A Brief Report of Occupancy Data Pipeline

1, Folder structure

The overall folder structure of module_y is shown as figure 1. “build” and “chamfer.egg-info” are self-generated folders so we can ignore them. “checkpoint” is used to stored network weights. “config” is used for configuration files, i.e. data path, occupanc size, kdtree depth, etc. “data/set/nuscenes” stores nuscenes dataset. It can be replaced by any other path as long as we set in config files for related dataset path. “dataset” stores derived class of pytorch Dataset to preprocess data for network. “extensions” stores self defined dependencies for the module. For example, campfer lib that is used to measure point cloud distance is implemented here (copy from SurroundOcc). We need to enter the folder and use “python3 setup.py install” to build and install the extension. Then “loss” folder is used for loss module definition. For example, here a dummy cross entropy loss is implemented here. In “model”, backbone, i.e. resnet, fusion model, head, and eventually occupancy model should be implemented here. We can find some dummy model for now. In “tools”, the ground truth generation and training script are stored. Run these two files we can have occupancy ground truth files and trained weights. Here, “utils” is totally empty. But it can be used for implementation of utilizations.

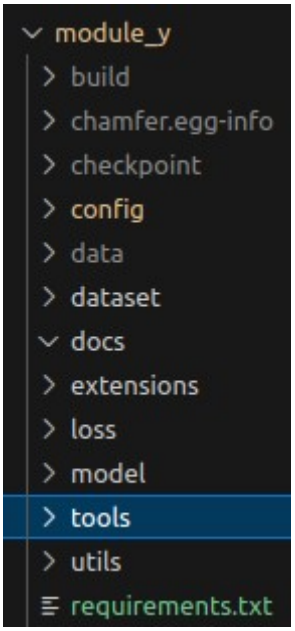

Figure 1. folder structure

2, Occupancy ground truth generation

The occupancy ground truth generation mainly refers to the open source work of SurroundOcc with removal of openmmcv dependency and some simplification. The overall pipeline is shown in figure 2. In general, lidar point cloud, 3D object list, 3D semantic segmentation, and ego pose are the main input for the pipeline. There are 32 classes of labels in original nuscenes dataset. Follow the rule in SurroundOcc, they are squeezed into 17. Firstly, point cloud are separated by in and out of bounding boxes. Points out of bounding boxes are aligned and converted into coordinate of first frame based on ego pose which is usually obtained by RTK and SLAM. Then they are stitched together. The whole data pipeline for ground truth generation from scratch will contain more content like SLAM, data synchronization, object tracking, static scene stitching, dynamic objects insertion, missing object refinement, object detection and segmentation mutual refinement, back projection into relevant coordinate, etc. But here we are using nuscenes mini dataset. So we can consider a lot of work, i.e. obtaining ego pose, point cloud segmentation, is done. Therefore, after being stitched together, the scene out of bounding boxes is converted into target frame. Then the points inside bounding boxes are inserted into the scene according to target frame. Then a selected range of target frame is remained. For each remained target frame, Possion surface reconstruction is used to generate a mesh representing the space. For the the 32 beam lidar used in nuscenes, the point cloud is super sparse, so Possion reconstruction is somehow necessary. But for lidars with more beams, i.e. the latest Hesai 1440 lidar, the point cloud can be dense already. Then voxelization is applied to have occupancy representation of the space. Finally, each voxel is assigned semantic by nearest neighbor.

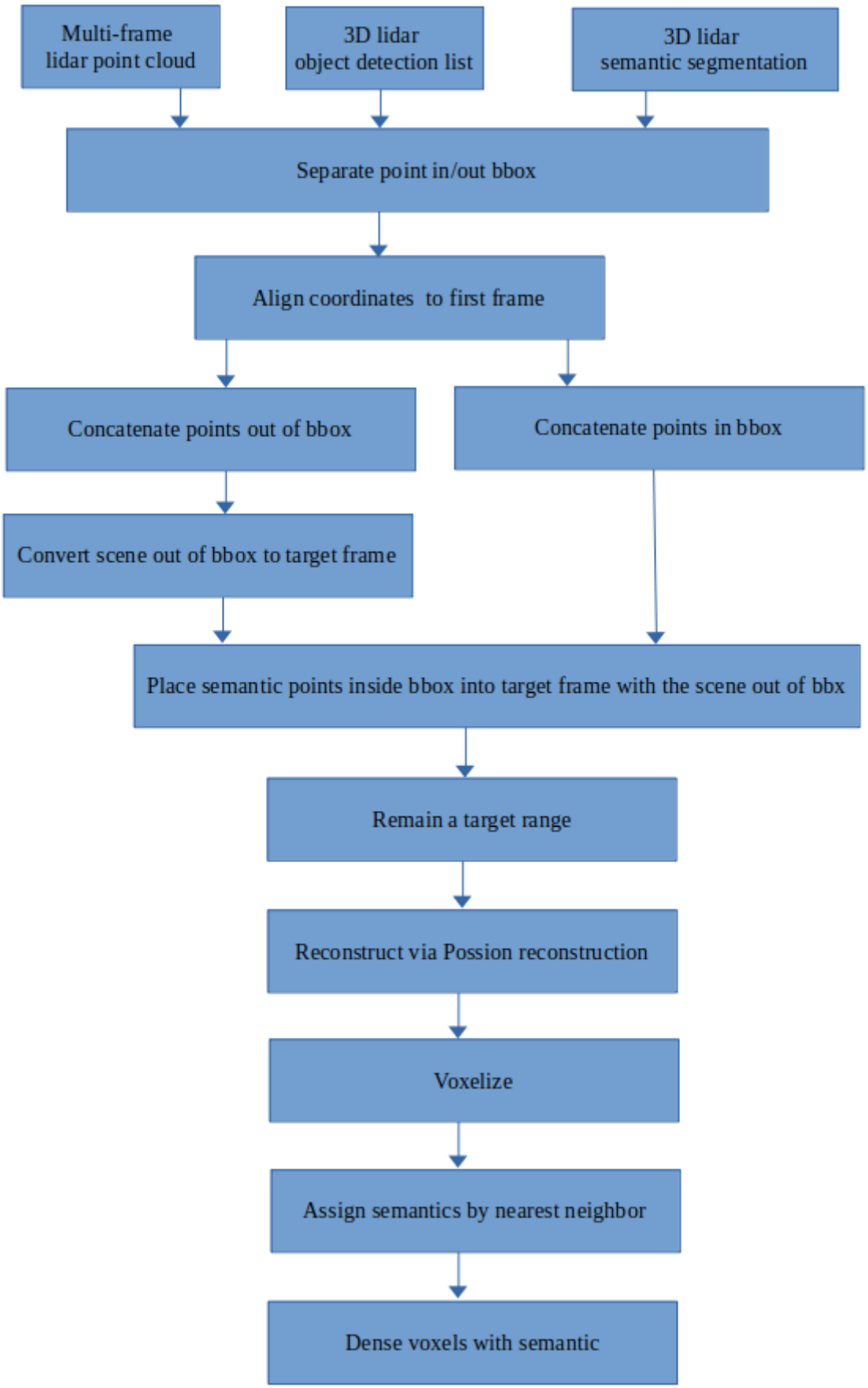

Figure 2. Pipeline of occupancy ground truth generation

3, Occupancy neural network

The data pipeline of using neural network to generate occupancy from vision is shown below in figure 3. Images from dataset are processed by resizing, augmenting, converting to tensor, and stacking in pytorch Dataset. So the images are turned into shape of [batch_size, cam_num, channel, height, width] before going into network. Then the tensor are processed by a backbone to extract features. Usually the backbone is pretrained, i.e. resnet. Then fusion part and head are designed by occupancy designer. Here, I made some dummy network to mimic the work of the entire network. Eventually the network output of shape of [batch_size, class_num, *occ_size] , goes into Loss to have cross entropy loss with ground truth which is already turned into shape of [batch_size, *occ_size] in Dataset. The weights are stored into checkpoint after training. A screenshot of training output is shown in figure 4.

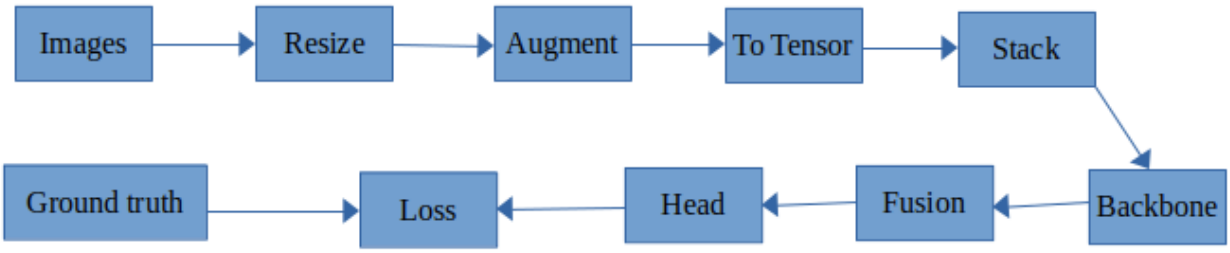

Figure 3. Data pipeline for the network

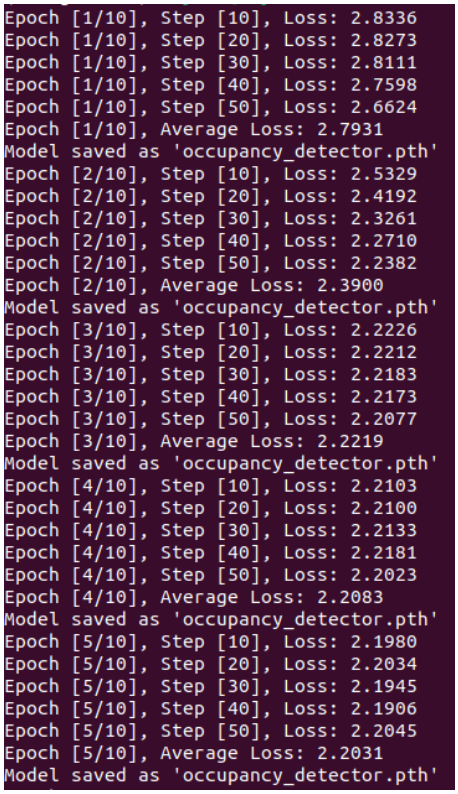

Figure 4. Screenshot of training output