mouse" và “tap keyboard". Do đó action “hold" sẽ có object “mouse" hợp lệ và action “tap" sẽ có action “keyboard" hợp lệ. Index trong list object của “mouse" là 2 và “keyboard" là, index trong list action của “hold" là 0 và “tap" là 1. Vì vậy action_to_object sẽ có dạng: [[2], [3]] - object hợp lệ ở action có index 0 là [2] và object hợp lệ ở action có index 1 là [3].

Field annotations là một list chứa các dict có dạng như sau:

```
{
      "boxes_h": [
        [165.31, 107.62, 394.93, 379.94], # index 0
        [165.31, 107.62, 394.93, 379.94]  # index 1
```

```
    ],
    "boxes_o": [
      [207.19, 238.22, 254.63, 287.93], # index 0
      [207.19, 238.22, 254.63, 287.93]  # index 1
    ],          #0  #1
    "actions": [3, 21],
               #0  #1
    "objects": [33, 33],
    "file_name": "COCO_train2014_000000000368.jpg"
  }
```

Trong đó:

- **boxes_h:** chứa tọa độ của các bounding box người.
- **boxes_o:** chứa tọa độ của các bounding box object.
- **actions:** chứa index của các action tương ứng với từng cặp human-object trong *boxes_h* và *boxes_o* theo thứ tự. Ví dụ: trong dict ở trên, action index 3 có index trong actions là 0 tương ứng với cặp *boxes_h[0]* và *boxes_o[0]*.
- **objects:** chứa index các object tương ứng với từng box trong *boxes_o*. Ví dụ: trong dict ở trên, objects index 33 có index trong objects là 0 là class của *box_o[0]*.

3. Tinh chỉnh mô hình để sử dụng trên custom dataset

Để train UPT trên dataset tùy chỉnh, ta cần tinh chỉnh UPT theo các bước sau:

1. Train object detector mới trên dữ liệu tùy chỉnh.

Do UPT được train trên DETR (source code: https://github.com/facebookresearch/detr) nên ta có thể dễ dàng sử dụng repo này train detector nhằm làm pretrained detector cho UPT.

Các bước train detector này như sau:

- Đầu tiên, chuẩn bị dữ liệu theo format COCO. Tìm hiểu tại đây: https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-coco-overview.html .
- Chuẩn bị dữ liệu theo format sau:

```
|—detr # Thư mục gốc của DETR
|—|—coco2017 # Thư mục chứa dataset
|—|—|—annotations
|—|—|—|—instances_train2017.json # File annotation của tập train
```

```
|—|—|—|—instances_val2017.json # File annotation của
tập val
|—|—|—train2017 # Hình trong tập train
|—|—|—val2017 # Hình trong tập val
|—|...
```

- Download pretrained DETR do các model họ transformer rất cần data, nếu dataset của ta quá nhỏ thì cần sử dụng pretrained để DETR có thể hội tụ.
- Chạy lệnh: `python main.py --coco_path coco2017 --world_size 2 --batch_size 8 --output_dir checkpoints --resume detr-r50-e632da11.pth --epochs 200` để train DETR, checkpoint của model sẽ được lưu vào folder *checkpoints*.

2. Custom code của UPT để phù hợp với dataset

Các thay đổi dưới đây là các thay đổi so với repo gốc

- Chuẩn bị các file annotation theo format VCOCO như đã mô tả.
- Chuẩn bị dữ liệu theo format sau:

```
|—upt # Thư mục gốc của UPT
|—|vcoco # Thư mục chứa class VCOCO và các dữ liệu
liên quan
|-|-...
|-|-instances_vcoco_trainval.json # File annotation
cho tập train và val
|-|-instances_vcoco_train.json # File annotation cho
tập train
|-|-instances_vcoco_val.json # File annotation cho tập
val
|-|-instances_vcoco_test.json # File annotation cho
tập test
|-|-mscoco2014
|-|-|-train2014 # Hình ảnh trong tập train
|-|-|-val2014 # Hình ảnh trong tập val
|-|-|-
|-|...
```

- Clone repo UPT - https://github.com/fredzzhang/upt .
- Điều chỉnh một số code nhằm tối ưu cho model hoạt động:

1. Đầu tiên, ta cần chỉnh sửa số lượng index của các object cho phù hợp với bộ dataset ở `UPT/vcoco/vcoco.py`, ta cần sửa phương thức `object_to_action` trong class `VCOCO`:

Phương thức này sẽ trả về một dict chứa các action hợp lệ tương ứng với từng object, ví dụ:

```
{
    0: [1, 2, 3],
    1: [2, 3, 4, 7],
}
```

Trong ví dụ trên, phương thức này trả về một dict với action index 0 có 3 object hợp lệ là [1, 2, 3] và action index 1 có 4 object hợp lệ là [2, 3, 4, 7].

Ban đầu: `object_to_action = {obj: [] for obj in list(range(1, 81))}`

Ta cần sửa đổi sao cho phương thức này trả về một dict đúng với object hợp lệ cho từng action mà ta mong muốn bằng cách sửa đổi dict khởi tạo `object_to_action`. Ví dụ:

```
object_to_action = {obj: [] for obj in list(range(0, 4))}
```

Để kiểm tra, ta có thể khởi tạo một instance của class `VCOCO`, sau đó gọi attribute `object_to_action`.

2. Lọc ra các label không sử dụng trong `upt/upt.py`, phương thức `prepare_region_proposals` trong class `UPT`:

Nếu sử dụng pretrained để fine-tune DETR, có thể sẽ có một số class không sử dụng trong dataset của chúng ta (do số lượng class ở pretrained DETR khác với số lượng class trong dataset), do đó trong hàm `prepare_region_proposals`, ta cần lọc các label này đi:

```
keep = điều kiện lọc, ví dụ các class trong dataset từ 0 đến 3, như vậy keep = lb < 4
sc = sc[keep].view(-1)
lb = lb[keep].view(-1)
bx = bx[keep].view(-1, 4)
```

```
hs = hs[keep].view(-1, 256)
```

3. Chỉnh sửa num_classes trong `upt/detr/models/detr.py`

Tương tự như ở trên , ta cần chỉnh sửa `num_classes` ở hàm build để DETR nhận đúng số lượng classes của ta, mặc định UPT là 80, ta cần sửa lại cho phù hợp.

4. Chỉnh sửa code khởi tạo các cặp human-object cho phù hợp trong `upt/interaction_head.py`, phương thức `forward:`

Ban đầu:

```
x_keep, y_keep = torch.nonzero(torch.logical_and(x != y, x < n_h)).unbind(1)
```

Trong code ban đầu, có thể các cặp human-human vẫn được match, do đó ta sẽ sửa lại như sau:

```
x_keep, y_keep = torch.nonzero(torch.logical_and(x != y, torch.logical_and(x < n_h, y >= n_h))).unbind(1)
```

3. Điều chỉnh các siêu tham số trong quá trình training UPT sao cho phù hợp.

```
python main.py --world-size 1 --dataset vcoco
--data-root vcoco/ --partitions trainval trainval
--output-dir checkpoints/upt-r50-vcoco
--print-interval 1 --epochs 300 --batch-size 8
--min-instances 1 --box-score-thresh 0.8 --lr-head
5e-8 --fg-iou-thresh 0.70 --pretrained
/kaggle/input/detr-real/checkpoint_new.pth
--num-classes 2 --human-idx 1
```

Một số siêu tham số quan trọng:

- **world-size:** số lượng GPU sử dụng để train.
- **dataset**: định dạng dataset - trong trường hợp này là vcoco.
- **data-root**: thư mục chứa data.
- **partitions**: subset của data dùng trong quá trình train và eval.
- **output-dir:** thư mục chứa các checkpoint.
- **epochs**: số lượng epoch để train.
- **batch-size**: kích thước batch.
- **min-instances:** số lượng interaction để lấy mẫu.
- **box-score-thresh**: threshold của object detection.
- **lr-head**: learning rate.

- **fg-iou-thresh**: threshold cho iou match giữa bounding box người và object giữa ground truth và detect được từ DETR. Các cặp human-object có iou giữa ground truth và detect được từ DETR lớn hơn ngưỡng này mới được xử lý tiếp theo.
- **pretrained:** checkpoint của pretrained DETR.
- **human-idx**: index của human object trong DETR.
- **num-classes**: số lượng action.

4. Tinh chỉnh code để tích hợp với các module khác

Như đã đề cập ở trên, ta có thể tận dụng detector trong two-stage HOI để làm output cho các module khác. Ta có thể tận dụng điều này bằng cách:

- Trong phương thức inference của UPT, thay vì trả về các <human, object, interaction>, ta sẽ tùy chỉnh hàm này để sử dụng bounding box từ DETR, sau đó thực hiện tracking, counting. Ví dụ: nếu sử dụng tracking thì ta sẽ thêm field vào phương thức `prepare_region_proposals` và `post_processing` để UPT có thể giữ lại field ID của từng người, cho ta biết được ID người nào đang có interaction.