Problem Description:

Occlusion occurs when one object in a scene partially or fully blocks another from view, resulting in incomplete visual evidence for the occluded object. In real-world images, this leads to missing pixels, truncated shapes, and overlapping bounding boxes. Standard object detection and segmentation models are typically designed to predict the visible portion of an object only. In contrast, amodal detection aims to infer the full spatial extent of an object, including the regions that are hidden due to occlusion. This requires the model to learn shapes, structural regularities, and contextual cues in order to reason about object completion beyond directly observable pixels.

Occlusion poses difficulty for most detectors because they rely heavily on visible features such as edges, textures, and distinctive object parts. When these objects of interest are obscured, it leads to reduced confidence, misclassification, or complete detection failure. Overlapping objects can also cause bounding box ambiguity, particularly in crowded scenes. Additionally, many datasets contain a bias toward fully visible objects, limiting a model's robustness under heavy occlusion. Amodal detection frameworks address these limitations by explicitly modeling hidden regions, thereby improving performance in complex, real-world environments where partial visibility is common.

Summary of Codebases:

pix2gestalt Is designed to synthesize RGB images for whole objects to give various computer vision methods the capability to handle occlusions. The model uses 22 to 28 gigabytes of VRAM so running it locally is an option if the hardware allows for it, otherwise SSH or cloud computing options are better for this instance. This method also uses CLIP and taming-transformers. Installation of the model following the GitHub instructions was easy, but when attempting to run the G radio demo, the process would error out saying that there were only two GPU's found when that is the case, but somewhere in the torch library, it was not automatically adjusting to run based off of the number of GPUs that the device actually has. This led to extensive troubleshooting but ultimately failed so the next method was looked into to save time.

Sequential Amodal Segmentation Uses RGB input images to sequentially predict amodal masks for each object in that image. Environment setup was done in Python 3.10.12. Installation for the requirements ran smoothly and without any issues in the virtual environment. Initially following the instructions for the model from the GitHub site, several flags were included for what the developers started with. Following along with the instructions from GitHub, training a model took several hours and produced checkpoints every 50,000 iterations. The developers include a folder structure for how the images are layered for the training. I believe this data structure can also be followed for training other objects of interest. An initial error that was occurring was that the location for the noise files was given a placeholder as there were no actual noise files when installing the repository. This leads me to believe that the developers made the model robust for future use for any data sets that do have images with some sort of noise included. To get around the error, a dummy directory was created and pointed to in the acomloader.py script. This allowed the program to read the location without throwing a file name not found error. Doing so allowed me to run the training with the noise rate set to 0 as per the example provided in the repo. Training flags were set as follows:

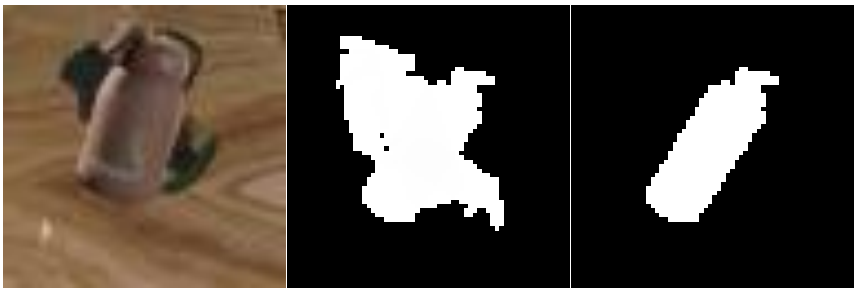
MODEL_FLAGS="--image_size 64 --num_channels 128 --class_cond False --num_res_blocks 2 --num_heads 1 --learn_sigma True --use_scale_shift_norm False --attention_resolutions 16"

DIFFUSION_FLAGS="--diffusion_steps 1000 --noise_schedule linear --rescale_learned_sigmas False --rescale_timesteps False"

TRAIN_FLAGS="--lr 1e-4 --batch_size 256"

Experimental Results:

After 150,000 iterations, the training was stopped. The sample script from the repository was then run to create an ensemble of three different segmentation masks. As far as the data set goes, each image is a 64 by 64 pixel size. This was one of the example flags the developers provided. Below is examples of the output images after savedmodel150000.pt was used as the checkpoint.



Output log for the model training shows that in the first step, model loss was 1.01. By step 100 it was 0.869 and by the 152000th step, loss was at 0.00331. Model loss could be significantly lower if model training progressed for longer resulting in better amodal output.

Analysis:

The model does succeed in detecting both objects in an image and somewhat succeeds in creating a mask for the object that is occluded. Training took 150,000 iterations and with the results above still had some struggles with creating a segmentation mask for the occluded object. The example above showed the second object behind the 1st being heavily obscured. This messed with the segmentation prediction heavily since the inferencing may not have had much to go on or the model was under trained.