Advance in 3D vision HW1: Single View to 3D

311553060

周庭韵

1. Loss function

To calculate the loss function for voxels, I used binary cross entropy loss provided by pytorch library.

For point clouds, I calculated chamfer_distance between source point clouds and target point clouds for chamfer loss.

1. For each point in source point cloud, find the nearest neighbor in target point cloud
2. Calculate the distance between each point in source point cloud and its nearest neighbor in target point cloud
3. Repeat the same process for each point in target point cloud to find its nearest neighbor in source point cloud and calculate the distances
4. Sum up all distances from step2 and step3 to get the chamfer loss

```
# loss
loss = nn.BCELoss()
# implement some loss for binary voxel grids
loss_vox = loss(voxel_src.squeeze(), voxel_tgt.float())
return loss_vox

def chamfer_loss(point_cloud_src,point_cloud_tgt):
    # loss_chamfer, _ = chamfer_distance(point_cloud_tgt, point_cloud_src)

    # implement chamfer loss from scratch
    dist_matrix = torch.cdist(point_cloud_src, point_cloud_tgt)  # 计算成对距离

    chamfer_dist = torch.min(dist_matrix, dim=2)[0].sum(dim=1)
    chamfer_dist_R = torch.min(dist_matrix, dim=1)[0].sum(dim=1)

    chamfer_loss = torch.mean(chamfer_dist_R, dim=0) + torch.mean(chamfer_dist_R, dim=0)

    loss_chamfer = chamfer_loss.mean()

    return loss_chamfer
```

2. Model Architecture

2-1. Encoder

For encoder, I used pretrained model without linear layer from torchvision.models, such as Resnet18, Resnet50, Resnet101

```
class SingleViewto3D(nn.Module):
    def __init__(self, args):
        super(SingleViewto3D, self).__init__()
        self.device = "cuda"
        self.image_encoder = torchvision.models.__dict__[args.arch](pretrained=True)
        self.current = torch.nn.Sequential(*list(list(self.image_encoder.children())[:-1]))
        self.current2 = torch.nn.Sequential(torch.nn.AdaptiveAvgPool2d((1,1)),
        torch.nn.Flatten(),torch.nn.Linear(in=512, out=512), torch.nn.ReLU(),torch.nn.Linear(in=512, out=512), torch.nn.ReLU(),torch.nn.Linear(in=512, out=512))
        self.cnum=512 # output feature dimension
        # define decoder
        if args.dtype == "voxel":
            # TODO:
            self.n_voxels = pow(32,3)
            self.decoder = SimpleVoxelDecoder(in_size = 512, latent_size = self.latent_size)
        elif args.dtype == "point":
            # TODO:
            self.n_points = args.n_points
            self.decoder = PointDecoder(in_size = 512, latent_size = self.latent_size)
        elif args.dtype == "mesh":
            # TODO:
            # raise NotImplemented
            self.n_points = 2562
            # initialize ellipsoid
            mesh = pytorch3d.utils.ico_sphere(level=3, device=self.device)
            verts = mesh.verts_packed()
            N = verts.shape[0]
            center = torch.mean(verts, dim=0)
            scale = torch.max(torch.abs(verts - center), dim=0)[0]
            verts = (verts - center) / scale
            self.ellipsoid = {"verts": [verts.cpu().tolist()],
                              "edges": [mesh.edges_packed().cpu().tolist()],
                              "tri": True}
            # define layers
            self.graph_layers = [[],[],[],[],[]]
            for i in range(3):
                self.graph_layers[i] = [pytorch3d.utils.ellipsoid.utils.get_pool_1ring(i, self.ellipsoid)]
            self.graph_layers = sum(self.graph_layers, [])
            self.graph_unpool = [pytorch3d.utils.ellipsoid.utils.get_unpool_info(1, self.ellipsoid, self.graph_layers[1:])]
            self.graph_unpool = sum(self.graph_unpool, [])
            self.graph_last = [pytorch3d.utils.ellipsoid.utils.get_pool_1ring(3, self.ellipsoid, self.graph_layers[2:])]
            self.graph_last = sum(self.graph_last, [])
            self.cnum = [512, 256, 128, 64]
            self.decoder = GraphConvDecoder(in_size = 512, latent_size = self.latent_size, n_blocks = 3, hidden_dim = 256, graph_layers = self.graph_layers, graph_unpool = self.graph_unpool, graph_last = self.graph_last, device = self.device)
```

```
    def forward(self, x):
        # TODO:
        # print("shape: ", x.shape)
        if self.training:
            x = x[:,0,:,:,:]
        else:
            x = x[:,1,:,:,:]
        # print("shape: ", x.shape)
        x = self.current(x)
        # print("shape: ", x.shape)
        x = self.current2(x)
        # print("shape: ", x.shape)
        # print("latent size: ", self.latent_size)
        # print("device: ", self.device)
        x = self.decoder(x)
        # print("shape: ", x.shape)
        return x
```

2-2. Decoder

For the decoder used for point cloud, called PointDecoder, the detail network architecture is shown below.

It consist of several linear layers, ReLU activation function and Tanh activation function.

After the encoded feature go through decoder, it would finally reshape to [B, N, 3] to get the final prediction, where B, N are batch size and number of points, respectively.

```
class PointDecoder(nn.Module):
    def __init__(self, in_size, latent_size):
        super(PointDecoder, self).__init__()
        self.size = latent_size
        self.fc1 = nn.Linear(in_size, 1024)
        self.fc2 = nn.Linear(1024, 1024)
        self.fc3 = nn.Linear(1024, 512)
        self.fc4 = nn.Linear(512, 256)
        self.fc5 = nn.Linear(256, 128)
        self.fc6 = nn.Linear(128, 64)
        self.fc7 = nn.Linear(64, 3)
        self.fc8 = nn.Linear(1024, self.size * 3)
        self.fc9 = nn.Tanh()

    def forward(self, x):
        batchsize = x.size()[0]
        x = self.fc1(x).relu()
        x = self.fc2(x).relu()
        x = self.fc3(x).relu()
        x = self.fc4(x).relu()
        x = self.fc5(x).relu()
        x = self.fc6(x).relu()
        x = self.fc7(x).relu()
        x = self.fc8(x).reshape(batchsize, 3)
        return x
```

For the decoder used for voxel, called VoxelDecoder, the detail network architecture is shown below.

The difference between PointDecoder and VoxelDecoder lies in two aspects: the activation function and the shape.

Firstly, I replaced the Tanh activation function with a sigmoid activation function because I wanted to know the probability for each voxel grid.

Secondly, the shape size modified due to the presence of voxels, the final shape is [B, 5, 5, 5], where B and 5 are batch size and voxel size, respectively.

```
class VoxelDecoder(nn.Module):
    def __init__(self, in_size, latent_size):
        super(VoxelDecoder, self).__init__()
        self.size = latent_size
        self.fc1 = nn.Linear(in_size, 100)
        self.fc2 = nn.Linear(100, 160)
        self.fc3 = nn.Linear(160, 320)
        self.fc4 = nn.Linear(320, 250)
        self.fc5 = nn.Linear(250, 200)
        self.fc6 = nn.Linear(200, 125)
        self.fc7 = nn.Linear(1024, self.size ** 3)
        self.fc8 = nn.Sigmoid()

    def forward(self, x):
        batchsize = x.size()[0]
        x = self.fc1(x).relu()
        x = self.fc2(x).relu()
        x = self.fc3(x).relu()
        x = self.fc4(x).relu()
        x = self.fc5(x).relu()
        x = self.fc6(x).relu()
        x = self.fc7(x).relu()
        x = self.fc8(x).reshape(batchsize, self.size, self.size, self.size)
        return x
```

3. Visualization

For voxel visualization, I used the first prediction from every batch, and marked red color and blue color for prediction and ground-truth, respectively.

As for voxel visualization, I select the probability above 0.5 to retain the voxel grid.

```
# TODO:
if pred_vox != torch.tensor(0):
    # print("visualization 1")
    voxel = prediction_3C[0].detach().cpu()
    pc = pytorch3d.ops.cubify(voxel,0.5)
    if pc.isempty():
        print("empty")
    wr = r"cq_add_vox3d/3D1_prediction/"+"0"+"
    wr = r"cq_add_vox3d/3D1_prediction/"+"0"+"

    if cfg.dtype == "voxel":
        ax1.scatter(torch.[:,0], torch.[:,1], torch.[:,2], c = 'r', marker='.')
        ax1.scatter(gt[:,0], gt[:,1], gt[:,2], c = 'b', marker='.')

    elif cfg.dtype == "point":
        ax1.scatter(pred[:,0], pred[:,1], pred[:,2], s=2)
        ax1.scatter(gt[:,0], gt[:,1], gt[:,2], s=2)
        ax1.scatter(gt[:,0], gt[:,1], gt[:,2], s=2, c='blue')

    ax1.axis(xmin=-1.1, xmax=1.1, ymin=-1.1, ymax=1.1, zmin=-1.1, zmax=1.1)
    plt.savefig(wr)
    plt.close(fig)
    # print("visualization 2")
    # print("shape: ", pc.verts_packed().shape)
    # print("shape: ", pc.faces_packed().shape)
    # print("shape: ", pc.edges_packed().shape)
    # print("shape: ", pc.verts_packed()[:,0].shape)
    # print("shape: ", pc.verts_packed()[:,1].shape)
    # print("shape: ", pc.verts_packed()[:,2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1).shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[1].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[2].shape)
    # print("shape: ", pc.verts_packed()[:,0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].transpose(0,1)[0:3].shape)
```

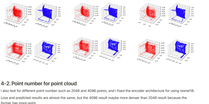
4. Comparison Results

4-1. Encoder architecture for point cloud

I used different encoder architecture, such as resnet18, resnet50, resnet101.

Loss and predicted results are almost the same in these three encoder architectures shown below.

ResNet18



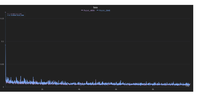
ResNet50

ResNet101

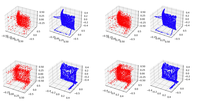
4-2. Point number for point cloud

I also test for different point number such as 2048 and 4096 points, and I fixed the encoder architecture for using resnet18.

Loss and predicted results are almost the same, but the 4096 result maybe more denser than 2048 result because the former has more point.



2048 points

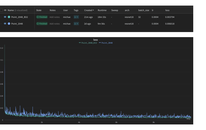


4096 points

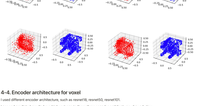
4-3. Different batch size for point cloud

I increased the batch size for training to 32, whereas the default batch size is 10.

Loss is lower than the default setting, and the predicted result is better than the default setting.



Batch size: 10 (Default)



Batch size: 32

4-4. Encoder architecture for voxel

I used different encoder architecture, such as resnet18, resnet50, resnet101.

Loss and predicted results have high variance in these three encoder architectures shown below.

It se