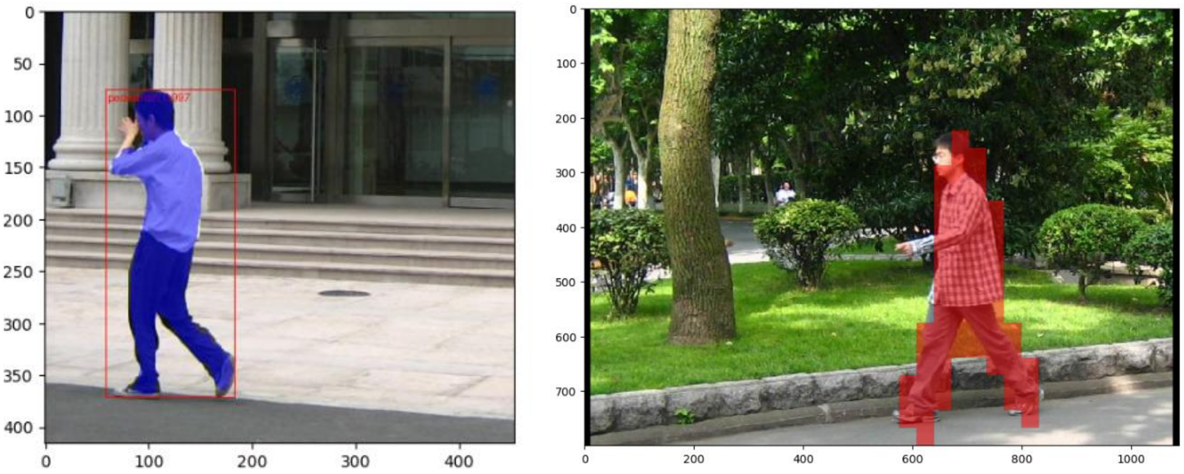
Pedestrian Detection using Instance Segmentation: A comparison of MASK R-CNN and SOLOv2

Jonathan Wang

Executive Summary: This project explores the feasibility of training a SOLO (Segmenting Objects by Locations) model on the Penn-Fudan Pedestrian Dataset to perform single-class instance segmentation on pedestrians and comparing its performance with a Mask R-CNN model. I had originally planned to train and compare a PolarMask model, but due to dependency issues with MMDetection I was forced to pivot to SOLO. My overall approach was to start with pre-trained models, modify the architecture and fine-tune them on the Penn-Fudan dataset, and evaluate them using COCO-style metrics. I found that while Mask R-CNN achieved strong results, the SOLO model struggled to perform well, which I suspect is due to a relatively small amount of training data as well as a lack of variety in the training data.

Below: Results from Mask R-CNN and SOLO, respectively.



Introduction

Object detection is a crucial task for autonomous vehicles as it helps cars identify and understand their surroundings better. Detecting pedestrians accurately is a critical aspect of this task, as it directly impacts navigation safety and decision-making in complex urban environments. Unlike static objects, pedestrians exhibit diverse poses and frequent occlusions, making them particularly challenging to detect. Traditional object detection methods often rely on bounding boxes to localize pedestrians, which can be insufficient in scenarios involving dense crowds or partial visibility. Instance segmentation, which provides pixel-level delineation of individual objects, offers a more precise and informative representation of pedestrians, enabling finer-grained reasoning about their spatial extent, motion, and interactions with other road users. This level of detail is essential for safe path planning, especially in scenarios such as crosswalks, sidewalks, or shared spaces where proximity between pedestrians and vehicles is common. In this work, I compare Mask R-CNN [1], a state-of-the-art instance segmentation model, with SOLO (Segmenting Objects by Locations) [2], a lesser-known approach to instance segmentation.

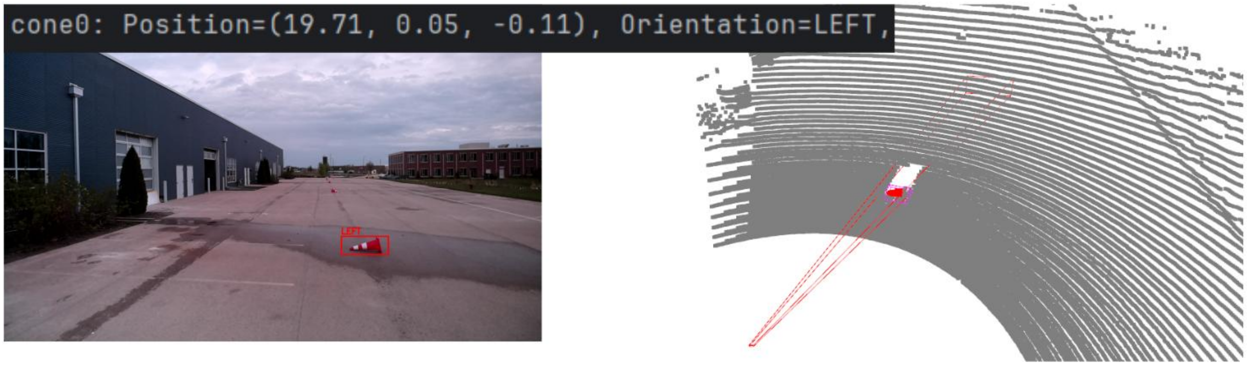
Using the Penn-Fudan dataset, a collection of images used for pedestrian detection experiments [3], I fine-tuned Mask R-CNN and SOLO models to perform pedestrian detection. After training, the Mask R-CNN model performed very well, but the SOLO model struggled to perform well, which I believe is not due to the SOLO model itself but the relatively small size of the Penn-Fudan dataset. Then, I analyze the results of the SOLO model as well as possible causes for the poor performance and discuss some takeaways from the results of the experiment.

Related Works

Object detection for autonomous vehicles: A typical approach to 3d object detection for autonomous vehicles [4] involves integrating 2d detectors with 3d point cloud data to generate 3d information for detected objects. First, a 2d object detection model generates bounding boxes for detected objects like pedestrians or vehicles. Then, by using the camera intrinsic matrix and a depth image, the 2d coordinates can be mapped to 3d in order to grab points in the 3d point cloud corresponding to the detected object. Then, a model fitting algorithm and an additional CNN can be used to refine the points into a 3d bounding box.

However, in crowded scenes, mapping 2D bounding boxes to 3D point clouds is difficult, as object segmentation becomes ambiguous due to overlapping point data. By using masks instead of boxes, instance segmentation can provide more precise 3d segmentation data, which is useful for route planning and obstacle avoidance.

Mask R-CNN: Mask R-CNN is a state-of-the-art instance segmentation model that can detect objects in an image and generate high-quality segmentation masks for those objects. As we learned in class, Mask R-CNN uses a Region Proposal Network to generate ROIs, uses ROIAlign to extract feature maps from each region and then predicts a class label, bounding box, and segmentation mask.



```
cone0: Position=(19.71, 0.05, -0.11), Orientation=LEFT,
```

An example of cone detection using YOLO and 3d point cloud data from my autonomous vehicles class

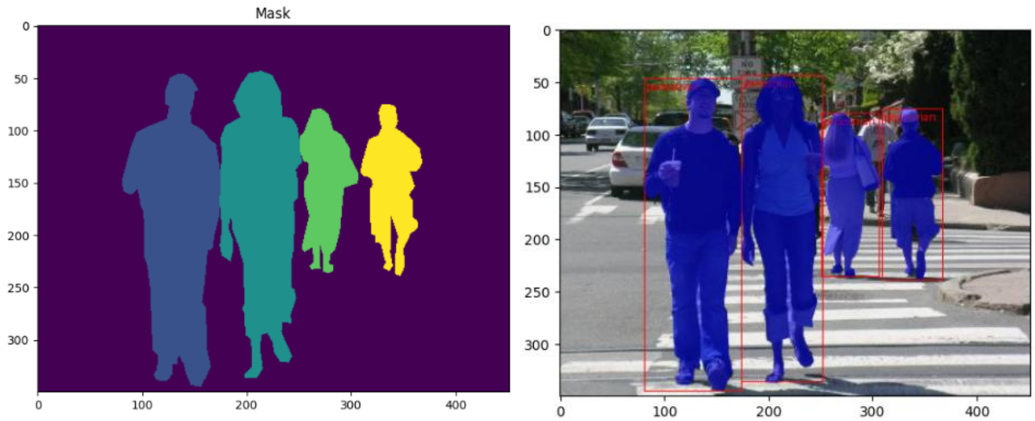
SOLO: SOLO differs from traditional instance segmentation models in that it directly segments instance masks based on full instance mask annotations instead of relying on bounding boxes or pixel-wise embeddings. It does this by dividing the image into a uniform grid and assigning each grid cell the responsibility of predicting the mask for a single object instance centered within it. This allows SOLO to achieve a simpler architecture that is more efficient while performing similarly to other instance segmentation models.

Implementation

Dataset: For this project, I chose the Penn-Fudan Pedestrian Detection dataset because it is a publicly available dataset specifically designed for pedestrian detection tasks. It consists of 170 high-resolution images taken in urban outdoor environments, with annotations for 345 instances of pedestrians. Each instance is labeled with both a bounding box and a pixel-wise segmentation mask, making it suitable for instance segmentation. Additionally, the data reflects real-world challenges like occlusions, various pedestrian poses, and background clutter, which are relevant to autonomous driving scenarios.

Another reason that I chose this dataset is because of its relatively small size compared to large-scale datasets such as COCO and Cityscapes. The small scale is ideal for prototyping instance segmentation models because it allows for faster training and experimentation.

I started by creating a custom dataset in Pytorch but later created a second custom dataset for SOLO because the model required slightly different inputs. In the custom dataset, I used the annotation mask images to generate labels and masks for each object, then generated bounding boxes based on the object masks. I also implemented data augmentation by normalizing the images and randomly flipping images for the training set. This helped improve generalization for the models since the dataset is small. For the SOLO model, I implemented additional changes to make the dataset compatible. Firstly, I resized and padded the input images and annotation masks because the model required all input images to be the same size. I also scaled the bounding boxes to match the new size. The SOLO model also used the inputs slightly differently, so I made slight modifications to my collate function to have separate lists for my images, masks, boxes, etc. instead of grouping the data element-wise.



An example annotation mask from the Penn-Fudan dataset and the corresponding masks and bounding boxes generated for the Pytorch dataset.

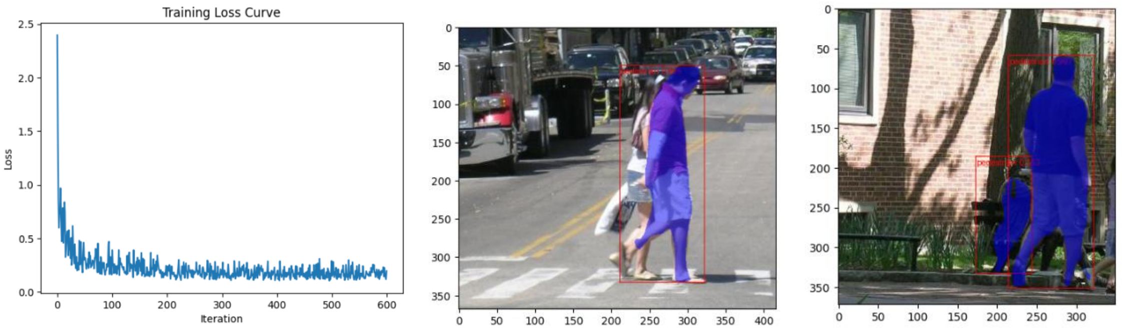
Implementing Mask R-CNN: For the Mask R-CNN model, I started with TorchVision's built-in Mask R-CNN model with a ResNet-50 and used pre-trained weights on the COCO dataset. Then I changed the model to only classify 2 classes (pedestrian and background) by replacing the box predictor and mask predictor modules. Most of the workflow for setting up my dataset, Mask R-CNN model, training, and evaluation were adapted from Pytorch's TorchVision Object Detection Finetuning Tutorial [5]**.**

Implementing SOLO: For the SOLO model, I used an open-source implementation based on an assignment from a UPenn course on machine perception 6]. The model uses the backbone from the pre-trained Mask R-CNN model as its backbone. It also uses an FPN to extract feature maps at different levels. For each level, the input image is divided into a grid (with grid size based on level) and for each grid cell, a category branch predicts a class label while a mask branch predicts a segmentation mask for the cell. Then, I use Points NMS and Matrix NMS to suppress instance predictions and mask predictions of low confidence, which results in an output of instances, labels, and masks.

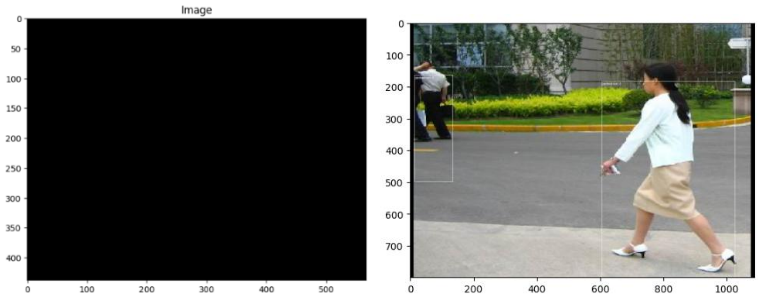
The implementation for the SOLO model used Pytorch Lightning (PL), a machine learning framework that helps to automate the training workflow. The PL trainer was helpful in making training easy, but much of my time working on this model was spent debugging and updating my code to integrate the PL framework.

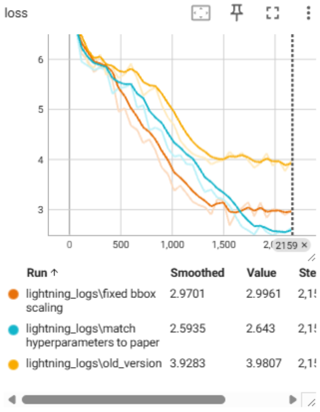
Training: Training for the Mask R-CNN went smoothly; I setup a basic training function based on what we learned in class and was able to achieve strong results after fine-tuning the learning rate and number of epochs. I used an SGD optimizer with a learning rate of 0.005, momentum of 0.9, and weight decay of 0.005. I also used a learning rate scheduler with step size of 3 and gamma of 0.1

While training the SOLO model, the first issue I ran into was that my loss wasn't changing so my model wasn't training. I started by checking the training data that was being passed in. I saw that the images being passed in were just black squares, and realized that the code in the SOLO implementation for normalizing the images expected the image data in numpy arrays, but since my dataset loaded the images as tensors, the normalization didn’t automatically cast the tensors to be floats.



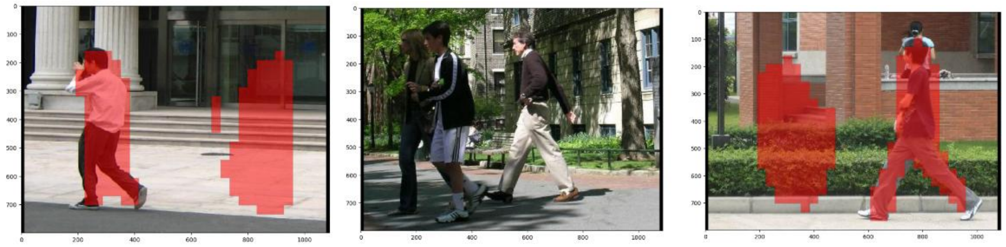
Loss Curve and Results for Mask R-CNN





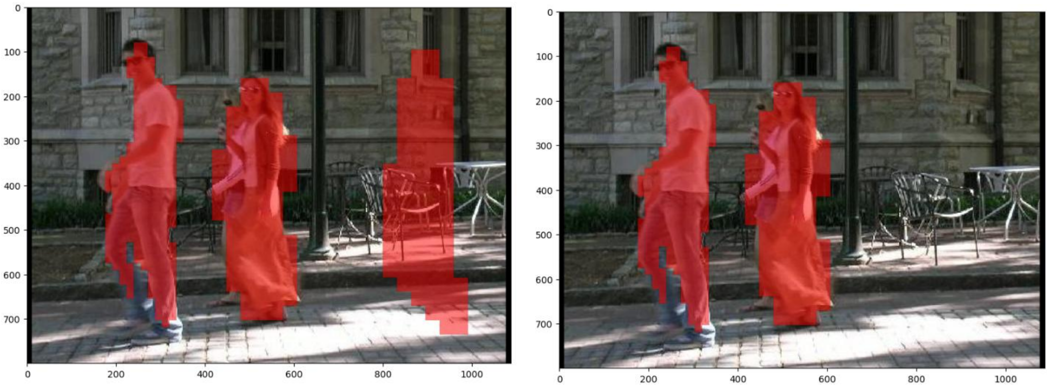
	Run ↑	Smoothed	Value	Ste
🟧	lightning_logs\fixed bbox scaling	2.9701	2.9961	2,1!
🟦	lightning_logs\match hyperparameters to paper	2.5935	2.643	2,1!
🟨	lightning_logs\old_version	3.9283	3.9807	2,1!

Some data preprocessing issues I ran into while training the SOLO model, and loss curves at different stages of debugging.



Some initial results from the SOLO model. The model would sometimes detect instances incorrectly or fail to detect instances of pedestrians, which was likely a result of the bounding boxes being misaligned.

After fixing that issue, I was able to start training the model but still saw pretty high losses. This time, I checked the masks and bounding boxes and found an issue with the way the bounding boxes were scaled during resizing. For some reason, the whole bounding box was being scaled by the horizontal width ratio, causing the box to become misaligned on the vertical axis (shown in the example above). Fixing this issue improved the model significantly, and it was able to generate decent predictions. I was able to achieve the best loss by using the hyperparameters from the original SOLO paper: 36 epochs, an initial learning rate of 0.01 which was divided by 10 at the 27th and 33th epoch, a weight decay of 0.0001, and momentum of 0.9.

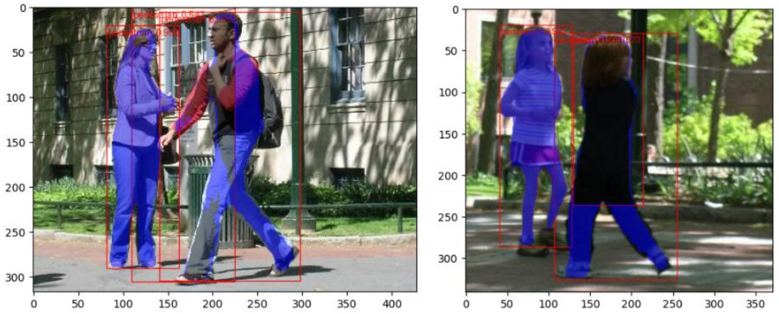


Before and after fine-tuning the confidence threshold used in postprocessing.

Something else I noticed was that the SOLO model consistently generated false positives or made the segmentation mask larger than it needed to be. By increasing the confidence threshold for NMS, I was able to remove many of the false positives. Unfortunately, I wasn't able to improve the results more than this.

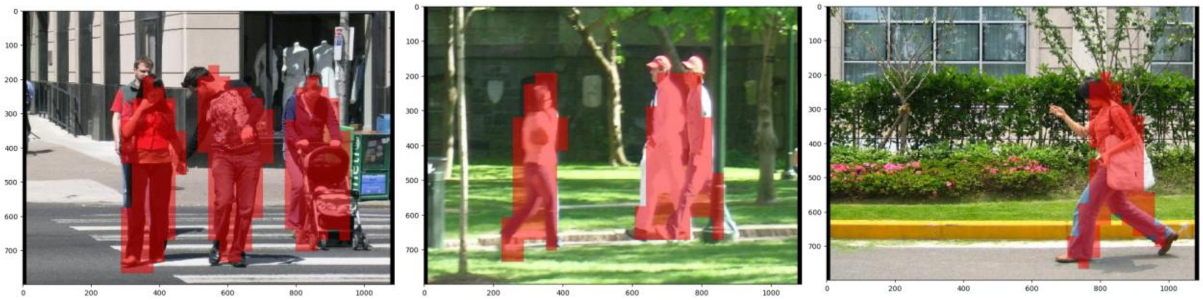
Results

Some Observations of generated results: The Mask R-CNN model was able to detect pedestrians most of the time and generated high-quality masks. I was impressed by its ability to produce accurate masks even under occlusion; however, it often failed to detect the pedestrian that was partially or fully occluded by another (see the example result from previous page). Another thing that the model seemed to struggle with was accurately generating masks when there were shadows on the detected pedestrians. However, this could probably be fixed with data augmentation techniques like ColorJitter or gamma correction.



The Mask R-CNN model runs into some trouble with shadows.

While the SOLO model was able to detect pedestrians decently well, the masks it generated were very blocky and not as accurate as the Mask R-CNN model's masks. This could have been a result of the resolution of the mask head; increasing the output resolution would probably improve the results at the cost of performance. The SOLO model also seemed to struggle more than the Mask R-CNN model with detecting clustered pedestrians and would sometimes combine them into a single instance.

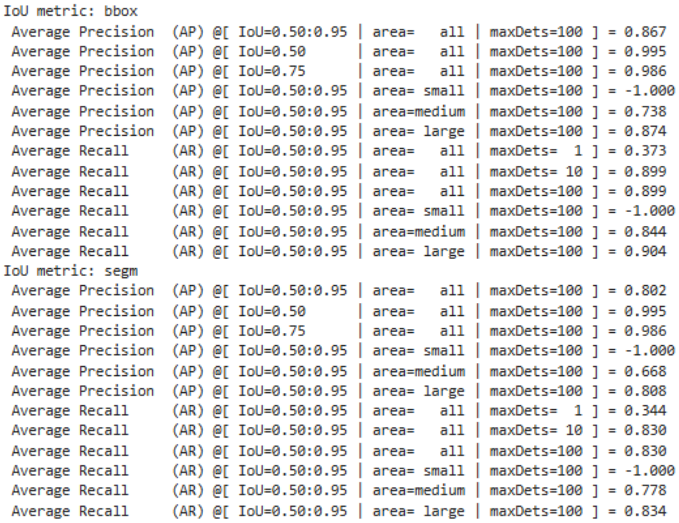


Example results from the SOLO model.

The main reason I believe the model didn't perform well is because the training data was simply too small for the model to properly train. Additionally, since the Penn-Fudan dataset focused on urban environments, there may have been a lack of variety in the data that made the model unable to generalize.

Performance Metrics: To quantitatively measure the performance of the models, I used the CocoEvaluator package from Torchvision. This allowed me to measure standard metrics such as Average Precision and Average Recall across many IoU thresholds and object sizes.

Here are the results from the Mask R-CNN model:



```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.867
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.995
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.986
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.738
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.874
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.373
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.899
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.899
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.844
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.904
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.802
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.995
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.986
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.668
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.808
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.344
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.830
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.830
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.778
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.834
```

The results indicate that the Mask R-CNN model performs well across all tasks, but performs best on larger objects. Notably, the Average Recall is very low when the max number of detections is 1, but that is expected since many images have multiple pedestrians. The results are not very surprising, given that Mask R-CNN is still one of the highest-ranking instance segmentation models.

Unfortunately, I wasn't able to generate metrics for the SOLO model (it returned zero for all precision and recall metrics.)

This is likely due to the model sometimes detecting false positives, being unable to detect overlapping pedestrians, or having low resolution masks. However, it is possible that there was a bug with my COCO evaluation code, since the results outputted by the models are formatted differently.

Discussion

One big limitation of my project is that I spent a lot of time trying to implement PolarMask, but was unable to load the model because of issues with MMDetection and related packages. The code for PolarMask is 6 years old, and I discovered while working on PolarMask that many of the functions referenced from the MMDetection framework were deprecated, renamed, or moved to different packages. After trying to downgrade packages as well as rewriting the code with the updated equivalent functions, I ultimately decided to pivot to a different instance segmentation model, but that meant that I did not have as much time to work on training the model. Here are some potential approaches for improving the SOLO model that I would implement if I had more time:

- Testing with different data: Training the model on data that is more varied or larger-scale may help clarify whether the performance gap stems from lack of data or implementation issues.
- Further data augmentation: Introducing more diverse and aggressive data augmentation techniques (e.g., random cropping, rotation, scaling) could improve the model's ability to generalize, especially given the relatively small size and uniformity of the Penn-Fudan dataset. This might help with the issue of false positives or missed detections.
- Changing the resolution of the mask head: Adjusting the resolution of the mask prediction head may enhance the model's ability to produce more accurate masks. A higher-resolution mask head could better preserve boundary details, improving both Average Precision and Average Recall for the segmentation task.
- Further debugging my COCO evaluation code: Because all the metrics returned zero for the SOLO model, it is possible that the evaluation code is not working correctly for the SOLO model. Debugging the code would help me determine if the low scores are a result of issues with the model or with the evaluation.

Overall, this project was very insightful and gave me a much deeper understanding of instance segmentation workflows, including data preparation, model architecture, training challenges, and evaluation with standardized metrics like COCO. Although I was unable to provide positive results with the SOLO model, working with the model, taught me how data issues and architecture choices can impact model performance.

The code that I used for this project can be found in my Github repository listed in the references.

References

[1] Mask R-CNN Paper: https://arxiv.org/pdf/1703.06870v3

[2] SOLO Paper: https://arxiv.org/pdf/1912.04488

[3] Penn-Fudan Database: https://www.cis.upenn.edu/~jshi/ped_html/

[4] A General Pipeline for 3D Detection of Vehicles Paper: https://arxiv.org/pdf/1803.00387

[5] TorchVision Object Detection Finetuning Tutorial: https://docs.pytorch.org/tutorials/intermediate/torchvision_tutorial.html#

[6] Github repository of SOLO implementation: https://github.com/zew013/SOLOv2--Dynamic-and-Fast-Instance-Segmentation

[7] My Github repository for this project: https://github.com/jwang501/CS-444-Final-Project