Evtek 2022 Summer Machine Learning Internship

Round 2 Application Assignment

Raghav Rastogi

Problem definition:

A multi-class instance segmentation problem with required outputs of the class, orientation and position/spatial boundaries.

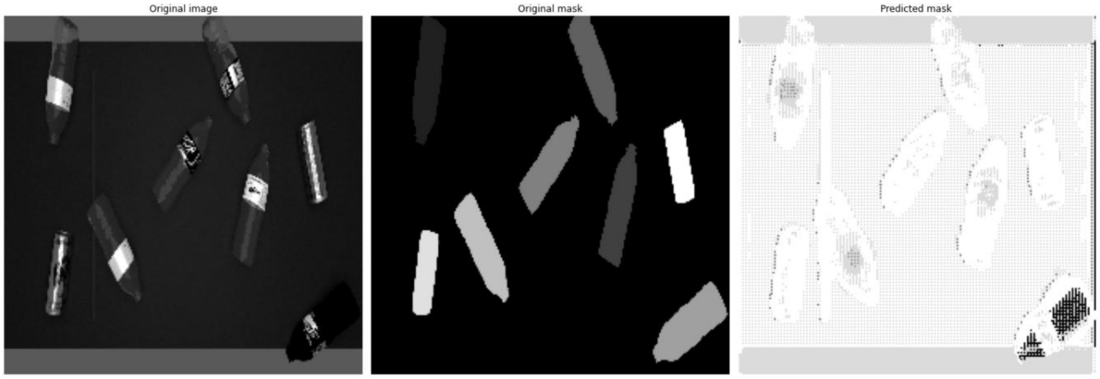
Preliminary steps:

- Understood the data - tried to understand the EXR file type and how to input it and use it in a training model. Read the documentation to get familiarity.
- Understood the requirement and type of outputs required.
- Read through the literature to see if there is already anything similar that accomplishes the task. Focus was to build a close working prototype and improve upon it.

Steps taken to understand the complexity of the problem.

- I was familiar with multi class object detection, but this is more complex since we need the spatial boundaries and orientation, thus making it multi-class instance segmentation problem.

- I started with the creation of semantic segmentation model using UNet in Pytorch to get familiar with how semantic segmentation plays a role in defining spatial boundaries. I was able to predict the spatial boundaries of the bottles and cans as shown below and thought I was done.



- Cleary I was able to create a mask given a new image, but it didn't solve the identification and classification of the objects using UNet. But I was able to understand the problem more deeply.

- I moved on to read more literature on how to solve this problem. I found out the instance segmentation in Pytorch using Mask RCNN architecture where the working model was able to locate, classify and output the image of each object and provide a bounding box coordinate with high accuracy. But this model demo only demonstrated a binary instance segmentation (object from background).

- My task was now to convert this model to make predictions on multiclass segmentation. There might be an existing architecture for the objective but given the time constraints, I decided this to be a good baseline to start.

Model Architecture

The model backbone has been referenced from the Pytorch source below:

https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

This model uses Mask R-CNN model pretrained on COCO on top of a Faster R-CNN model that predicts both bounding boxes and class scores for potential objects in the image. Mask R-CNN adds the extra branch of faster R-CNN which also predicts the segmentation mask of each instance. This is part of Detectron.

[Source: https://arxiv.org/abs/1703.06870]

Data preprocessing

Given the time constraints, I could not preprocess the data a lot. But the transform function does some augmentation by randomly flipping the images and ground truth horizontally. We can't rescale or normalize the images since it is done internally by the Mask R-CNN model.

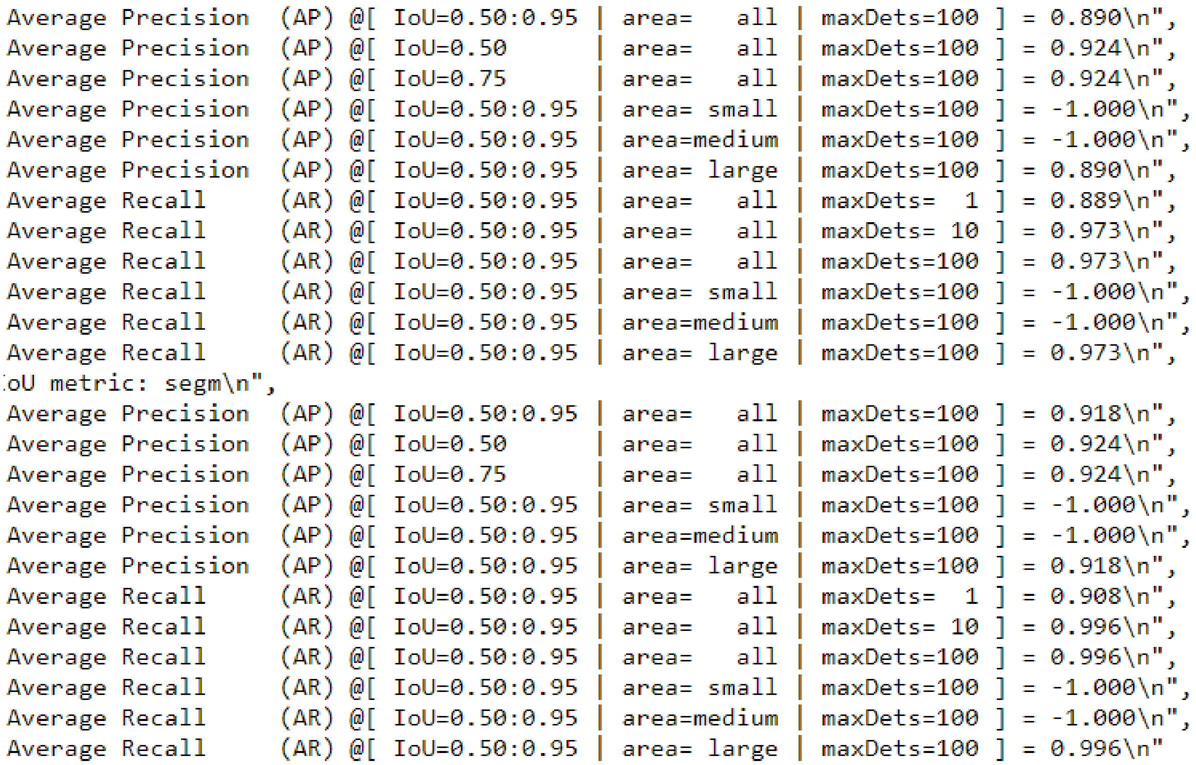
Model Evaluation

Given the computing constraints, I was able to run the epoch 10 times with all the given images as input and got the following final metrics:

```
Epoch: [9]
loss: 0.1719 (0.1784)
loss_classifier: 0.0744 (0.0770)
loss_box_reg: 0.0408 (0.0426)
loss_mask: 0.0567 (0.0574)
loss_objectness: 0.0001 (0.0002)
loss_rpn_box_reg: 0.0011 (0.0012)
```

The losses asymptotically reached the values above.

We see below that the mAP and mAR are more than 0.9 for most of the cases when the max detections, IOU thresholds and area values are varied. Following is the evaluation result of the last epoch.



```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.890\n",
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.924\n",
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.924\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.890\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.889\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.973\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.973\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.973\n",
IoU metric: segm\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.918\n",
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.924\n",
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.924\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000\n",
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.918\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.908\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.996\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.996\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000\n",
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.996\n"
```

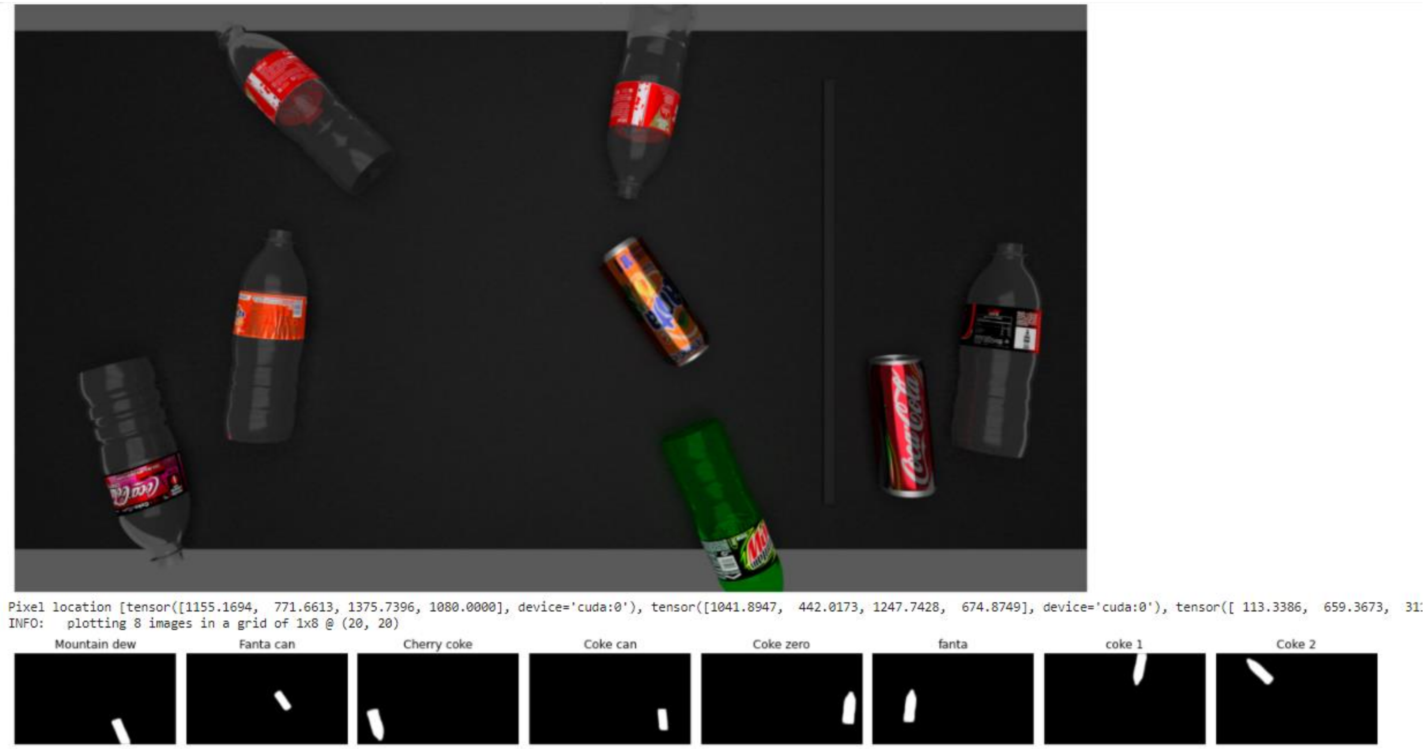
The results above show that the model can provide decent quality output given the amount of data and computing constraints. We can use this model as a baseline and incorporate more data and increase complexity to further improve the results.

Model Results:

Below is the input picture fed to the model.

The model outputs the following:

- Score: Probability of a class (0-1)
- Mask tensors ranked in decreasing value of “score” as shown below
- Class of the mask tensor
- Location of the bottles – Pixel coordinates of bounding boxes



```
Pixel location [tensor([1155.1694,  771.6613, 1375.7396, 1080.0000], device='cuda:0'), tensor([1041.8947,  442.0173, 1247.7428,  674.8749], device='cuda:0'), tensor([ 113.3386,  659.3673,  31
INFO:   plotting 8 images in a grid of 1x8 @ (20, 20)
```

Further improvements:

If we were to use the same model, we can further improve the model by fine tuning it and increase the training layer. We can further improve the model by using dectectron2 which might give better accuracy.