COMPUTER VISION ASSIGNMENT DOCUMENTATION

Step 1: I have used the below code to make a sample dataset of 500 images and then split it into train,val,test folders in the ratio 7:2:1

The below code has a pylint score of 10/10 (mentioned in the code notebook)

```
import zipfile
import os
import random
zip_file_path = "train.zip"
output_zip_path = "selected_images.zip"
def select_random_images(zip_file_path, num_images=500):
    selected_images = []
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        image_filenames = [name for name in zip_ref.namelist() if name.endswith('.jpg')]
        selected_images = random.sample(image_filenames, num_images)
    return selected_images
def create_zip_with_selected_images(zip_file_path, selected_images, output_zip_path):
    with zipfile.ZipFile(output_zip_path, 'w') as output_zip:
        with zipfile.ZipFile(zip_file_path, 'r') as input_zip:
            for image_name in selected_images:
                with input_zip.open(image_name) as image_file:
                    output_zip.writestr(image_name, image_file.read())
selected_images = select_random_images(zip_file_path)
create_zip_with_selected_images(zip_file_path, selected_images, output_zip_path)
print("Selected images saved to:", output_zip_path)
```

```python
#performing the next step that is dividing the dataset into train,val,test
#in the ratio 7:2:1
import shutil
def split_dataset(source_folder, train_folder, val_folder, test_folder, split_ratio=(0.7, 0.2, 0.1)):
    for folder in [train_folder, val_folder, test_folder]:
        if not os.path.exists(folder):
            os.makedirs(folder)
    image_files = [file for file in os.listdir(source_folder) if file.endswith(('.jpg', '.jpeg', '.png'))]
    random.shuffle(image_files)
    num_images = len(image_files)
    num_train = int(num_images * split_ratio[0])
    num_val = int(num_images * split_ratio[1])
    num_test = num_images - num_train - num_val
    for i, file in enumerate(image_files):
        if i < num_train:
            shutil.copy(os.path.join(source_folder, file), os.path.join(train_folder, file))
        elif i < num_train + num_val:
            shutil.copy(os.path.join(source_folder, file), os.path.join(val_folder, file))
        else:
            shutil.copy(os.path.join(source_folder, file), os.path.join(test_folder, file))
source_folder = "selected_images"
train_folder = "train"
val_folder = "val"
test_folder = "test"
split_ratio = (0.7, 0.2, 0.1)
split_dataset(source_folder, train_folder, val_folder, test_folder, split_ratio)
```

STEP2: For performing this step I have used the labelme software in order convert the images to MSCOCO format first in order to have the annotations and then used the labelme2yolo library to convert the MSCOCO formatted files to yolo format ones

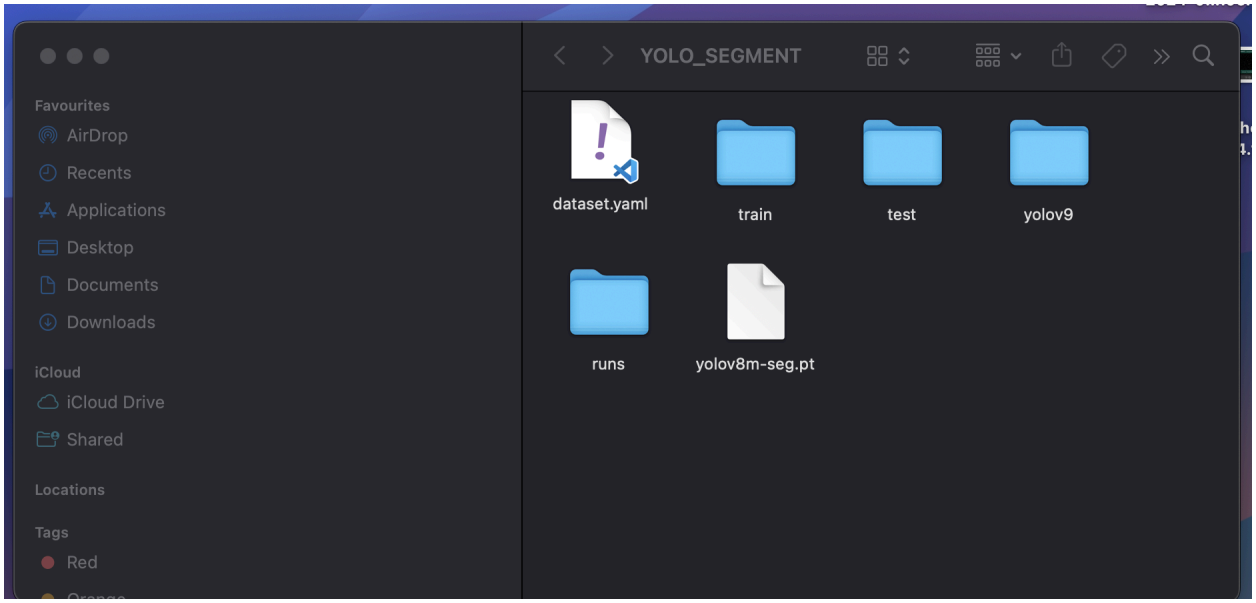
I have done this step in my terminal as I use macintosh and it is easy to install all the dependencies there

COMMANDS USED:1) pip install labelme

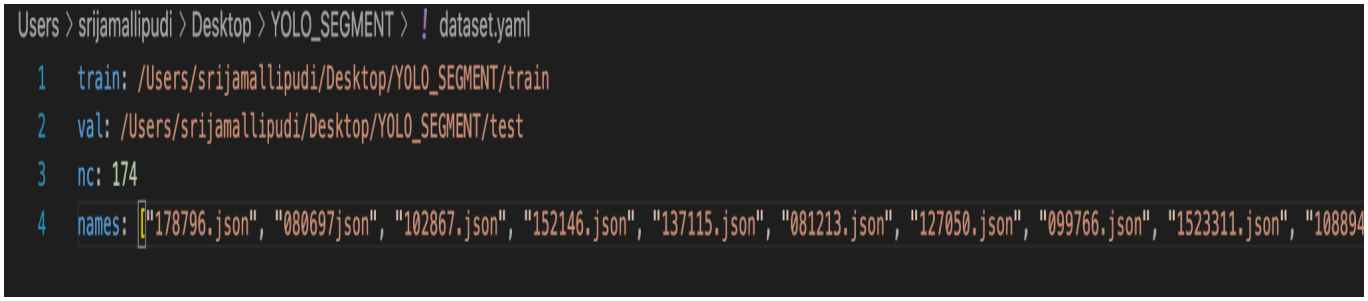
2) pip install labelme2yolo

3) labelme2yolo –json_dir dataset/train

3a) labelme2yolo –json_dir dataset/train → this folder gave me the dataset.yaml file for the yolo converted files



This was how I had the train , test and the **dataset.yaml** file after the completing the above step that i have explained



This is how my dataset.yaml file looked, I did not spend much time on giving images various names in order to reduce classes as our task is instance segmentation and not detection of specific type of dress like that, that's the reason why there are 174 classes.

STEP3:

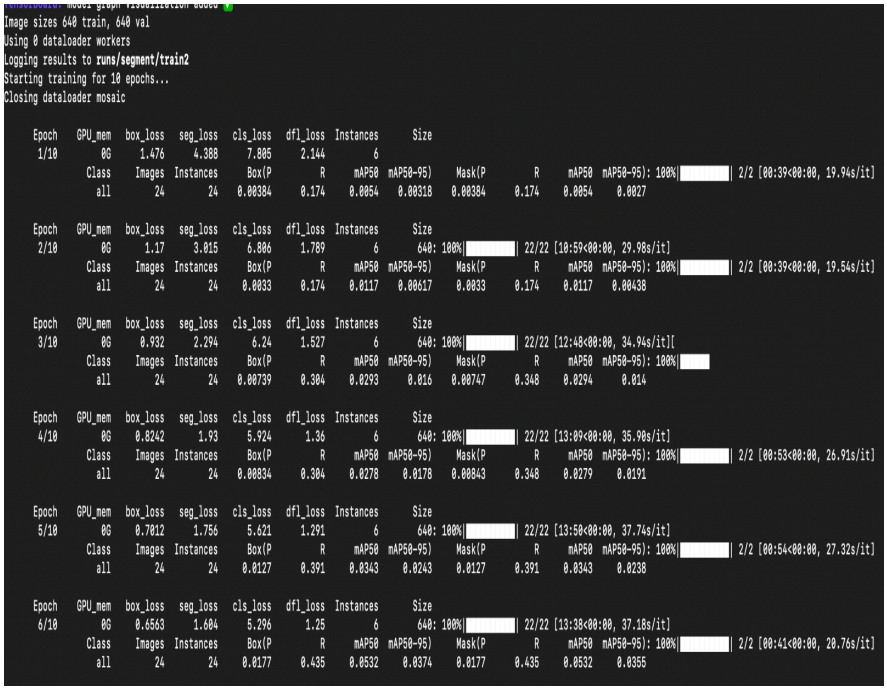
I have used the yolov8m-seg.pt model to train this instance segmentation model, so i have installed the required dependencies using the command

→ pip install ultralytics

Then, I have run the below command in the terminal to train the model on the yolov8m-seg.pt Model and the results were,

```
YOLO_SEGMENT % yolo task=segment mode=train epochs=10 data=dataset.yaml
model=yolov8m-seg.pt imgsz=640 batch=8
```

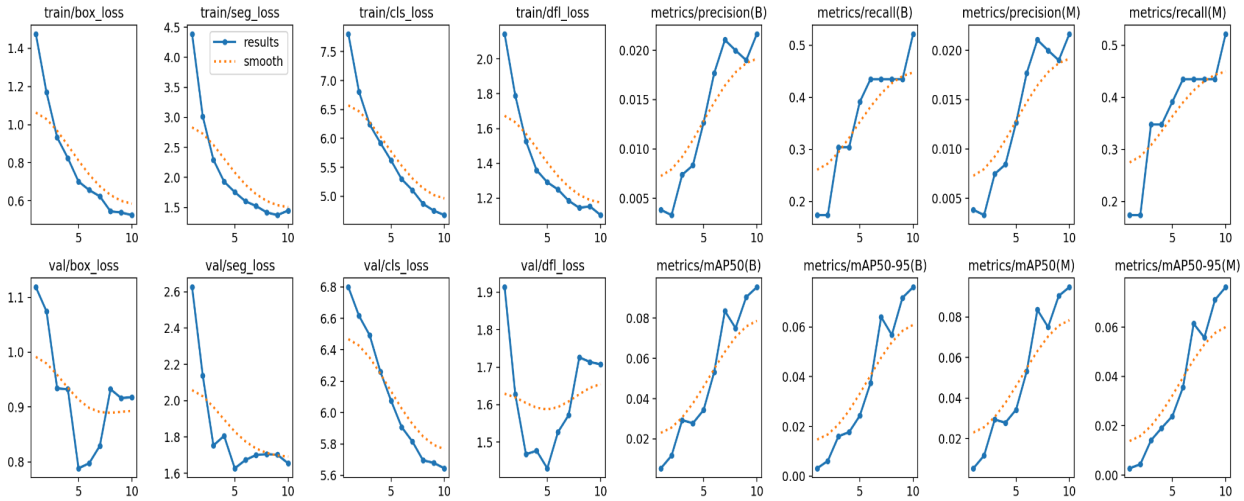
```
(base) srijamallipudi@Srijas-MacBook-Pro ~ % cd Desktop
(base) srijamallipudi@Srijas-MacBook-Pro Desktop % cd YOLO_SEGMENT
(base) srijamallipudi@Srijas-MacBook-Pro YOLO_SEGMENT % yolo task=segment mode=train epochs=10 data=dataset.yaml model=yolov8m-seg.pt imgsz=640 batch=8
Ultralytics YOLOv8.2.13 🚀 Python-3.10.9 torch-2.3.0 CPU (Apple M2)
engine/trainer: task=segment, mode=train, model=yolov8m-seg.pt, data=dataset.yaml, epochs=10, time=None, patience=100, batch=8, imgsz=640, save=True, save_period=-1, cache=False, de
vice=None, workers=8, project=None, name=train2, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=Fals
e, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multi_scale=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json
=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnost
ic_nms=False, classes=None, retina_masks=False, embed=None, show=False, save_frames=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, show_b
oxes=True, line_width=None, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momen
tum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015,
 hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0, auto_augment=randau
gment, erasing=0.4, crop_fraction=1.0, cfg=None, tracker=botsort.yaml, save_dir=runs/segment/train2
Overriding model.yaml nc=80 with nc=174

                   from  n    params  module                                       arguments
  0                  -1  1      1392  ultralytics.nn.modules.conv.Conv             [3, 48, 3, 2]
  1                  -1  1     41664  ultralytics.nn.modules.conv.Conv             [48, 96, 3, 2]
  2                  -1  2    111360  ultralytics.nn.modules.block.C2f             [96, 96, 2, True]
  3                  -1  1    166272  ultralytics.nn.modules.conv.Conv             [96, 192, 3, 2]
  4                  -1  4    813312  ultralytics.nn.modules.block.C2f             [192, 192, 4, True]
  5                  -1  1    664320  ultralytics.nn.modules.conv.Conv             [192, 384, 3, 2]
  6                  -1  4   3248640  ultralytics.nn.modules.block.C2f             [384, 384, 4, True]
  7                  -1  1   1991808  ultralytics.nn.modules.conv.Conv             [384, 576, 3, 2]
  8                  -1  2   3985920  ultralytics.nn.modules.block.C2f             [576, 576, 2, True]
  9                  -1  1    831168  ultralytics.nn.modules.block.SPPF            [576, 576, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  2   1993728  ultralytics.nn.modules.block.C2f             [960, 384, 2]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  2    517632  ultralytics.nn.modules.block.C2f             [576, 192, 2]
 16                  -1  1    332160  ultralytics.nn.modules.conv.Conv             [192, 192, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  2   1846272  ultralytics.nn.modules.block.C2f             [576, 384, 2]
 19                  -1  1   1327872  ultralytics.nn.modules.conv.Conv             [384, 384, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  2   4207104  ultralytics.nn.modules.block.C2f             [960, 576, 2]
 22        [15, 18, 21]  1   5259770  ultralytics.nn.modules.head.Segment          [174, 32, 192, [192, 384, 576]]
YOLOv8m-seg summary: 331 layers, 27340394 parameters, 27340378 gradients, 110.9 GFLOPs

Transferred 531/537 items from pretrained weights
```

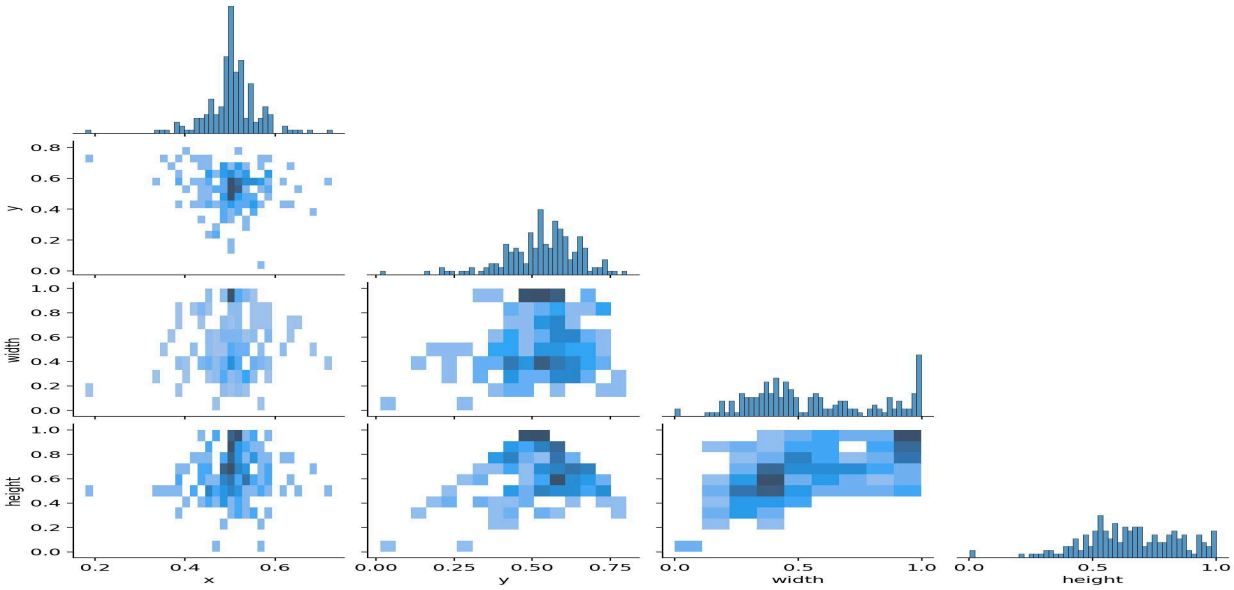

TensorBoard: model graph visualization added ✔
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs/segment/train2
Starting training for 10 epochs...
Closing dataloader mosaic
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
1/10 0G 1.476 4.388 7.805 2.144 6
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100% 2/2 [00:39<00:00, 19.94s/it]
all 24 24 0.00384 0.174 0.0054 0.00318 0.00384 0.174 0.0054 0.0027
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
2/10 0G 1.17 3.015 6.806 1.789 6 640: 100% 22/22 [10:59<00:00, 29.98s/it]
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100% 2/2 [00:39<00:00, 19.54s/it]
all 24 24 0.0033 0.174 0.0117 0.00617 0.0033 0.174 0.0117 0.00438
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
3/10 0G 0.932 2.294 6.24 1.527 6 640: 100% 22/22 [12:48<00:00, 34.94s/it]
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100%
all 24 24 0.00739 0.304 0.0293 0.016 0.00747 0.348 0.0294 0.014
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
4/10 0G 0.8242 1.93 5.924 1.36 6 640: 100% 22/22 [13:09<00:00, 35.98s/it]
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100% 2/2 [00:53<00:00, 26.91s/it]
all 24 24 0.00834 0.304 0.0278 0.0178 0.00843 0.348 0.0279 0.0191
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
5/10 0G 0.7812 1.756 5.621 1.291 6 640: 100% 22/22 [13:58<00:00, 37.74s/it]
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100% 2/2 [00:54<00:00, 27.32s/it]
all 24 24 0.0127 0.391 0.0343 0.0243 0.0127 0.391 0.0343 0.0238
Epoch GPU_mem box_loss seg_loss cls_loss dfl_loss Instances Size
6/10 0G 0.6563 1.604 5.296 1.25 6 640: 100% 22/22 [13:38<00:00, 37.18s/it]
Class Images Instances Box(P R mAP50 mAP50-95) Mask(P R mAP50 mAP50-95): 100% 2/2 [00:41<00:00, 20.76s/it]
all 24 24 0.0177 0.435 0.0532 0.0374 0.0177 0.435 0.0532 0.0355

```
      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
       6/10         0G     0.6563      1.604      5.296       1.25          6        640: 100%|██████████| 22/22 [13:38<00:00, 37.18s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:41<00:00, 20.76s/
                   all         24         24     0.0177      0.435     0.0532     0.0374     0.0177      0.435     0.0532     0.0355

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
       7/10         0G     0.6231      1.524      5.108      1.185          6        640: 100%|██████████| 22/22 [13:17<00:00, 36.23s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:43<00:00, 21.84s/
                   all         24         24      0.021      0.435     0.0835     0.0639      0.021      0.435     0.0835     0.0613

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
       8/10         0G     0.5445      1.418       4.87      1.144          6        640: 100%|██████████| 22/22 [13:47<00:00, 37.62s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:40<00:00, 20.19s/
                   all         24         24       0.02      0.435      0.075     0.0568       0.02      0.435      0.075     0.0557

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
       9/10         0G     0.5385      1.373      4.754      1.152          6        640: 100%|██████████| 22/22 [13:21<00:00, 36.43s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:44<00:00, 22.46s/
                   all         24         24      0.019      0.435     0.0904     0.0717      0.019      0.435     0.0904     0.0711

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      10/10         0G     0.5246       1.45      4.676      1.102          6        640: 100%|██████████| 22/22 [14:37<00:00, 39.89s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:55<00:00, 27.89s/
                   all         24         24     0.0216      0.522     0.0953      0.076     0.0216      0.522     0.0947     0.0762

10 epochs completed in 2.331 hours.
Optimizer stripped from runs/segment/train2/weights/last.pt, 55.0MB
Optimizer stripped from runs/segment/train2/weights/best.pt, 55.0MB

Validating runs/segment/train2/weights/best.pt...
Ultralytics YOLOv8.2.13 🚀 Python-3.10.9 torch-2.3.0 CPU (Apple M2)
YOLOv8m-seg summary (fused): 245 layers, 27323130 parameters, 0 gradients, 110.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95):  50%|████      | 1/2 [00:38<00:38, 38.72s/
```

I have trained the model for 10 epochs and it took 2.3 hours for the training to complete

Results after training looked like below for various batches





Instance Segmentation

Comparison with onnx and openvino

ONNX: ONNX serves as an intermediary format for interoperability between different deep learning frameworks. Models in ONNX format can be easily converted and deployed across various frameworks and platforms. Whereas OpenVINO's Intermediate Representation (IR) format is optimized for deployment on Intel hardware. While it can be converted from ONNX, its primary focus is on Intel architectures, offering optimized performance for CPUs, GPUs, FPGAs,

and VPUs so performance wise ONNX and OPENVINO are more open for the optimization of the code