CS 498: Assignment 4: Point Cloud Segmentation

Hongqing Liu, hl85

Apr 15, 2023

Submission

In this assignment, you will implement 3D point cloud segmentation using neural networks. The starter code consists of an iPython notebook "mp4.ipynb" which can be opened and run on Google colab. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: BBX6NE). We recommend you add your answers to the latex template files we provided. More details on what to report are in the provided notebook.

Reminder: Your submission should include your report pdf and filled out mp4.ipynb.

Segmentation

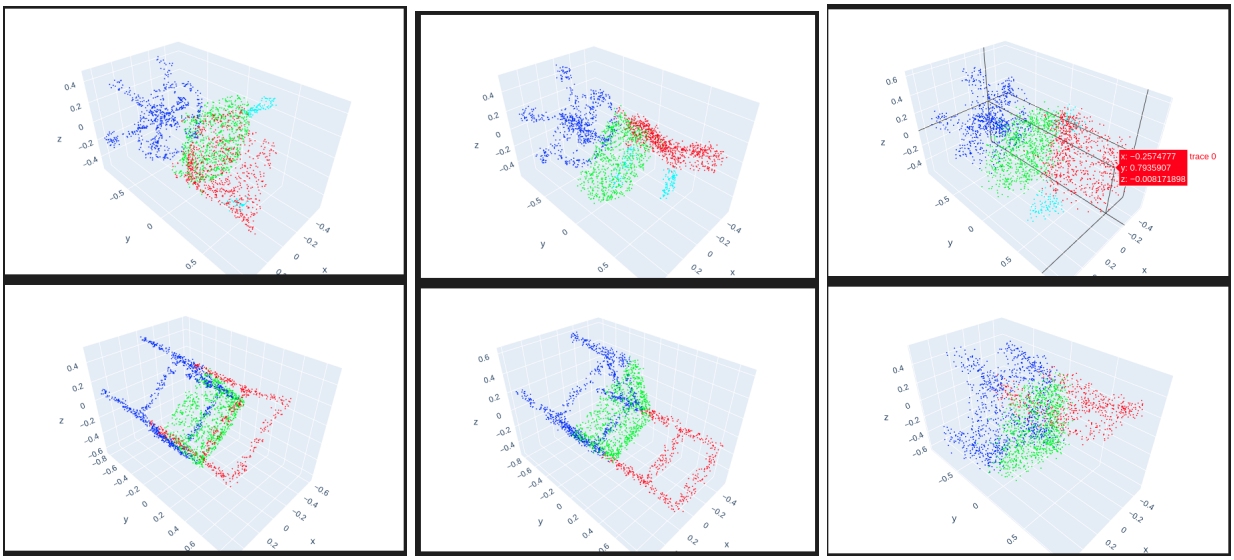
Question 1 (Data Loading and Augmentation)[1 pt]: We provide code that loads the segmentation data. In this part you will need to perform data augmentation on the loaded data within the "ShapeNet-Dataset" class. In particular you should rotate the point cloud around the y-axis by a random angle, and also add some random Gaussian noise to the 3D coordinates. You should experiment with different noise levels and report the results in your pdf.

Answer: The codes are shown as follows, firstly I rotate the point cloud around y-axis for a random angle, then I add some noise.

```
if self.data_augmentation: # Perform random rotation and jittering
    # Your code
    # ------------------------
    angle_range = np.pi
    rotation_angle = np.random.uniform(-angle_range, angle_range)
    rotation_matrix = np.array([[np.cos(rotation_angle), 0, np.sin(rotation_angle)],
                                [0, 1, 0],
                                [-np.sin(rotation_angle), 0, np.cos(rotation_angle)]])
    point_set = np.dot(point_set, rotation_matrix)

    jitter_range = 0.01
    jitter = np.random.normal(0, jitter_range, size=point_set.shape)
    point_set += jitter
```

The following results are the visualization for augmentation data with noise 0.01, 0.02 and 0.05.



It seems like that noise equal to 0.01 is the best noise for data augmentation.

Question 2 (Simple Baseline) [2 pts]: In this part you will be modifying "simple_model" and "simple_predict" to implement a simple chair part segmentor. For each point cloud, you should segment it into thirds and predict the top third as the chair back, the middle third as the chair seat, and the bottom third as the chair legs. Determine which integer label corresponds to each class based on the visualizations from the previous question. You can run the evaluation code (from the next question) with your simple baseline and see its mean average precision which we provide - it should be around 0.54.

Answer: Here I sort points y value in each point cloud and put the top third as class 0 while the bottom third as class 2. The pred seg is in shape of $(16, 4, 2500)$. The codes are shown as follows:

```
third = int(n_pts / 3)
bottom_third = int(2 * n_pts / 3)

for batch in range(batch_size):
  y_values = pts_batch[batch,1,:].to(device)
  sorted_indices = torch.argsort(y_values, descending = True).to(device) # 从大到小排列
  top_third_y_value = y_values[sorted_indices[third]].to(device)
  bottom_third_y_value = y_values[sorted_indices[bottom_third]].to(device)
  pred_seg[batch, 0, y_values>=top_third_y_value] = 1
  pred_seg[batch, 1, ((y_values<top_third_y_value) & (y_values>=bottom_third_y_value))] = 1
  pred_seg[batch, 2, y_values<bottom_third_y_value] = 1
```

The gt seg's shape is $(16, 2500)$. The code is shown as follows:

```
for batch in range(batch_size):
  # gt_seg[batch, 0, label[batch]==0] = 1
  # gt_seg[batch, 1, label[batch]==1] = 1
  # gt_seg[batch, 2, label[batch]==2] = 1
  # gt_seg[batch, 3, label[batch]==3] = 1

  gt_seg[batch, label[batch]==0] = 0
  gt_seg[batch, label[batch]==1] = 1
  gt_seg[batch, label[batch]==2] = 2
  gt_seg[batch, label[batch]==3] = 3
```

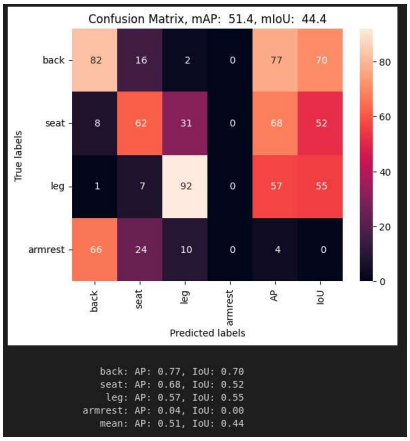
When I write this assignment on google colab, the simple prediction function takes a lot of time, sometimes even 40 minutes. In the beginning, I think the problem is that cpu has a low speed to load and process the data, so I put the computation to gpu. Turns out that the real problem is that the pogram has to load the data from google drive every time I restart the kernel, and my internet is slow.

Question 3 (Evaluation Metrics) [1 pts]: We must evaluate the quality of our predictions. In this part you will fill in "compute_confusion_matrix". You should write code to compute the confusion matrix as well as IoU for the predicted segmentation when compared to ground truth. We provide code for visualizing the computed values as well as computing mean average precision.

Answer: The confusion matrix is an eye matrix with n classes dimeension while the IOU imetrics can be computed as the intersection / union. The codes and results are shown as follow, the mAP is 51.4 while the mIoU is 44.4.

```
def compute_confusion_matrix(gts, preds): # (2656, 2500), (2656, 4, 2500)
    # print("gtss shape: ", gts.shape, "preds shape:", preds.shape)

    n_classes = preds.shape[1]
    IoU = np.zeros((n_classes))
    conf = np.eye(n_classes)
    # Your code
    # ----------------------
    for i in range(n_classes):
      for j in range(n_classes):
        conf[i, j] = np.sum(np.logical_and(gts == i, preds[:,j,:] == preds.max(axis = 1)))
    intersection = np.diag(conf)
    union = np.sum(conf, axis=0) + np.sum(conf, axis=1) - np.diag(conf)
    IoU = intersection / union
    # ----------------------
    return IoU, conf
```



Question 4 (Loss Function) [2 pt]: To train a model we need a loss function. In this part you will fill in "cross_entropy_criterion" with your implementation of the weighted cross entropy between predicted class probabilities and ground truth class labels.

Answer: The cross entropy loss can be written as follows:

```
def cross_entropy_criterion(predictions, labels, weights):
    #predictions: (16,4,2500)
    # labels: (16,2500) 0,1,2,3
    # Your code
    # ----------------------
    predictions = predictions.transpose(1,2)
    predictions = predictions.contiguous().view(-1, predictions.size(2))
    # print(predictions.is_contiguous())
    # print(predictions.shape) # ([40000, 4])
    log_softmax = F.log_softmax(predictions, dim=1) # (40000, 4)
    weights = torch.tensor(weights).to(device)
    # print(weights.shape) # (1,4)
    predictions_value = torch.mul(weights, log_softmax)
    # print(predictions_value.shape) # (40000, 4)

    labels = labels.view(1, -1)
    # print("labels:", labels[:5])
    element = torch.arange(0, labels.shape[1]).to(device)
    # print("element:", element[:5])
    labels = labels + element * predictions.shape[1]
    # print("labels:", labels[:5])
    loss = -1 * torch.take(predictions_value, labels)
    # print("loss:", loss)
    # ----------------------
    return loss.mean()
```

Firstly, I transpose the prediction matrix as (16,2500,4) and view it as (-1, 4). Here I learn how to use contiguous to make the elements stored in a contiguous block of memory. After that, I log softmax it in its second dimension and multiply it with corresponding weights.

To choose the write log softmax value to calculate the loss, I use torch.take() function. The indices of element I choose is generated as follows.

Firstly, I reshape labels matrix into (1, 40000), each value are corresponding to the indices of the weighted log softmax we want to choose. For instance, if the first element is 2, it means we want to choose weighted log softmax[0][2] to calculate loss and if the second element is 1, it means we want to choose weighted log softmax[1][1], which is also weighted log softmax[$1 * 4 + 1$] if we reshape the log softmax matrix to one row to calculate loss.

In that case, I use

$element = torch.arange(0, labels.shape[1]).to(device)$
$labels = labels + element * predictions.shape[1]$
to choose the write labels for calculating loss.

Question 5 (Training Loop) [2 pt]: In this part you will implement the stochastic gradient descent training loop in PyTorch, modifying "train". We provide code to validate a trained model and a skeleton for training.

Answer: The train part is pretty straight forward, firstly I put train input and train labels to gpu, secondly I zero grad the optimizer, thirdly I put the train input into the model and calculate its criterion. After that, I calculate the network's backward by $loss.backward()$ and let the optimizer train the model by $optimizer.step()$.

The validation part only calculate the loss of the model using val images and val labels, without any training. In that case, we put this part into $model.eval()$.

Question 6 (Model Definition and Training) [3 pt + 1 bonus point]: Implement a segmentation network based on the PointNet architecture (`https://arxiv.org/pdf/1612.00593.pdf`, see Fig. 2). Train your models with the code you wrote for Question 5, and report results following the instructions in the notebook.

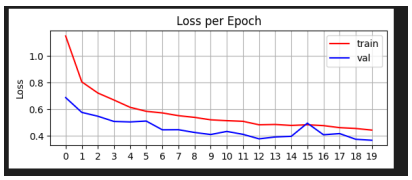
As mentioned in the notebook, you may ignore the input-transform and feature-transform portions of PointNet and receive full credit. However, for one bonus point, you can try implementing these transforms. Details about the T-Net architecture and regularization loss can be found in the supplementary material of the paper. If you do implement the transforms, please report results with and without using them in your network. (Bonus 1 pt)

Answer: In this part, I will introduce the network with 3 parts. The first part is input transform and feature transform, here I utilize two models to integrate the input transform and feature transform network. The firt model is combined with three conv1d networks, followed by batch normalization and ReLU activation functions. The second model is combied with three linear model, followed by batch normalization and ReLU activation functions. The different between input transform and featrue transform is that the former one has 3 input channel in the first conv layer while the second one has 64 input channel for the first conv layer.

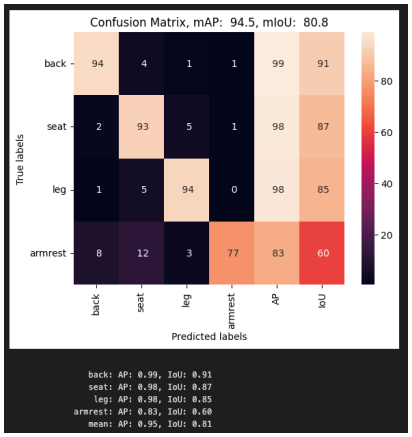
The second part is named as point net feat, it is the part from the beginning to the concatation of global and local features. The integration primarily consists of three shared MLP models, which have been substituted with Conv1d models. t utilizes input transform before the first MLP and feature transform after the first MLP and it output the concatation of global and local feature. The output dimension is 1088.

The third part outputs per point scores. This part shows the whole network, it is combined by point net feat network and a segmentation network. The segmentation is integrated by 4 conv network, followed by batch normalization and ReLU.

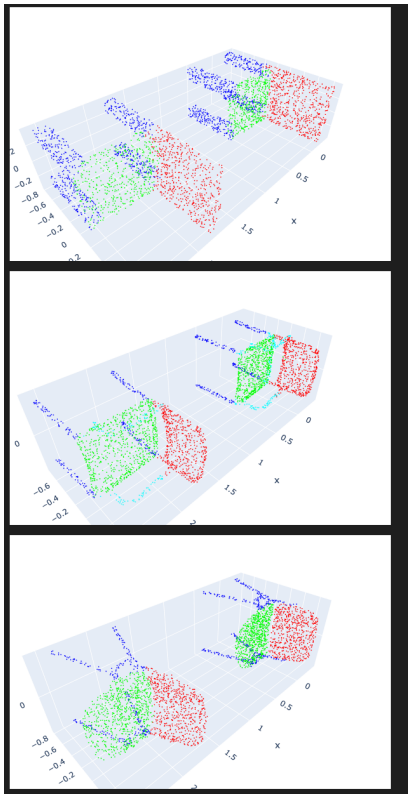
The loss per epoch is shown as follows, after 20 epoches, the training loss is 0.445 and the val loss is 0.376.



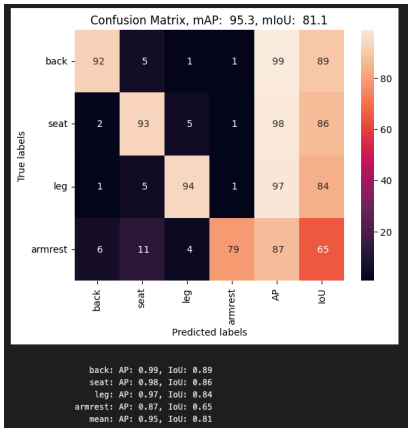
The confusion matrix of val set is shown as follows, the mAP is 94.5 and the mIoU is 80.8.



The visulization of 3 random result in val-dataset is shown as follows:

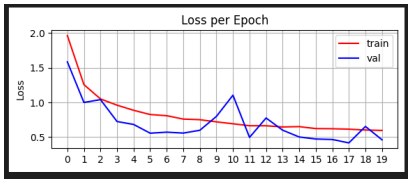


The confusion matrix of test set is shown as follows, the mAP is 95.3 and the mIoU is 81.1.

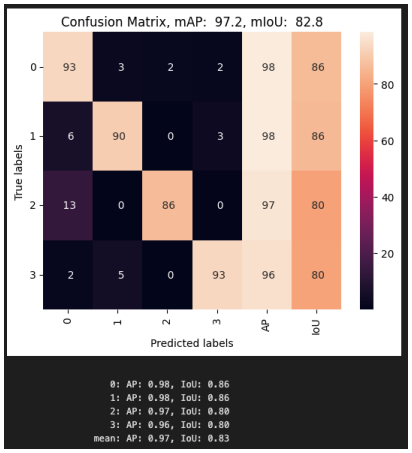


Question 7 (Another Object Category) [3 pt]: So far we have trained a model for chair part segmentation. However, there are many other object categories available in our dataset. Choose whichever one you like, extract the corresponding data, train a new model for that object category, and report the results. Make sure to include, training plots, final accuracy metrics, and a few visualizations of your model predictions. Also, describe what changes you made, if any, from your chair segmentation algorithm.

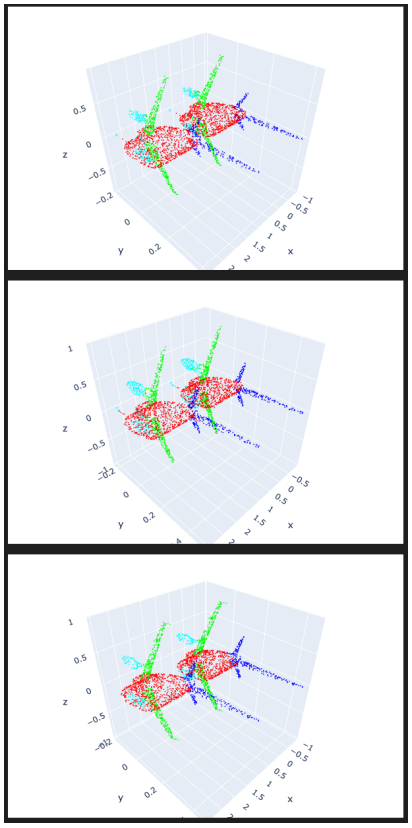
Answer: Here I use the airplane dataset, it has 4 classes. After 20 epoches of training, the results are shown as follows: The loss per epoch is shown as follows, the training loss is 0.596 and the val loss is 0.461.



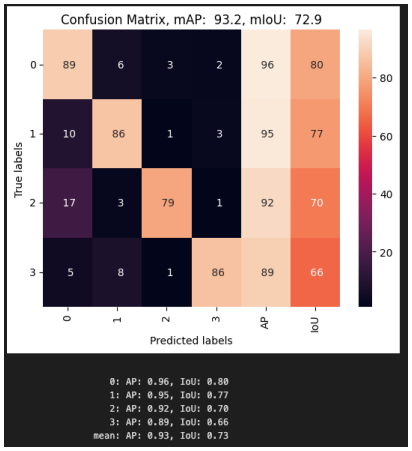
The confusion matrix of val set is shown as follows, the mAP is 97.2 and the mIoU is 82.8.



The visulization of 3 random result in val-dataset is shown as follows:



The confusion matrix of test set is shown as follows, the mAP is 93.2 and the mIoU is 72.9.



PointNet++ (Bonus 3pt)

PointNet achieved excellent results when it was published in 2016, but many works have built upon and improved the architecture since then. For example, PointNet++ (`https://arxiv.org/pdf/1706.02413.pdf`) was developed one year later and introduced hierarchical grouping to learn better features. Implement PointNet++ or another more recent point cloud segmentation architecture and evaluate its performance on our dataset.