3DV2022: single-view 3D reconstruction model

310551083 許嘉倫

1 Introduction

在這項作業中我們需要實作 point cloud 和 voxel 的 decoder 將模型結果輸出為 (number point, 3) point cloud 的 xyz 座標和 (33, 33, 33) voxel 的 occupancy score。最後再將輸出的結果 render 在 2D image 上。

2 Point cloud

在訓練過程中，我們利用 chamfer loss，來當作 model的 loss function。

2.1 Image encoder

首先我們可以看到 Image encoder 的部份，利用 ResNet18 提取 feature map 並得到 dim 為 512 的 latent code。

```
class SingleViewto3D(nn.Module):
    def __init__(self, cfg):
        super(SingleViewto3D, self).__init__()
        self.device = "cuda"
        vision_model = torchvision_models.__dict__[cfg.arch](pretrained=True)
        self.encoder = torch.nn.Sequential(*(list(vision_model.children())[:-1]))
        self.normalize = transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
        # define decoder
        if cfg.dtype == "voxel":
            # TODO:
            self.decoder = VoxelDecoder(512)
        elif cfg.dtype == "point":
            self.n_point = cfg.n_points
            # TODO:
            self.decoder = PointDecoder(cfg.n_points, 512)
```

2.2 Point decoder

接下來是 decoder 的部份，輸出為 (number point, 3) point cloud 的 xyz 座標，最後利用 tanh activation function 將座標 normalize 到 -1 和 1 之間。以下是 decoder 實作的部份。

```
class PointDecoder(nn.Module):
    def __init__(self, num_points, latent_size):
        super(PointDecoder, self).__init__()
        self.num_points = num_points
        self.fc0 = nn.Linear(latent_size, 100)
        self.fc1 = nn.Linear(100, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 512)
        self.fc4 = nn.Linear(512, 1024)
        self.fc5 = nn.Linear(1024, self.num_points * 3)
        self.th = nn.Tanh()

    def forward(self, x):
        batchsize = x.size()[0]
        x = F.relu(self.fc0(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.th(self.fc5(x))
        x = x.view(batchsize, self.num_points, 3)
        return x
```

2.3 Model result

Figure 1 是 model 輸出結果的視覺化呈現。

3 Voxel

在訓練過程中，我們利用 Binary cross entropy 來計算 occupancy loss，來當作 model的 loss function。

3.1 Image encoder

Image encoder 和 point cloud 一樣，使用 ResNet18 當作 backbone 提取 feature map 並得到 dim 為 512 的 latent code。

3.2 Voxel decoder

接下來是 Voxel decoder 的部份，輸出為 (33, 33, 33)大小 的 voxel volume，最後利用 sigmoid activation function 來當作每個 voxel 的 occupancy score。

```
class VoxelDecoder(nn.Module):
    def __init__(self, latent_size):
        super(VoxelDecoder, self).__init__()
        self.fc0 = nn.Linear(latent_size, 100)
        self.fc1 = nn.Linear(100, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 512)
        self.fc4 = nn.Linear(512, 1024)
        self.fc5 = nn.Linear(1024, 33 * 33 * 33)
```

```
        self.th = nn.Sigmoid()

    def forward(self, x):
        batchsize = x.size()[0]
        x = F.relu(self.fc0(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.th(self.fc5(x))
        x = x.view(batchsize, 33, 33, 33)
        return x
```

3.3 Model result

Figure 2 是 model 輸出結果的視覺化呈現，在視覺化 voxel 時我會挑選 occupancy score 大於 0.4 才輸出，小於 0.4的 voxel 會過濾掉。

```
elif cfg.dtype == 'voxel':
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.voxels(prediction_3d.detach().cpu().numpy()[0] > 0.4, edgecolor='k')
    plt.show()
    R, T = look_at_view_transform(3, 0, 90)
    cameras = FoVPerspectiveCameras(device='cuda', R=R, T=T)
    raysampler = NDCMultinomialRaysampler(
        image_width=256,
        image_height=256,
        n_pts_per_ray=150,
        min_depth=0.1,
        max_depth=3.0,
    )
    raymarcher = EmissionAbsorptionRaymarcher()
    renderer = VolumeRenderer(
        raysampler=raysampler, raymarcher=raymarcher,
    )
    prediction_3d = prediction_3d[0].view(-1, 1, 33, 33, 33)
    prediction_3d[prediction_3d<0.4] = 0.0
    prediction_3d[prediction_3d>=0.4] = 1.0
    features = torch.ones(1, 3, 33, 33, 33).cuda()
    volumes = Volumes(
        densities=prediction_3d,
        features=features,
        voxel_size=3.0/33
    )
    rend, _ = renderer(cameras=cameras, volumes=volumes)[0].split([3, 1], dim=-1)
```

4 Bonus: Mesh

在訓練過程中，我利用 chamfer loss, laplacian smoothening loss, normal loss, edge loss，來當作model的 loss function。

4.1 Image encoder

Image encoder的部份和前面一樣，利用 ResNet18來提取 feature map 並得到 dim為 512的 latent code。

```
class SingleViewto3D(nn.Module):
    def __init__(self, cfg):
        super(SingleViewto3D, self).__init__()
        self.device = "cuda"
        vision_model = torchvision_models.__dict__[cfg.arch](pretrained=True)
        self.encoder = torch.nn.Sequential(*(list(vision_model.children())[:-1]))
        self.normalize = transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )

        # define decoder
        '''
        .......... 省略 point cloud and voxel 的實作
        '''
        elif cfg.dtype == "mesh":
            # try different mesh initializations
            mesh_pred = ico_sphere(4,'cuda')
            self.mesh_pred = pytorch3d.structures.Meshes(mesh_pred.verts_list()*cfg.batch_size,
            # TODO:
            self.decoder = MeshDecoder(mesh_pred.verts_packed().shape[0], 512)

    def forward(self, images, cfg):
        results = dict()

        total_loss = 0.0
        start_time = time.time()

        B = images.shape[0]

        images_normalize = self.normalize(images.permute(0,3,1,2))
        encoded_feat = self.encoder(images_normalize).squeeze(-1).squeeze(-1)
        '''
        .......... 省略 point cloud and voxel 的實作
        '''
        elif cfg.dtype == "mesh":
            # TODO:
            deform_vertices_pred = self.decoder(encoded_feat)
            mesh_pred = self.mesh_pred.offset_verts(deform_vertices_pred.reshape([-1,3]))
            return  mesh_pred
```

4.2 Mesh decoder

Mesh decoder 的部份因為要從 sphere deform 成 target mesh 的形狀，因此 model 的 output 必須是 shape 為 (number vertex of sphere, 3) 的 offset，有了這些 offset 我們就可以從 sphere deform 成 target mesh。Mesh decoder的架構和 Point decoder相同，差別在於 Mesh decoder的num_points代表 number vertex of sphere。

```
class MeshDecoder(nn.Module):
    def __init__(self, num_points, latent_size):
        super(MeshDecoder, self).__init__()
        self.num_points = num_points
        self.fc0 = nn.Linear(latent_size, 100)
        self.fc1 = nn.Linear(100, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 512)
        self.fc4 = nn.Linear(512, 1024)
        self.fc5 = nn.Linear(1024, self.num_points * 3)
        self.th = nn.Tanh()

    def forward(self, x):
        batchsize = x.size()[0]
        x = F.relu(self.fc0(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.th(self.fc5(x))
        x = x.view(batchsize, self.num_points, 3)
        return x
```

4.3 Dataloader

因為 pytorch default的collate_fn 不能回傳 pytorch3d Mesh object 因此我使用 pytorch3d 的 collate_batched_meshes，另外在 dataset.py 也需要修改成 collate_batched_meshes 可以讀取的資料型態。

```
def __getitem__(self, idx):
    '''
    ..... 省略 point cloud and voxel 的實作
    '''
    elif self.data_type == 'mesh':
        img, img_id = self.load_img(idx)
        collated_dict, object_id = self.load_mesh(idx)
        collated_dict['image'] = img
        collated_dict['object_id'] = object_id
        assert img_id == object_id

        return collated_dict

def load_mesh(self, idx):
    path = os.path.join(self.db[idx], 'model.obj')
```

```
    verts, faces, _ = load_obj(path, load_textures=False)
    faces_idx = faces.verts_idx

    # normalize
    center = verts.mean(0)
    verts = verts - center
    scale = max(verts.abs().max(0)[0])
    verts = verts / scale

    # make white texturre
    verts_rgb = torch.ones_like(verts)[None]  # (1, V, 3)
    # print(verts.shape)
    textures = TexturesVertex(verts_features=verts_rgb)

    mesh = Meshes(
        verts=[verts],
        faces=[faces_idx],
        textures=textures
    )
    collated_dict = dict()
    collated_dict['verts'] = verts
    collated_dict['faces'] = faces_idx
    collated_dict['mesh'] = mesh
    object_id = self.db[idx].split('/')[self.id_index]

    return collated_dict, object_id
```

另外在 train.py 也需要修改讀取 dataloader data 的方式。

```
def train_model(cfg: DictConfig):
    print(cfg.data_dir)
    shapenetdb = ShapeNetDB(cfg.data_dir, cfg.dtype)
    if cfg.dtype=='mesh':
        loader = torch.utils.data.DataLoader(
        shapenetdb,
        batch_size=cfg.batch_size,
        num_workers=cfg.num_workers,
        pin_memory=True,
        drop_last=True,
        collate_fn=collate_batched_meshes)
    '''
    ......省略
    '''
    for step in range(start_iter, cfg.max_iter):
        '''
        ......省略
        '''
        if cfg.dtype == 'mesh':
            ground_truth_3d = next(train_loader)
```

```
            images_gt = torch.Tensor([t.numpy() for t in ground_truth_3d['image']]).cuda()
            ground_truth_3d = ground_truth_3d['mesh'].cuda()
```

4.4 Model result

Figure 3 是 model 輸出結果的視覺化呈現，在視覺化 mesh 時我會從 mesh sample 5000 points 來呈現。另外我也有嘗試只用 chamfer loss和 laplacian smoothening loss 來訓練模型，結果如 figure 4 所示，每張圖片產生的 mesh 基本上都一樣。

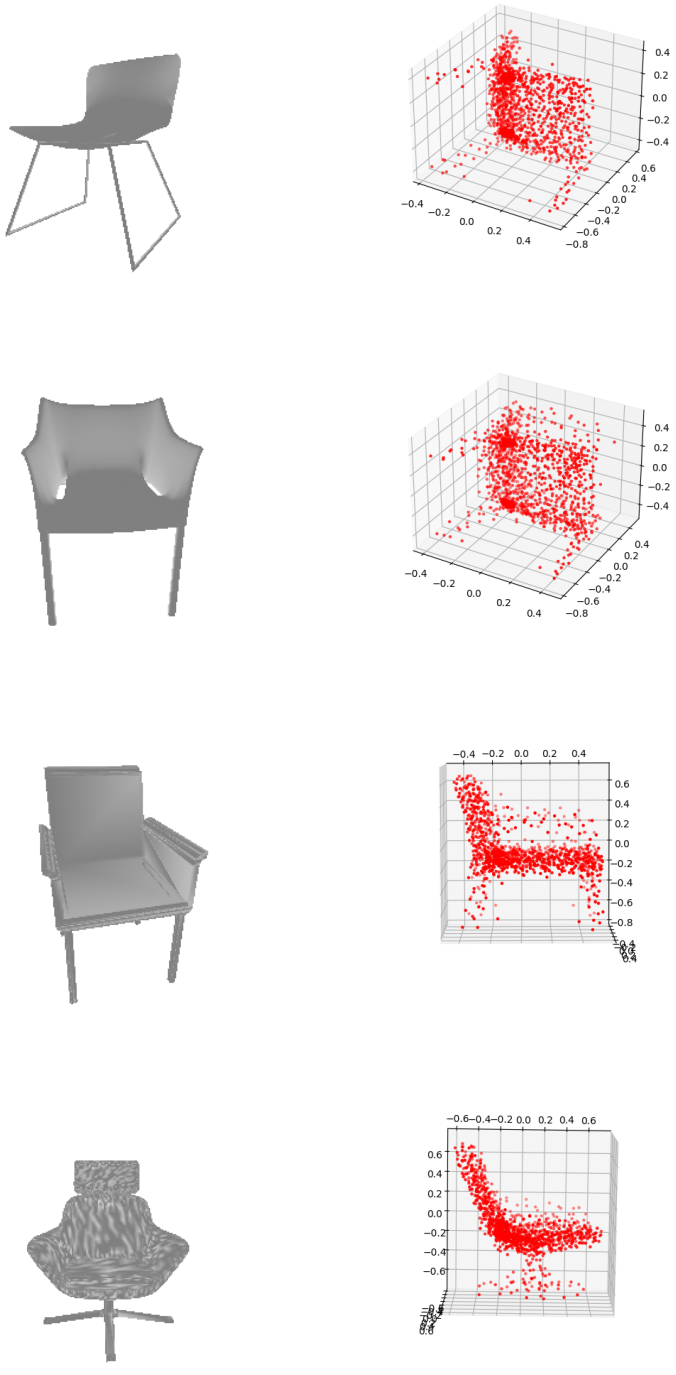

Figure 1: Point cloud visualization

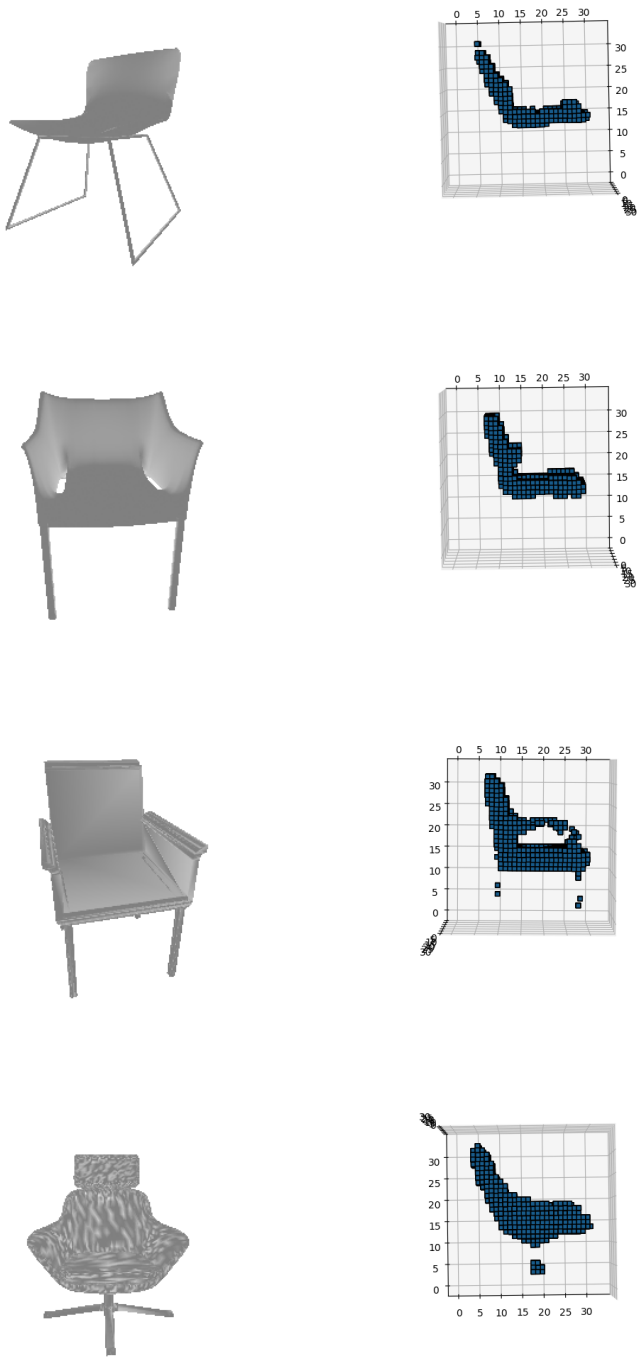

Figure 2: Voxel visualization

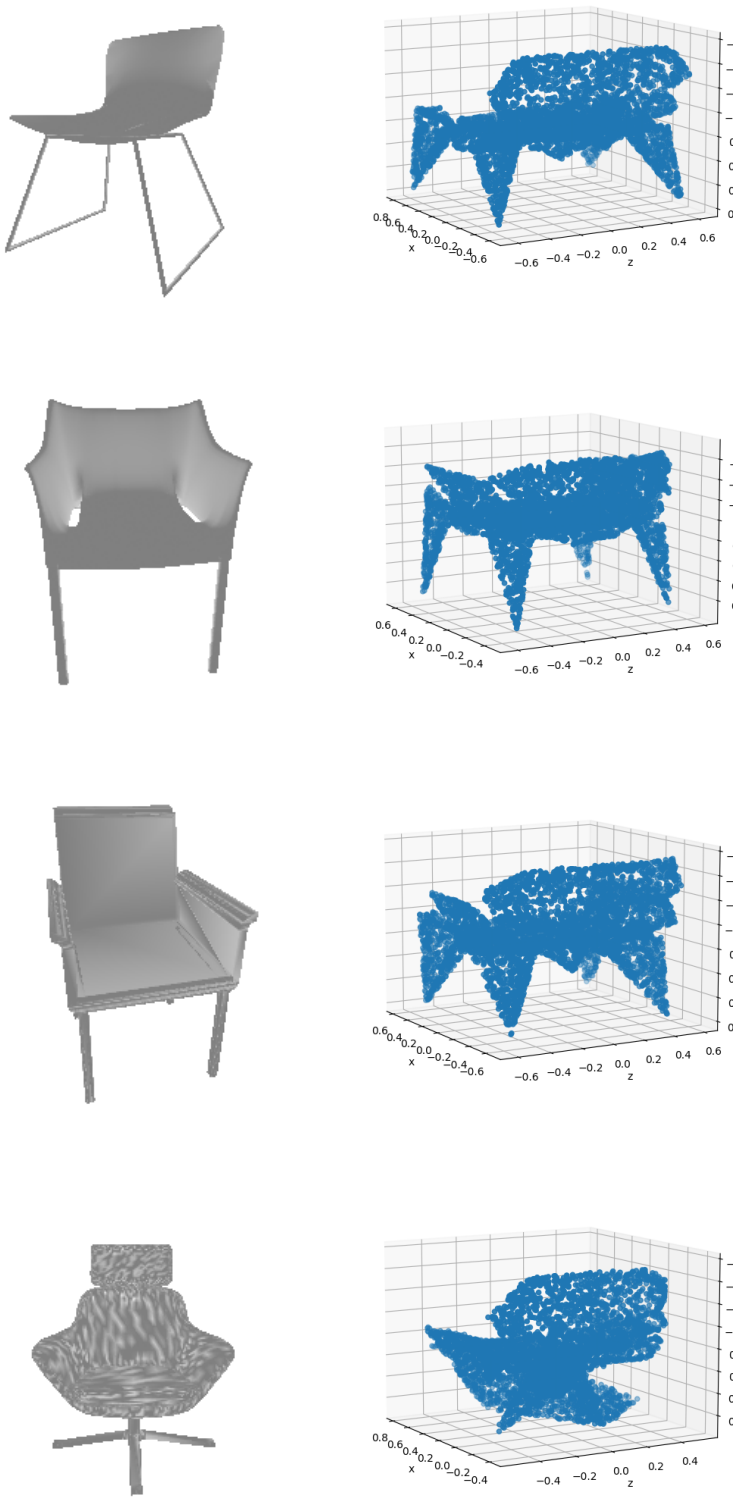

Figure 3: Mesh visualization

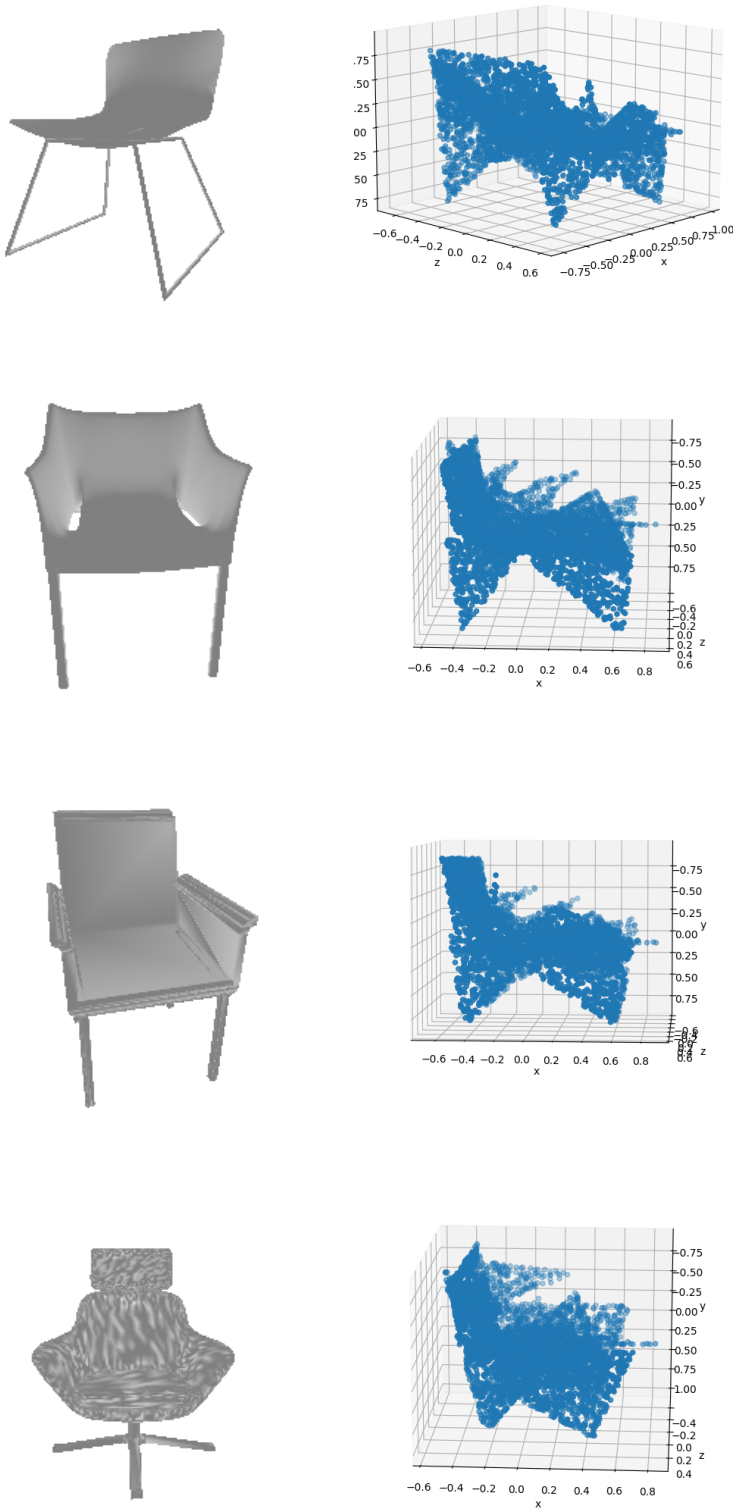

Figure 4: Mesh visualization only train with chamfer and laplacian loss