COMS 4733 **Ruogu Li** **rl3318**

Homework 2

Notes you want the TAs to consider when grading.

Problem 2.1

1. Instance segmentation is the task of labeling each pixel in an image as a part of an object and separating it from other objects. The input is an image, and the output is the class label assigned to each pixel and the bounding box or mask for each object. We need pixel-level annotations, i.e., providing the corresponding class label or object bounding box or mask for every pixel in the image.

Problem 2.4

1. In convolutional neural networks, most of the convolutional layers use 3x3 kernels, and this is because 3x3 kernels have the following advantages: They have a relatively small number of parameters, which can reduce the model parameters and lower the risk of overfitting;3x3 kernels have a good local receptive field and can capture local features; By stacking multiple 3x3 convolutional layers, the same receptive field as a larger kernel can be achieved while maintaining a smaller number of parameters and lower computational cost; The stride and padding of the 3x3 kernels can be adjusted, making it possible to flexibly control the output size of the convolutional layer. The main purpose of using a 1x1 convolutional kernel before the output is to reduce the number of channels in the feature maps, thereby reducing the number of parameters, speeding up computation, and helping to prevent overfitting. The computation of the 1x1 convolutional kernel is relatively small, and it can reduce the number of channels without decreasing the performance of the network, thereby reducing the computational complexity and memory consumption.

2. In this problem, we use a task called instance segmentation. In this task, we need to assign a label to each pixel in the input image, and this label is used to distinguish between different objects and backgrounds. Therefore, the label information of each pixel needs to be included in the network output.

   In this MiniUNet model, we use 6 output channels to predict the label of each pixel. Specifically, we divide the 6 channels into two groups, with the first 3 channels representing the foreground segmentation mask and the last 3 channels

representing the background segmentation mask. During the training period, we use a cross-entropy loss function to measure the difference between these predicted masks and the true label masks. In this way, we can optimize the network weights by minimizing the loss to achieve accurate segmentation of objects and backgrounds in the image.

Therefore, six channels are needed for the output. These 6 channels represent the predicted foreground and background segmentation masks for each pixel. Each channel represents the following:
(1) Foreground Mask: This channel indicates whether each pixel belongs to the foreground. (the segmented object), with 1 indicating yes and 0 indicating no
(2) Background Mask: This channel indicates whether each pixel belongs to the background, with 1 indicating yes and 0 indicating no.
(3) Foreground Contour: This channel indicates the edge contour between the foreground and the background, which can be used to more accurately locate the segmented object.
(4) Segmentation Object 1: If there are multiple objects in the image that need to be segmented, this channel indicates the first segmented object.
(5) Segmentation Object 2: If there are multiple objects in the image that need to be segmented, this channel indicates the second segmented object.
(6) Segmentation Object 3: If there are multiple objects in the image that need to be segmented, this channel indicates the third segmented object.

We can compare the 6-channel output image with the 1-channel ground truth mask. Each pixel value in the ground truth mask represents which class the pixel belongs to, so it can be considered as a classification task. Each channel in the 6-channel output image represents the predicted probability of the corresponding pixel belonging to a certain class. Therefore, we can obtain the comparison result with the ground truth mask by calculating which class has the highest predicted probability for each pixel in the 6 channels. Specifically, we can take the maximum value of the 6-channel output image along the channel dimension, and obtain a matrix of size (H, W), which represents the most likely class for each pixel among all classes. Then we can compare this matrix with the ground truth mask to obtain the model's prediction accuracy. In the comparison process, we can use common classification evaluation metrics such as accuracy, recall, F1 score to measure the performance of the model.

Problem 2.5

1. My learning curve graph after 28 iterations shows a train loss and mIoU of 0.00 and 0.98, respectively, and a validation loss and mIoU of 0.00 and 0.94, respectively. I ran the Segmentation file on GCP, and spent a significant amount of time tuning the parameters in Segmentation.py. I found that my model achieved the best

performance when the learning rate was set to 0.0015 and the batch size for train_loader, val_loader, and test_loader was set to 1.

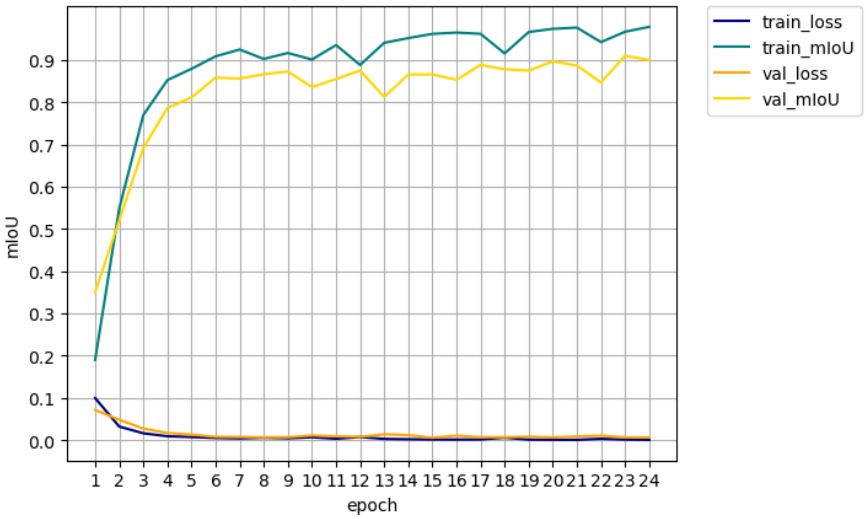

Figure 1

Problem 3.2

1. Minimum change in cost (threshold): This parameter sets the minimum change in cost that is acceptable during the ICP optimization. When the change in cost between two consecutive iterations is smaller than the threshold, the optimization process stops. A smaller threshold will result in a more precise alignment, but it may also lead to longer runtime and a higher risk of getting stuck in local minima.
2. Maximum number of iterations: This parameter sets the maximum number of iterations that the ICP algorithm will run before stopping, even if the cost is still changing. A larger number of iterations may lead to a more accurate alignment, but it also increases the runtime and the risk of overfitting to the noisy data.
3. Initial transformation: This parameter specifies the initial transformation matrix that is used to align the two point clouds. A good initial transformation can significantly improve the efficiency and accuracy of the ICP algorithm. However, if the initial transformation is far from the true transformation, it may cause the algorithm to converge to a local minimum instead of the global minimum.
4. During the debugging process, I tried different values for the maximum number of iterations within the range of 0 to 62, and for the threshold within the range of 1e-03 to 1e-09. Finally, I found that the best results were obtained when the maximum number of iterations was set to 40 and the threshold was set to 1e-07.

Implement Method

1.camera.py

(1) Compute_camera_matrix():

The function works by computing the camera's intrinsic matrix and projection matrix. The input is an instance variable of the Camera class, and the output is the camera's intrinsic matrix and projection matrix. In the function, first, FocalX, FocalY, CenterX, and CenterY are calculated based on the camera parameters, and then these parameters are used to construct the camera's intrinsic matrix. Finally, the camera's projection matrix is calculated using the function p.computeProjectionMatrixFOV(). FocalX represents the focal length in the pixel width direction, FocalY represents the focal length in the pixel height direction, and CenterX and CenterY represent the position of the center point of the pixel coordinate system in the image coordinate system.

The equation 'projection_matrix = p.computeProjectionMatrixFOV' calls the function 'computeProjectionMatrixFOV()' in the PyBullet library, which computes the camera projection matrix based on the input parameters such as field of view (FOV), aspect ratio, near plane distance, and far plane distance. The projection matrix is a 4x4 matrix used to project points from a 3D object coordinate system to a 2D image plane.

The equation 'Imageratio = self.image_size[1] / self.image_size[0]' calculates the image aspect ratio, which is the ratio of the image width to the image height. The image width and height are given in the parameter 'self.image_size', where 'self.image_size[1]' represents the image width and 'self.image_size[0]' represents the image height. The aspect ratio is calculated to be passed as input to the 'p.computeProjectionMatrixFOV()' function to compute the camera projection matrix. Here are results after running the gen_dataset.py file and show_mask():

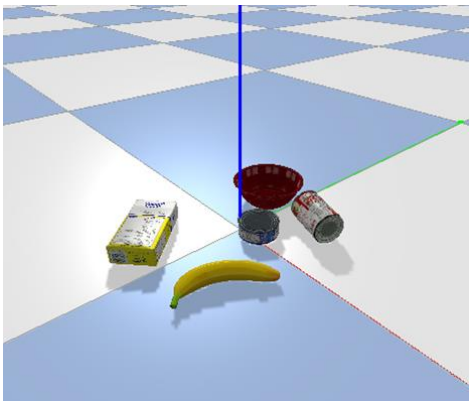

Figure 2

Figure 3

2.dataset.py

(1) def __init__(self, dataset_dir, has_gt):

The main purpose of this function is to initialize the instance variables of the RGBDataset class. The parameter dataset_dir is the directory path of the dataset, and has_gt indicates whether the dataset contains ground truth masks. In the TODO section, a data transformation pipeline is created using transforms.Compose to convert RGB images to tensors and perform normalization. mean_rgb and std_rgb store the mean and standard deviation required for normalization of RGB images. The transforms library in PyTorch is used here, specifically ToTensor() to convert RGB images to Tensor format, and Normalize(mean_rgb, std_rgb) to normalize the images.

(2) def __getitem__(self, idx):

The main purpose of this function is to provide a data reading function for the dataset.The function implementation involves reading the RGB image and ground truth mask, applying a data transformation pipeline to process the read data, and encapsulating the processed data into a sample and returning it. Specifically, in the function, the validity of the given index is first checked, then the corresponding RGB image is read, and a data transformation pipeline is used to convert it into a Tensor format. If the dataset contains ground truth mask, the corresponding mask will also be read and converted into a LongTensor format. Finally, the processed data is encapsulated into a sample and returned.

3.model.py

(1) Class MiniUNet(nn.Module):

The MiniUNet is a neural network designed for image segmentation tasks. It consists of 10 convolutional and deconvolutional layers with ReLU activation functions and a final 1x1 convolutional layer for classification. The input is an RGB image Tensor, and the output is a Tensor with the classification results. The network uses skip connections to extract high-level and low-level features for more accurate classification. The forward function executes the network's forward propagation process, applying ReLU, max pooling, deconvolution, and skip-connection operations.

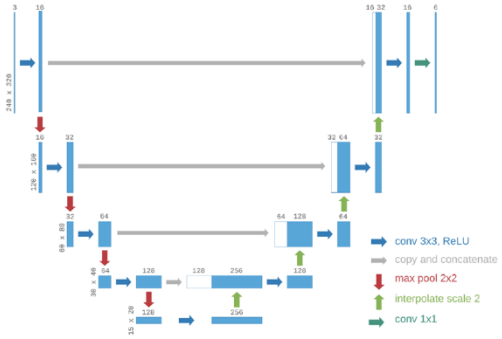

Figure 4

3.segmentation.py

(1) train() & val():

The train() method is used to train a neural network model. For each batch of data, it first retrieves the input data and corresponding target output data from the DataLoader. Then, it feeds the input data into the model for forward propagation to obtain the output result. The difference between the output and target output is used to compute the loss value, and the gradients of each parameter are computed using the backpropagation algorithm. The optimizer algorithm is then used to adjust the values of each parameter to minimize the loss value. During the training process, the train() method also calculates the average loss and Intersection over Union (IoU) score for each batch.

The val() method is used to validate the neural network model. Similar to the train() method, it calculates the average loss and IoU score for each batch, but it does not perform backpropagation to update the parameters. Instead, it only performs forward propagation to compute the output result and the difference between the output and target output.

Both methods use the cross-entropy loss function to compute the difference between the predicted output and target output. In the train() method, to prevent overfitting, the model's weights are saved at the end of each epoch, and the mIoU score on the validation set is recorded. Later, in the subsequent training, the best model is selected based on the comparison of the mIoU score on the validation set. The results of the segmentation run are shown in Problem 2.5.

4.icp.py

(1) obj_mesh2pts:

The obj_mesh2pts() method in icp.py file is used to convert an obj file to a point cloud. This method first loads the obj file and then converts it to a point cloud by sampling the obj file. The obj file can be transformed by passing a transformation matrix to better fit the scene. These point clouds can be used for subsequent ICP (Iterative Closest Point) algorithm to calculate the relative pose between two scenes.

The meaning of this statement is that it first obtains the path of the corresponding obj file based on the passed obj_id, loads the obj file using the load() method of the trimesh library, and then transforms the obj file according to the passed transformation matrix. Then, the sample_surface() method of the trimesh library is used to sample the transformed obj file and obtain a certain number of point clouds. Finally, the method returns the point cloud array pts.

(2) gen_obj_depth:

The function obj_depth2pts() in icp.py is used to convert a depth image of a specific object into a point cloud. The function first makes a copy of the input depth image,

and then selectively sets the depth values to zero based on the object ID or mask. If an object ID is provided, then the function sets the depth values to zero for all pixels in the image that do not belong to the specified object. If an object ID is not provided, then the function sets the depth values to zero for all pixels in the image that do not belong to any of the five objects of interest. Finally, the function returns the modified depth image as a numpy array.

(3) align_pts():

The align_pts() function in icp.py works to align two point clouds, pts_a and pts_b, for subsequent pose estimation. This function uses Procrustes transformation and ICP (Iterative Closest Point) algorithm. First, Procrustes transformation is applied to align the two point clouds and obtain an initial transformation matrix. Then, ICP algorithm is used to iteratively align the two point clouds and obtain a more accurate transformation matrix. Finally, the cost of ICP is checked against a threshold value. If the cost is above the threshold, a warning message is printed indicating that the alignment may not be accurate. If pts_a is empty, the function returns None. If there is a linear algebra error during computation, the function also returns None.

(4) estimate_pose ():

The function estimate_pose() in icp.py estimates the pose of each object in the scene by aligning the projected points and the sample points of each object.

The function first initializes an empty list list_obj_pose, and then iterates through the five objects in the scene. For each object, the function uses obj_depth2pts() and obj_mesh2pts() to obtain the projected points and the sample points, respectively. Then, align_pts() is used to align the two sets of points and estimate the transformation matrix.

The transformation matrix is then appended to the list list_obj_pose, or None is appended if the pose estimation fails. Finally, the function returns the list of transformation matrices.

The parameters max_iterations and threshold control the maximum number of iterations and the convergence threshold of the alignment process.

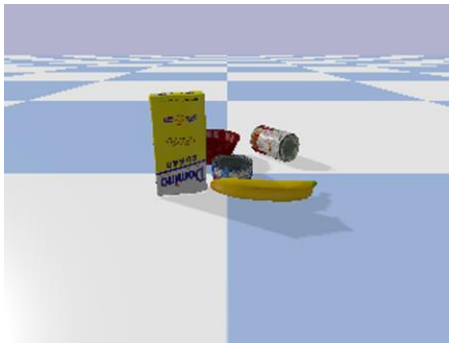

Figure 5

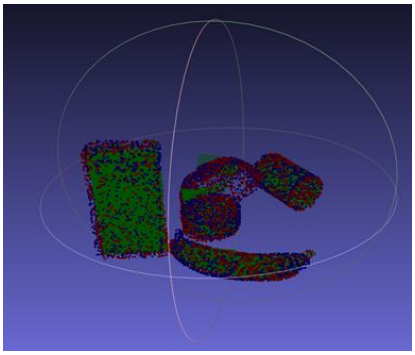

Figure 6

(5) main ():

The main() function is responsible for estimating the pose for each scene in the validation or test set and outputting the results. Firstly, it selects whether to process the validation or test set based on the input arguments (args). Then, it sets up the camera parameters and creates folders to store the output files. For each scene, it reads the depth image, mask image, and view matrix. For each scene in the validation set, it estimates the pose using the estimate_pose() function based on the depth image and predicted or ground truth mask image, and exports the result as a PLY file and a text file. For each scene in the test set, it estimates the pose using the depth image and predicted mask image, and exports the result as a PLY file and a text file. Finally, it calls the estimate_pose(), export_gt_ply(), export_pred_ply(), and save_pose() functions to perform pose estimation, output the ground truth point cloud, export the transformed point cloud, and save the estimated poses, respectively. The point cloud images are shown in the figure below.

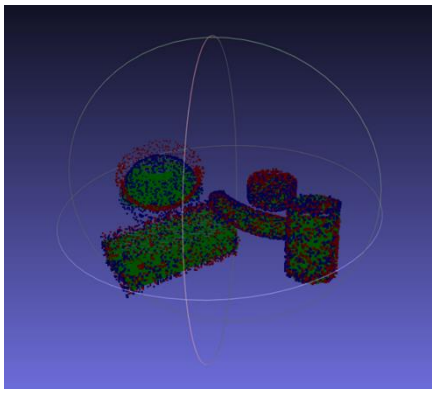

Figure 7

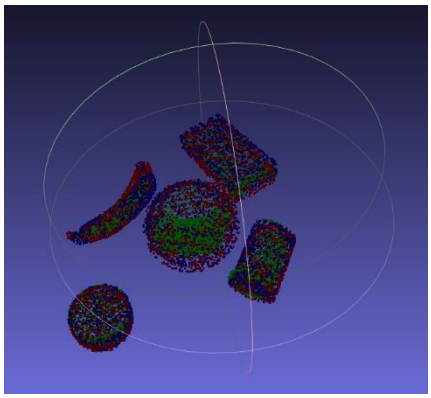

Figure 8