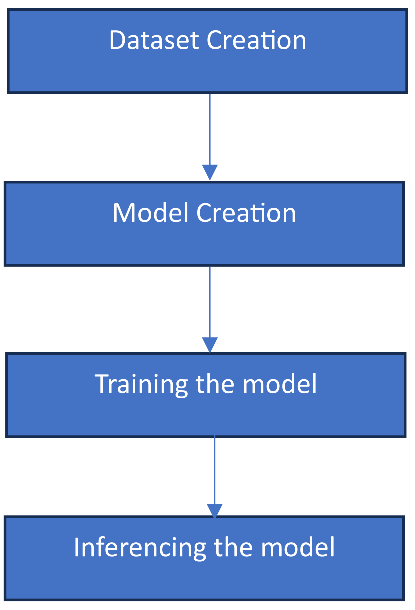
MATRICE HIRING: CODING ASSIGNMENT

Mogilipalem Hemanth Kumar

The includes two different tasks. One is Object detection (bounding box) and instance segmentation. In object detection, we assign a class label to bounding boxes that contain objects. Instance Segmentation involves classifying each pixel or voxel of a given image or volume to a particular class and assigning a unique identity to the pixels of individual objects.



I followed the above four stage approach to solve this assignment. First step is creating the dataset. Then model creation. Further, the model is trained using the dataset created and finally, inferencing the model with few examples.

Dataset Creation:

Initially, I used the **requests** library to download a tar archive file from https://s3.us-west-2.amazonaws.com/testing.resources/datasets/deepfashion/deep_fashion.tar.

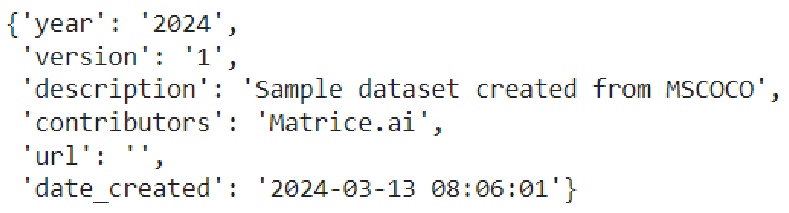
Subsequently, utilized the **tarfile** module, it unpacks the contents of the downloaded archive into a designated directory.

In this dataset, there are two different folders:

1. Annotations
2. Images

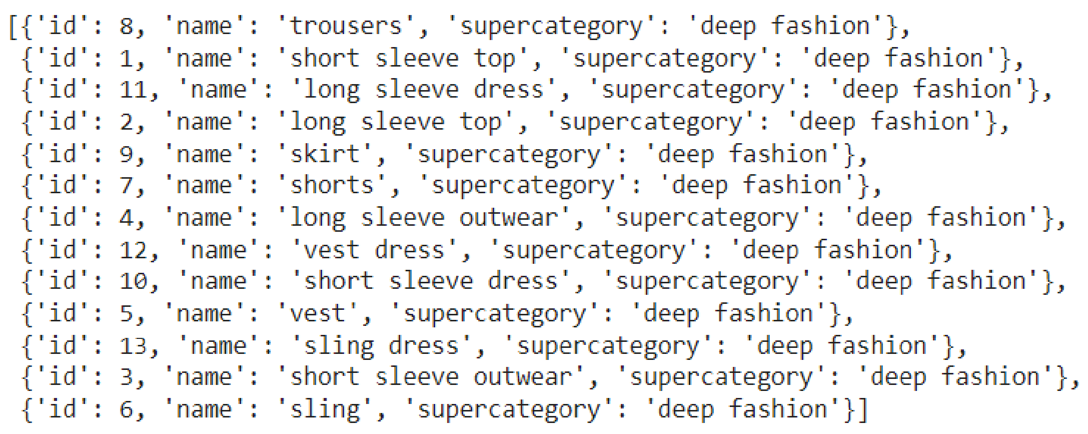
In the Annotations folder, it's actually a json file. That contains four keys namely ['info', 'categories', 'annotations', 'images'].

Info contains the metadata for the given dataset:



```
{'year': '2024',
 'version': '1',
 'description': 'Sample dataset created from MSCOCO',
 'contributors': 'Matrice.ai',
 'url': '',
 'date_created': '2024-03-13 08:06:01'}
```

There are 13 categories in the dataset as shown in the below figure.



```
[{'id': 8, 'name': 'trousers', 'supercategory': 'deep fashion'},
 {'id': 1, 'name': 'short sleeve top', 'supercategory': 'deep fashion'},
 {'id': 11, 'name': 'long sleeve dress', 'supercategory': 'deep fashion'},
 {'id': 2, 'name': 'long sleeve top', 'supercategory': 'deep fashion'},
 {'id': 9, 'name': 'skirt', 'supercategory': 'deep fashion'},
 {'id': 7, 'name': 'shorts', 'supercategory': 'deep fashion'},
 {'id': 4, 'name': 'long sleeve outwear', 'supercategory': 'deep fashion'},
 {'id': 12, 'name': 'vest dress', 'supercategory': 'deep fashion'},
 {'id': 10, 'name': 'short sleeve dress', 'supercategory': 'deep fashion'},
 {'id': 5, 'name': 'vest', 'supercategory': 'deep fashion'},
 {'id': 13, 'name': 'sling dress', 'supercategory': 'deep fashion'},
 {'id': 3, 'name': 'short sleeve outwear', 'supercategory': 'deep fashion'},
 {'id': 6, 'name': 'sling', 'supercategory': 'deep fashion'}]
```

The annotations contains keys as follows [segmentation, landmark, category_id, bbox,is_crowd, image_id,id].

I analyzed and filtered data from a JSON file containing annotations for the Deep Fashion dataset. Initially, it reads the JSON file, parsing its contents into a Python dictionary using the **json.load** function. Then, it iterates over the annotations, counting the occurrences of each class across all images and storing the counts in a dictionary called **class_counts**. Subsequently, it filters out classes with more than 50 instances, creating a list of included classes named **included_classes**. Following this, it filters images based on the included classes, appending the IDs of relevant images to the list **included_images.** In this way, I Only included classes with more than 50 images over the three splits.
Now, from this images 500 images are selected randomly and divided into train(70%), val (20%) and test(10%). As this is a supervised learning task. Inorder to create labels, I used the annotations part of dataset.

In YOLO Instance segmentation format, annotations are stored in a separate **.txt** file corresponding to each image, with both image and annotation files sharing the same base filename.

Each row in a .txt file corresponds to a single object instance found in that associated image file.

Each row contains the following information about an instance:

Object Class Index: An integer referring to the object's class

Object Bounding Coordinates: These are the normalized bounding coordinates around the object's segmented area, ensuring that annotations are scalable across images of different dimensions.

Therefore, each row in a .txt file would look like

<class-index> <x1> <y1> <x2> <y2> ... <xn> <yn>

Now, for each image, a corresponding .txt annotation file is created, category_id is extracted and polygon of the current object instance in a recurring loop. The polygon is then normalized with relative to the current image dimensions. Thus, a set of .txt files for all images is created in the output labels dir, which will be utilized during fine-tuning.

We will dump the required data set in a separate **data.yaml** file to fit our training pipeline for any YOLO models. This file contains the paths for train, validation, and test image directories along with the names of classes and the number of classes in the dataset.

Image:

Label:

```
7 0.982906 0.701923 0.799145 0.775641 0.536325 0.833333 0.574786 0.939103 0.636752 0.996795 0.876068 0.998397 0.876068 0.908654 0.882479
0.947115 0.897436 0.998397 0.993590 0.996795 0.974359 0.899038 0.995726 0.807692 0.982906 0.701923
0 0.549145 0.056090 0.557692 0.142628 0.487179 0.197115 0.292735 0.165064 0.096154 0.145833 0.002137 0.282051 0.000000 0.532051 0.100427
0.716346 0.322650 0.642628 0.301282 0.586538 0.275641 0.525641 0.301282 0.583333 0.467949 0.777244 0.585470 0.966346 0.856838 0.945513
0.997863 0.804487 0.944444 0.538462 0.788462 0.312500 0.743590 0.221154 0.775641 0.261218 0.794872 0.315705 0.925214 0.219551 0.846154
0.147436 0.728632 0.056090 0.549145 0.056090
```

Model Creation:

Training code is taken from https://github.com/WongKinYiu/yolov9/tree/main

Added another head to YOLOv9-c model architecture and named it as YOLOv9c-seg so that you can predict both detection (bounding box) and instance segmentation mask at same time, along with category and confidence scores.

I used a pre-trained YOLOv9c-seg model weights on the **COCO dataset** with **all layers unfreezed**.

Modified the training script to takes a single argument which contains a yaml config file that has all the arguments and hyperparameters.

1. **Existing YOLOv9-c Architecture**:
   - YOLOv9-c (Compact) already consists of a backbone (CSPDarknet53), a neck (PANet), and a detection head with three YOLO layers.
   - The existing detection head predicts bounding boxes, object ness scores, and class probabilities.
2. **Adding an Instance Segmentation Head**:
   - We'll introduce a new head specifically for instance segmentation.
   - This head will predict pixel-wise masks for each detected object.
3. **Combined Architecture**:
   - The combined architecture will have two parallel heads:
     - **Detection Head**:
       - Predicts bounding boxes, object ness scores, and class probabilities (similar to the existing YOLOv9-c head).
     - **Instance Segmentation Head**:
       - Predicts binary masks for each object instance.
       - These masks indicate which pixels belong to the detected object.
4. **Output**:
   - The final output will include:

- Bounding boxes (coordinates and sizes).
- Object ness scores (confidence in the presence of an object).
- Class probabilities (category labels).
- Instance segmentation masks (pixel-wise masks).

Model Training:

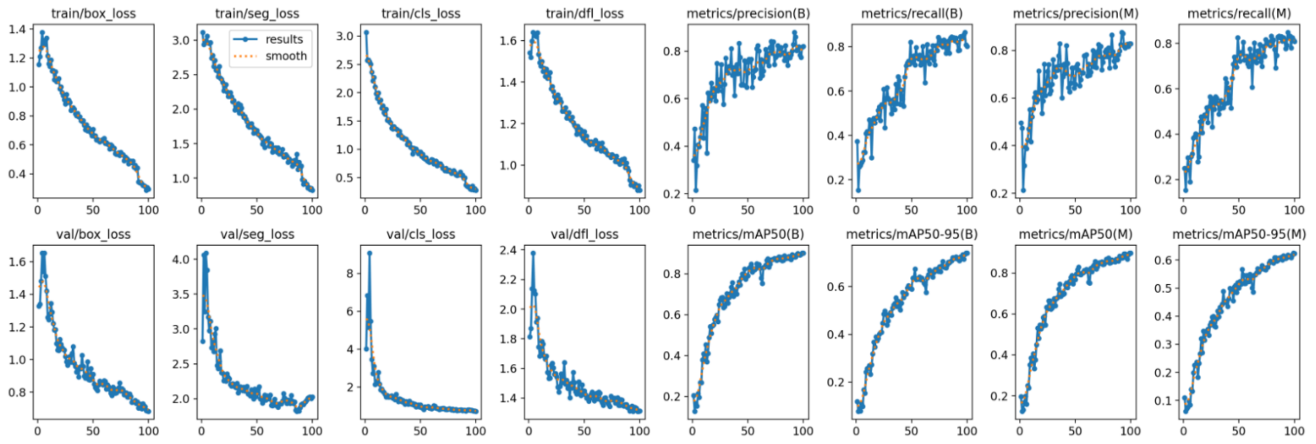
The parameters used for training the model are:

Parameter	Value
Epochs	100
Patience	0
Batch size	4

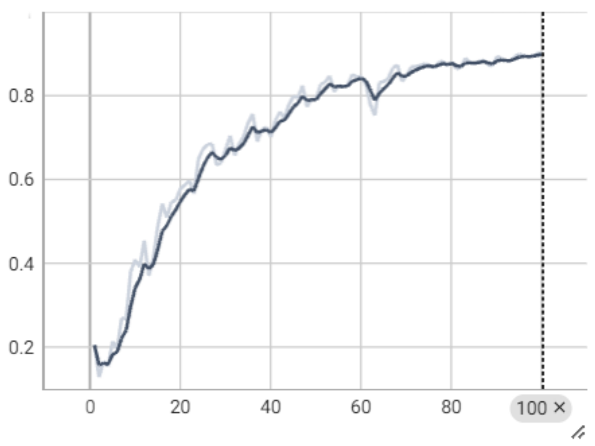
System Specifications:

Hardware	Specifications
GPU	1xTesla K80 , compute 3.7, having 2496 CUDA cores , 12GB GDDR5 VRAM
CPU	1xsingle core hyper threaded Xeon Processors @2.3Ghz i.e(1 core, 2 threads)
RAM	12.6 GB Available
DISK	33 GB Available

Let’s look at the **results.png** . The Ultralytics training summary and results.png gives an overall idea of how the model fine-tuning has been.

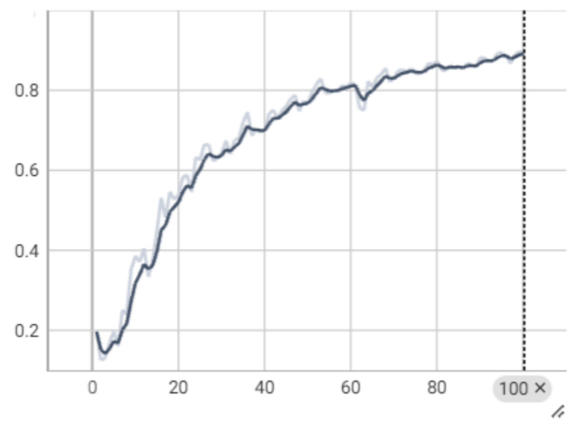


metrics/mAP50(B)



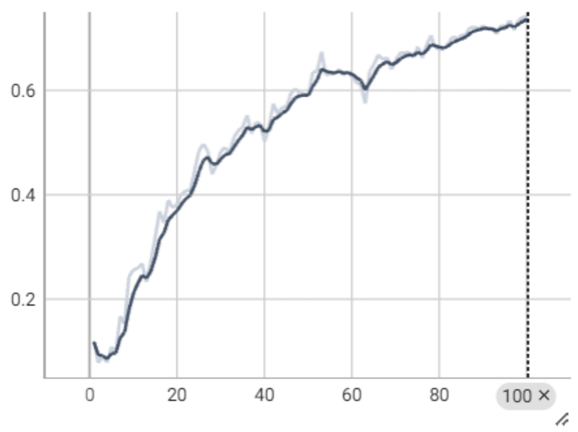
Run ↑	Smoothed	Value	Step	Relative
● .	0.8984	0.9008	100	1.142 hr

metrics/mAP50(M)



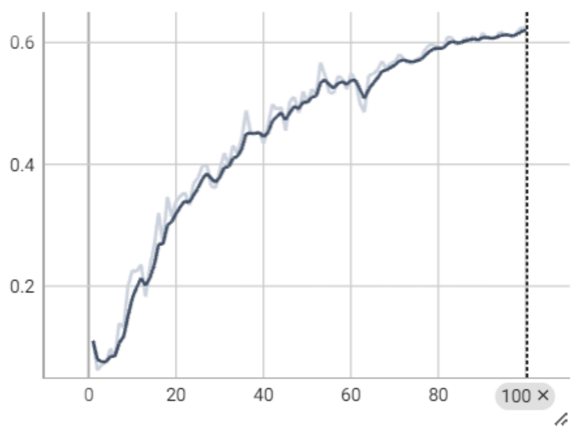
Run ↑	Smoothed	Value	Step	Relative
● .	0.8917	0.8964	100	1.142 hr

metrics/mAP50-95(B)



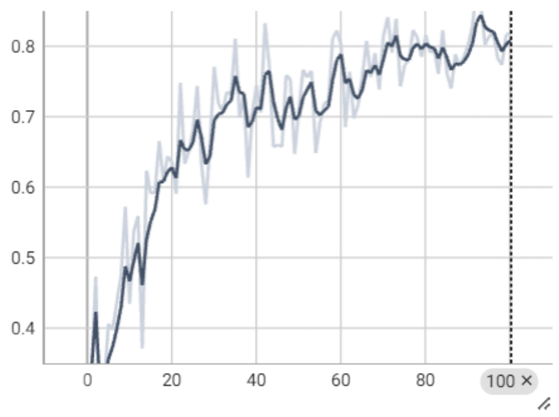
Run ↑	Smoothed	Value	Step	Relative
● .	0.7362	0.7426	100	1.142 hr

metrics/mAP50-95(M)



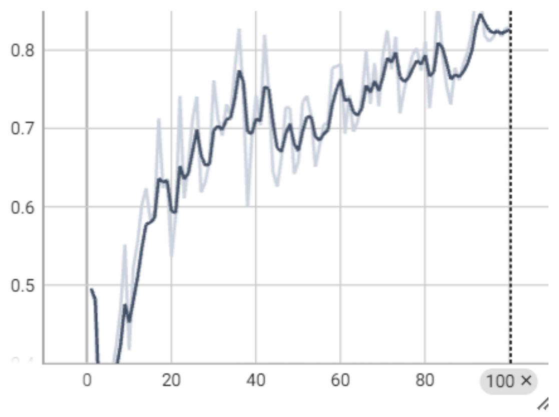
Run ↑	Smoothed	Value	Step	Relative
● .	0.6205	0.6239	100	1.142 hr

metrics/precision(B)

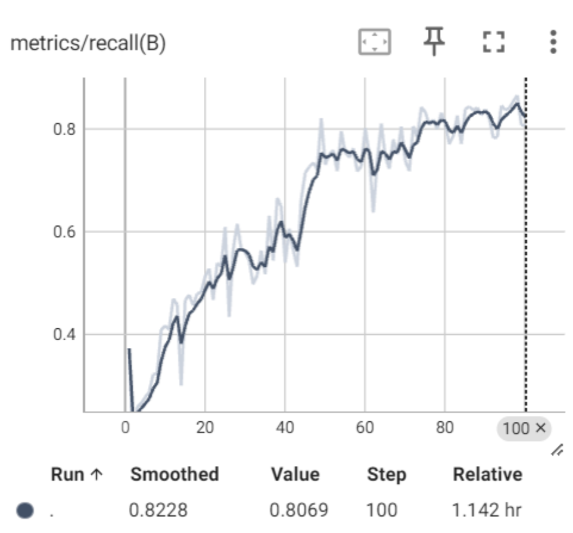


Run ↑	Smoothed	Value	Step	Relative
● .	0.8084	0.8175	100	1.142 hr

metrics/precision(M)

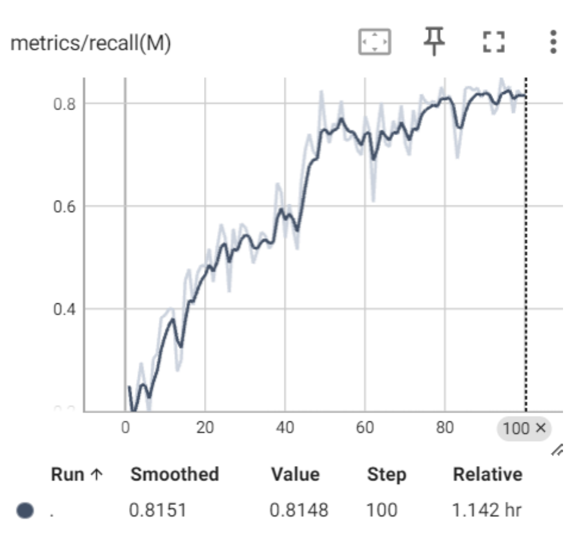


Run ↑	Smoothed	Value	Step	Relative
● .	0.8258	0.8285	100	1.142 hr



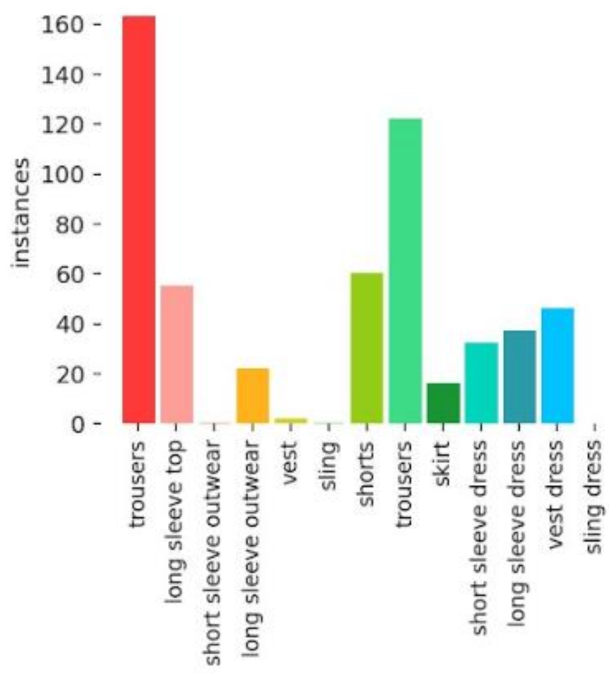
metrics/recall(B)

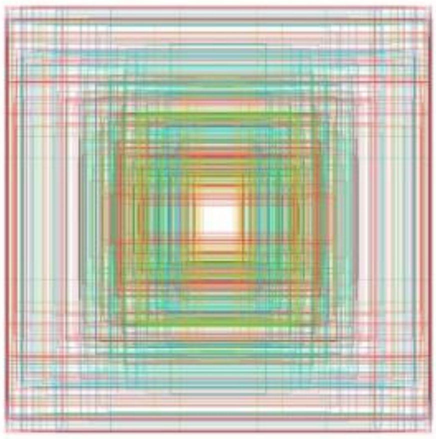
Run ↑	Smoothed	Value	Step	Relative
● .	0.8228	0.8069	100	1.142 hr

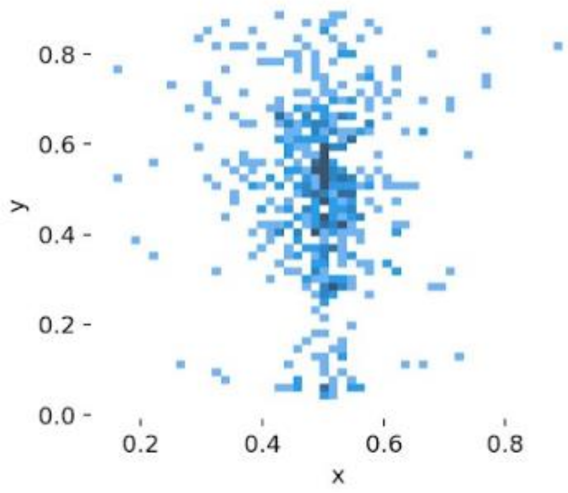


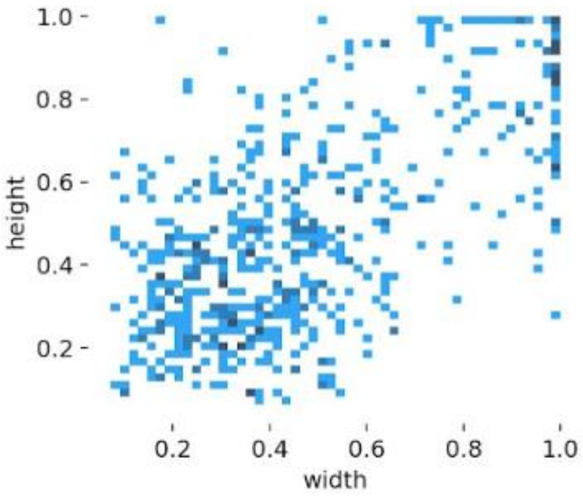
metrics/recall(M)

Run ↑	Smoothed	Value	Step	Relative
● .	0.8151	0.8148	100	1.142 hr









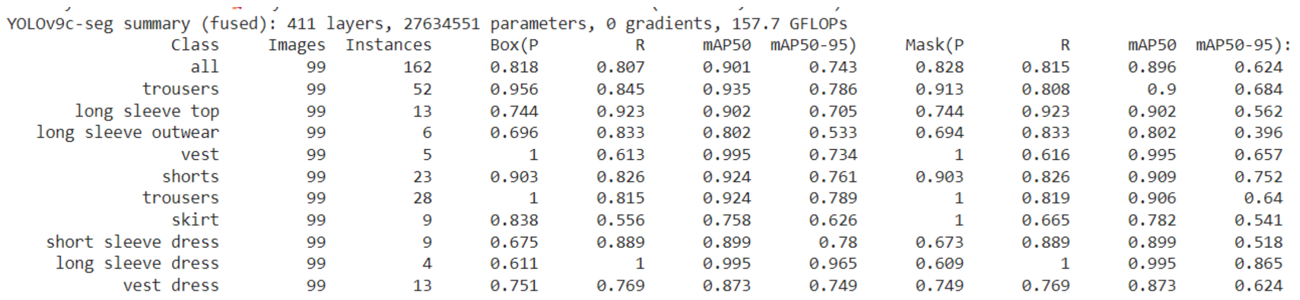
Model Inference:

The following are metrics used and their respective values:

Metric	Value
mAP50(B)	0.8984
mAP50(M)	0.8917
mAP50-95(B)	0.7362
mAP50-95(M)	0.6205
Precision(B)	0.8084
Precision(M)	0.8258
Recall(B)	0.8228
Recall(M)	0.8151

Loss	Value
Box Loss	0.2936
Segmentation loss	0.825
Cls loss	0.2758
Dfl loss	0.8788

Class wise Predictions:



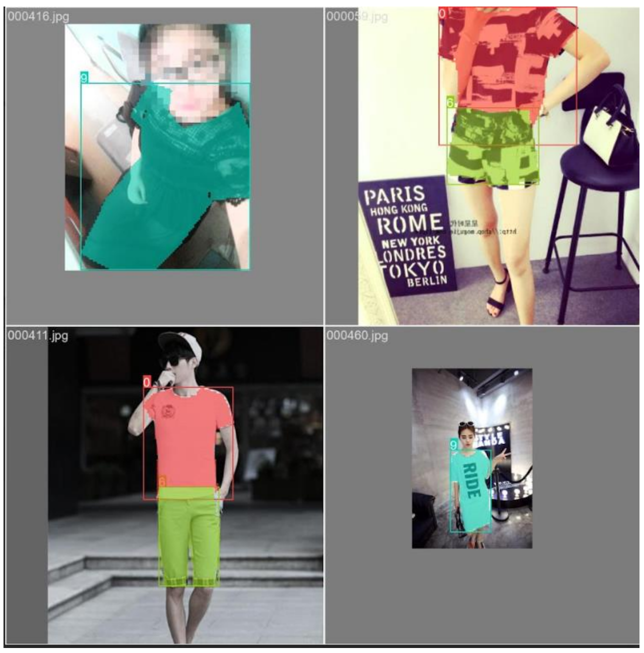
```
YOLOv9c-seg summary (fused): 411 layers, 27634551 parameters, 0 gradients, 157.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95):
                   all         99        162      0.818      0.807      0.901      0.743      0.828      0.815      0.896      0.624
              trousers         99         52      0.956      0.845      0.935      0.786      0.913      0.808        0.9      0.684
        long sleeve top         99         13      0.744      0.923      0.902      0.705      0.744      0.923      0.902      0.562
   long sleeve outwear         99          6      0.696      0.833      0.802      0.533      0.694      0.833      0.802      0.396
                  vest         99          5          1      0.613      0.995      0.734          1      0.616      0.995      0.657
                shorts         99         23      0.903      0.826      0.924      0.761      0.903      0.826      0.909      0.752
              trousers         99         28          1      0.815      0.924      0.789          1      0.819      0.906       0.64
                 skirt         99          9      0.838      0.556      0.758      0.626          1      0.665      0.782      0.541
    short sleeve dress         99          9      0.675      0.889      0.899       0.78      0.673      0.889      0.899      0.518
     long sleeve dress         99          4      0.611          1      0.995      0.965      0.609          1      0.995      0.865
            vest dress         99         13      0.751      0.769      0.873      0.749      0.749      0.769      0.873      0.624
```

The evaluation metrics for the Deep Fashion dataset reveal promising performance across various aspects of object detection and segmentation. With a mean Average Precision (mAP) of 0.8984 for bounding boxes and 0.8917 for masks, the model demonstrates strong capability in accurately localizing and segmenting objects within images. However, the mAP values drop slightly to 0.7362 (bounding boxes) and 0.6205 (masks) when considering a wider confidence interval of 50% to 95%, indicating a reduced performance margin at higher confidence thresholds. Precision scores of 0.8084 (bounding boxes) and 0.8258 (masks) suggest a high level of accuracy in object detection, while recall rates of 0.8228 (bounding boxes) and 0.8151 (masks) indicate effective retrieval of relevant objects within the dataset.

On the training front, the model exhibits reasonable loss values across different components. The Box Loss stands at 0.2936, indicating a relatively low discrepancy between predicted bounding box coordinates and ground truth annotations. Segmentation Loss, with a value of 0.825, signifies the model's ability to accurately delineate object boundaries. Classification Loss (Cls loss) is recorded at 0.2758, indicating efficient classification of objects into respective categories. Additionally, the model incurs a Dfl (Domain-Friendly Loss) of 0.8788, reflecting its adaptability and performance across different domains

Here are the few Example Predictions:

During Training:



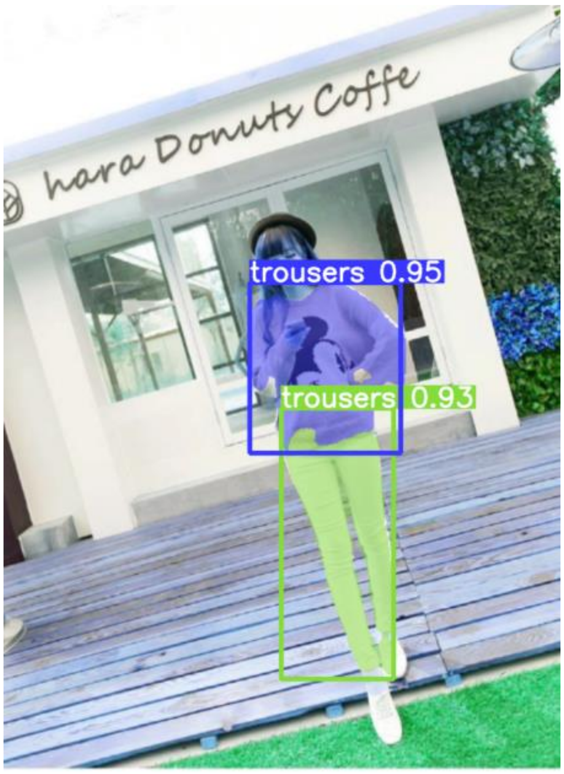
Note:

pycodestyle is an equivalent of pylint for Jupyter Notebook which is able to check my code against the PEP8 style guide.

During Testing:

Original Image

Predicted Image



Original Image

Predicted Image