Masked Autoencoders for Point Cloud Self-supervised Learning

Yatian Pang, Wenxiao Wang, Francis E.H. Tay, Wei Liu, Yonghong Tian, Li Yuan

2022

Abstract

Point-MAE is a model that introduces the masked autoencoder into point cloud learning, demonstrating strong performance in tasks like part segmentation and object classification. However, during the replication process, we noticed its limitations in fine-grained tasks. By refining the network structure, we achieved enhanced performance, particularly in fine-grained classification tasks such as 3D face recognition.

Keywords: Point cloud learning, 3D face recognition.

1 Introduction

Self-supervised learning revolutionizes the process of feature acquisition by extracting latent features from unlabeled data, eliminating the need for human-defined annotations. The methodology involves crafting a pretext task for initial model pre-training, followed by fine-tuning downstream tasks to enhance task-specific performance. The distinctive feature of self-supervised learning lies in its reduced reliance on labeled data, a factor that has propelled significant advancements in the field of natural language processing (NLP).

Point-MAE [7] was introduced with the utilization of masked autoencoder (MAE) to achieve self-supervised learning on 3D point clouds. The methodology involves patching the point cloud through the Farthest Point Sampling (FPS) & K-Nearest Neighbor (KNN) algorithm, followed by tokenization of the patches using PointNet. Subsequently, the model learns high-level features of objects by performing random masking reconstruction.

Despite these efforts, in our replication process, when we trained and tested Point-MAE on a 3D facial dataset for 3D face recognition, we observed that Point-MAE lacked the capability to extract fine-grained features. As a result, the accuracy of facial recognition did not meet our expectations.

To delve deeper into Point-MAE's capacity for extracting fine-grained features, we implemented the following improvements: 1) As we know, the scale of the 3D facial dataset is relatively limited, and synthetic facial data may cause a domain consistency problem with the real facial data. To deal with this problem, we generate a combined facial dataset for the pre-training stage of Point-MAE. 2) In order to capture more discriminative features of the face, especially the information from distinct facial components, is considered crucial. To enhance the extraction of key facial features, we employed the Differences of Normals (DoN) algorithm

to initially extract essential facial information. 3) We leverage Point-MAE to perform patch-based random masking reconstruction and super-resolution of part of pre-training for the encoder.

2 Related works

2.1 Autoencoders

Typically, an autoencoder comprises an encoder followed by a decoder. The encoder is tasked with transforming inputs into high-level latent features, while the decoder reconstructs these latent features to replicate the original input. The optimization objective is to minimize the dissimilarity between the reconstructed data and the initial input, often measured using metrics like mean squared error loss in pixel space for images. Our approach specifically falls within the denoising autoencoder category, aiming to enhance model robustness by introducing input noise. Masked autoencoders, following a similar principle, introduce input noise through a masking operation. For instance, in natural language processing (NLP), BERT [1] employs masked language modeling, randomly masking tokens in the input and utilizing an autoencoder to predict vocabulary corresponding to the masked tokens. In computer vision, both MAE [4] and SimMIM [11] propose analogous masked image modeling, randomly masking input image patches and employing autoencoders to predict the masked patches in pixel space. Drawing inspiration from these concepts, our work focuses on integrating masked autoencoders into the realm of point cloud data.

2.2 Transformers

Transformers [9] leverage the self-attention mechanism to capture global dependencies in input data and have achieved remarkable success in natural language processing (NLP). Following the emergence of Vision Transformer (ViT) [2], Transformer architectures gained popularity in computer vision. However, when it comes to serving as backbones for masked autoencoders in point cloud representation learning, transformer architectures are not as extensively developed. PCT [3] addresses this gap by designing a dedicated input embedding layer and modifying the self-attention mechanism in Transformer layers. Similarly, PointTransformer [14] adjusts the Transformer layer and incorporates additional aggregation operations between Transformer blocks. In contrast, the recent approach Point-BERT [12] introduces a standard Transformer architecture but relies on DGCNN [10] for pre-training support. Diverging from prior works, our approach introduces an architecture solely based on standard Transformers.

2.3 3D face recognition based on deep learning

3D face recognition based on deep learning can be categorized into depth-based methods and point-based methods. Depth-based methods first convert depth image into point cloud data to segment the facial region and then convert it back into a depth image as input into a 2D CNN network for feature extraction. Some previous works such as Led3D [6] and LMFNet [15] already made some progress. Nevertheless, during the preprocessing of depth maps, there is a potential loss of crucial geometric information, which may result in a decrease in accuracy. To alleviate this issue, methods based on direct feature extraction from point clouds have been proposed. PointFace [5] employs a PointNet++ [8] network architecture for feature extraction,

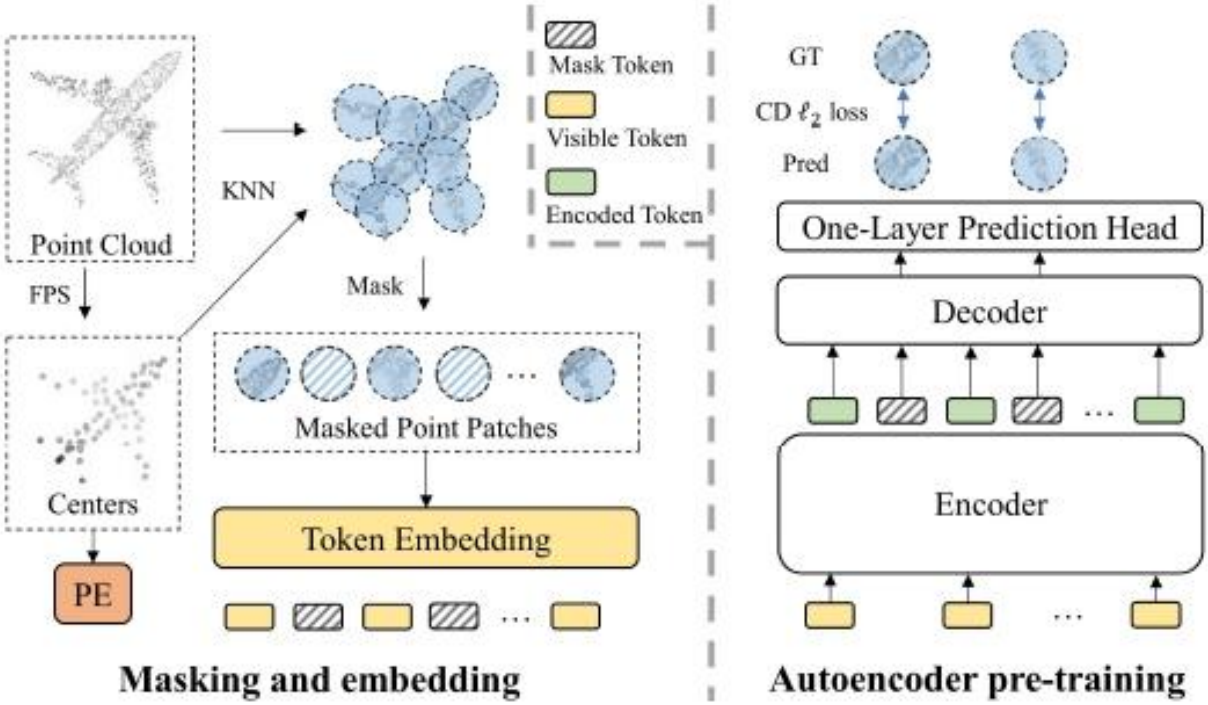

Figure 1. Overview of Point-MAE.

simultaneously utilizing a pairing selection strategy to employ contrastive learning loss functions for extracting discriminative features. Zhang [13] using GPMM model to generate a large-scale synthetic facial dataset for training and using a lightweight PointNet network to extract features and test it on "in-the-wild" dataset, which may lead to domain inconsistencies with "in-the-wild" dataset.

3 Method

3.1 Overview of Point-MAE

As Fig. 1 shows, for the pre-training stage, given a point cloud P, we first use furthest point sampling (FPS) to sample the center point of each patch and then with the K-nearest neighbor (KNN) algorithm to generate point patches.

$$\mathcal{P} = \text{KNN}(\text{FPS}(P)), \tag{1}$$

then we adopt a PointNet network to embed the patches into tokens,

$$T = \text{PointNet}(\mathcal{P}). \tag{2}$$

We use random masking strategy to generate masked point tokens $T_{\text{mask}}$and visible point tokens T_{vis}. Visible point tokens are used as input for the transformer encoder to perform feature extraction. The resulting features, denoted as f_{enc}, are concatenated with masked empty tokens, which is a learnable parameter. Position embedding will be added. This combined information is then input into the transformer decoder for the reconstruction of the masked regions.

$$f_{\text{enc}} = \text{Encoder}(T_{\text{vis}}), \tag{3}$$

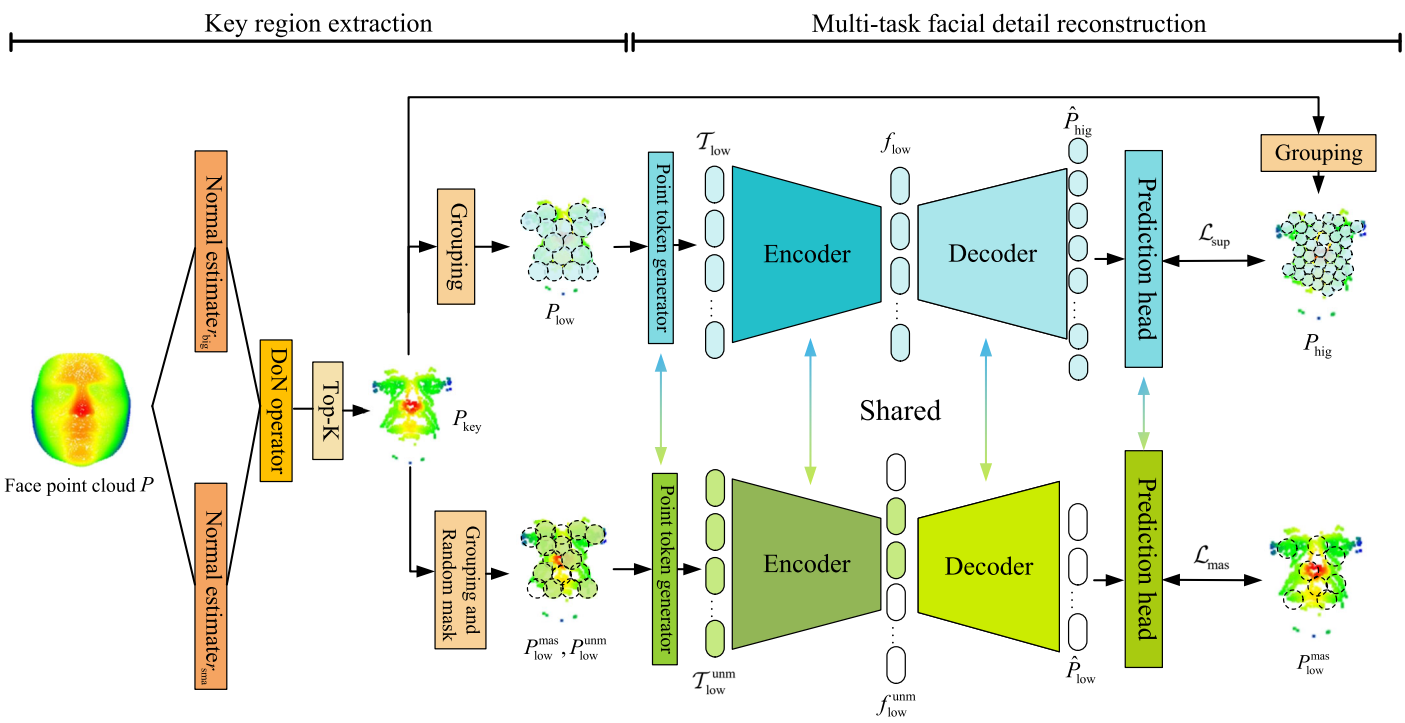

Figure 2. Overall architecture of Point-MAE for 3D face recognition.

$$\mathcal{P}_{\text{rec}} = \text{Decoder}(\text{Cat}(f_{\text{enc}}, f_{\text{mask}}) + pos). \tag{4}$$

The Chamfer distance loss function will be utilized to supervise the reconstruction effectiveness of the masked point patches.

For the fine-tuning stage, the decoder will be discarded, and instead, downstream networks tailored for specific tasks will be introduced. The entire model parameters, including the encoder, will undergo fine-tuning for various downstream tasks.

3.2 Point-MAE for 3D face recognition

3.2.1 Hybrid facial dataset

To address the domain inconsistency between synthesized and real faces, we employed a hybrid facial dataset comprising both real and synthesized faces for pretraining. Initially, we generated approximately 125,000 synthetic faces using the GPMM model. Subsequently, we introduced the FRGC v2 dataset for a combined training approach.

3.2.2 Facial key region extraction

In order to enable the model to learn more information about key facial regions such as the nose, eyes, and mouth, we have designed a method for extracting key facial regions. Given facial point cloud $\mathcal{P}$, we extract the key region with difference of normals (DoN) operator,

$$P_{\text{key}} = \text{DoN}(P) = \{p : \Delta n_p > t\}, \tag{5}$$

where t is a threshold,Δn_p is the normal difference for any point $p \in P$,

$$\Delta n_p = \frac{n_p^{\text{big}} - n_p^{\text{sma}}}{2} \tag{6}$$

$$n_p = \text{Norm}(P, p, r). \tag{7}$$

The function Norm computes the normal vector n_p at p using the point within a neighborhood of radius r on the facial point cloud P. The difference between n_p^{big} and n_p^{sma} lies in their respective values of r, where $r_{\text{sma}} < r_{\text{big}}$. The variation in the calculated normals using different radii for key facial regions, such as facial features, will be significantly higher than some smoother regions of the face. This method allows for a rough segmentation of these areas.

3.2.3 Patch-based super-resolution

In addition to the random masking reconstruction using Point-MAE, we have also implemented a patch-based super-resolution method to learn more intricate geometric features. After the patchization and tokenization using Eq.1 and Eq. 2, we now have N patches and N point token. We do not mask the tokens; instead, we input the complete point tokens into the encoder. Subsequently, we concatenate the output features with $2N$ empty vectors and feed them into the decoder for super-resolution, resulting in generating twice as many point patches, which have $2N$ point patches.

3.2.4 Loss function

For different stages, we employ distinct loss functions for supervision. During the pre-training phase, we utilize the Chamfer Distance loss function to simultaneously supervise random occlusion reconstruction and patch-based super-resolution. This helps ensure consistency between the reconstructed point cloud and the occluded point cloud,

$$\mathcal{L}_{\text{CD}} = \frac{1}{|\mathcal{P}|} \sum_{\text{Pa} \in \mathcal{P}} \min_{\text{Pa}' \in \hat{\mathcal{P}}} \|\text{Pa} - \text{Pa}'\|_2^2 + \frac{1}{|\hat{\mathcal{P}}|} \sum_{\text{Pa} \in \hat{\mathcal{P}}} \min_{\text{Pa}' \in \mathcal{P}} \|\text{Pa} - \text{Pa}'\|_2^2. \tag{8}$$

During the fine-tuning stage, we concurrently apply cross-entropy loss and a feature similarity loss for supervision. This is done to ensure the minimization of distances within the same class while maximizing distances between different classes, thereby promoting effective fine-tuning,

$$\mathcal{L}_{\text{fin}} = \mathcal{L}_{\text{ce}} + \mathcal{L}_{\text{sim}} = \text{CrossEntropy}(y, \hat{y}) + \max\left(\cos(f, f^{\text{pos}}) - \cos(f, f^{\text{neg}}) + m, 0\right). \tag{9}$$

4 Implemenbtation details

4.1 Comparing with the released source codes

Our code is primarily based on the Point-MAE implementation, with additional improvements incorporated.

The first improvement involves writing code for computing normals and implementing the DoN operator based on Open3D. This code is included in our dataset loader for data loading. However, in practical use,

since this part is only utilized during the pre-training stage, we perform key region extraction directly in the preprocessing phase:

```
def points_to_o3d(points):
        cloud_o3d = o3d.geometry.PointCloud()
        cloud_o3d.points = o3d.utility.Vector3dVector(points[:, :3])
        return cloud_o3d
def points_to_cloud(points):
        if points.shape[1] == 3:
            cloud = pcl.PointCloud()
        elif points.shape[1] == 4:
            cloud = pcl.PointCloud_PointXYZI()
        cloud.from_array(np.asarray(points, dtype='float32'))
        return cloud
def don_segment(points, threshold, radius_s, radius_l):
        cloud_o3d = points_to_o3d(points[:, :3])
        cloud_o3d.estimate_normals(search_param=o3d.geometry.
        KDTreeSearchParamHybrid(radius_s, 64))
        normals = np.array(cloud_o3d.normals)
        cloud_o3d.estimate_normals(search_param=o3d.geometry.
        KDTreeSearchParamHybrid(radius_l, 64))
        normall = np.asarray(cloud_o3d.normals)
        don = (normals - normall) / 2
        removed = []
        for i in range(points.shape[0]):
            mod = np.linalg.norm(don[i])
            if mod >= threshold:
                removed.append(i)
        small_points=points[removed,:]
       # print(tosmall_points.shape)
        small_points=misc.fps(torch.tensor(small_points)\
        .float().to(device).unsqueeze(0),128)
        #points_list=torch.cat([points_list,small_points],dim=0)
        return small_points
```

The second improvement involves rewriting the Group class to implement patch-based super-resolution.

```
class Group(nn.Module):  # FPS + KNN
    def __init__(self, num_group, group_size):
        super().__init__()
        self.num_group = num_group
        self.group_size = group_size
        self.knn = KNN(k=self.group_size, transpose_mode=True)
        self.knn_2=KNN(k=self.group_size//2,transpose_mode=True)
        self.ln=64

    def forward(self, xyz,key):
        '''
            input: B N 3
            ---------------------------
```

```
            output: B G M 3
            center : B G 3
        '''
        batch_size, num_points, _ = xyz.shape
        #print(self.group_size)
        # fps the centers out
        #center=misc.fps(xyz,64)
        #center = self.don_segment(xyz,0.1,0.05,0.15) # B G 3
        key=misc.fps(key,64)
        #print(center.size())
        # knn to get the neighborhood
        _, idx = self.knn(xyz, key.squeeze(1)) # B G M
        #print(idx)
        assert idx.size(1) == self.num_group
        assert idx.size(2) == self.group_size
        idx_base = torch.arange(0, batch_size, device=xyz.device)\
        .view(-1, 1, 1) * num_points
        idx = idx + idx_base
        idx = idx.view(-1)
        #print(xyz.size())
        vis_neighborhood = xyz.view(batch_size * num_points, -1)[idx, :]
        vis_point=vis_neighborhood.view(batch_size,self.num_group\
        *self.group_size,3)
        center=key

        mask_point=xyz.view(batch_size*num_points,3)[~idx,:]
        mask_point=vis_point.view(batch_size,num_points-self.group_size\
        *self.num_group,3)
        mask_center=misc.fps(mask_point,128)
        _,idx=self.knn_2(mask_point,mask_center)
        idx_base=torch.arange(0,batch_size,device=xyz.device)\
        .view(-1,1,1)*2048
        idx=idx+idx_base
        idx=idx.view(-1)
        neighborhood=mask_point.view(batch_size*2048,-1)[idx,:]
        neighborhood=neighborhood.view(batch_size,self.num_group*2,\
        self.group_size//2,3).contiguous()
        neighborhood=neighborhood-mask_center.unsqueeze(2)
        vis_neighborhood = vis_neighborhood.view(batch_size,\
        self.num_group,self.group_size, 3).contiguous()
        # normalize
        vis_neighborhood = vis_neighborhood - center.unsqueeze(2)
        #print(neighborhood.size())
        #print(mask_center.size())
        return vis_neighborhood, center,neighborhood,mask_center
```

We also create a new dataloader for our hybrid facial dataset,

```
class synthetic_train(data.Dataset):
        def __init__(self, num_points, root, transforms=None, train=True):
```

```
3                 super().__init__()
4
5                 self.transforms = transforms
6                 self.num_points = num_points
7                 self.root = os.path.abspath(root)
8                 self.folder = 'FRGC_virtual_id_aug2_normal'
9                 self.data_dir = os.path.join("./data", self.folder)
10                self.data=os.listdir("./data/syn_Data/")
11                self.ln=64
12                self.train = train
13                if self.train:
14                        self.files, self.labels = _get_data_files(
15                os.path.join(self.data_dir, 'FRGC_virtual_id_all.txt')
16                                                                    )
17                else:
18                        self.files, self.labels = _get_data_files(
19                os.path.join(self.data_dir, 'FRGC_virtual_id_all.txt')
20                                                                    )
21                self.data=self.files+self.data
22        def xyz2sphere(self,xyz, normalize=True):
23                rho = torch.sqrt(torch.sum(torch.pow(xyz, 2), dim=-1\
24  , keepdim=True))
25                rho = torch.clamp(rho, min=0)  # range: [0, inf]
26                theta = torch.acos(xyz[..., 2, None] / rho)
27                phi = torch.atan2(xyz[..., 1, None], xyz[..., 0, None])
28   # check nan
29                idx = rho == 0
30                theta[idx] = 0
31                if normalize:
32                        theta = theta / np.pi  # [0, 1]
33                        phi = phi / (2 * np.pi) + .5  # [0, 1]
34                out = torch.cat([rho, theta, phi], dim=-1)
35                return out
36        def __getitem__(self, idx):
37                #file1, label1, file2, label2 = self.pair_files[idx]
38                file=self.data[idx]
39                if len(file.split("/"))>2:
40                        single_p1 = np.loadtxt(os.path.join(self.root,\
41   file))single_p1_key=np.load(os.path.join(self.root,file)\
42   .replace("FRGC_virtual_id_aug2_normal",\
43   "FRGC_virtual_id_aug2_normal_key")+".npy")
44                        labels=1
45                else:
46                        single_p1 = pd.read_csv(os.path.join(self.root,\
47   "syn_Data",file),sep=" ")
48                        single_p1_key=np.load(os.path.join(self.root,\
49   "syn_Data_key",file+".npy"))
50                        data_frames = pd.DataFrame(single_p1)
```

```
51
52                      single_p1 = np.array(data_frames.values)
53                      labels=0
54
55              single_p1 = pc_normalize(single_p1)
56
57              if self.num_points > single_p1.shape[0]:
58                      idx = np.ones(single_p1.shape[0], dtype=np.int32)
59                      idx[-1] = self.num_points - single_p1.shape[0]\
60  + 1
61                      single_p_part = np.repeat(single_p1, idx, axis=0)
62              else:
63                      single_p_idx = np.arange(0, single_p1.shape[0])
64                      single_p_idx = np.random.choice(single_p_idx\
65  , size=(self.num_points,), replace=False)
66                      single_p_part = single_p1[single_p_idx, :]
67              points1 = single_p_part.copy()
68              pt_idxs1 = np.arange(0, points1.shape[0])
69              if self.train:
70                      np.random.shuffle(pt_idxs1)
71                      #np.random.shuffle(pt_idxs2)
72
73              current_points1= points1[pt_idxs1]
74              if self.transforms is not None:
75                      current_points1 = self.transforms\
76  (current_points1)
77                      center=self.transforms(single_p1_key)
78              else:
79                      current_points1 = torch.\
80  from_numpy(current_points1).float()
81                      center=torch.from_numpy(single_p1_key).float()
82              return current_points1,labels
83      def __len__(self):
84              return len(self.data)
```

4.2 Experimental environment setup

Our experimental environment is set up on Ubuntu 18.04 using Anaconda. The specific package versions include PyTorch 1.10.0 with CUDA 11.1, torchvision 1.11.0, Open3D 0.9.0, and Python 3.6.0.

4.3 Main contributions

The main contributions of our work can be summarized as follows:

- We design a pretraining process using a hybrid dataset that combines both real facial data and synthetic facial data.
- We utilized a facial key region extraction module to learn more discriminative facial features.

Method	Input	Rank-one recognition rate (%)					
		FE	NU	OC	PS	TM	Total
VGG-16	Depth	94.76	99.57	44.68	49.21	34.50	70.58
ResNet-34	Depth	96.09	99.29	54.91	61.39	45.00	76.56
Inception-v3	Depth	93.56	98.97	56.98	54.14	42.17	74.44
MobileNet-v2	Depth	95.74	98.91	61.44	69.92	43.00	79.49
Led3D	Depth	97.62	99.62	68.93	64.81	64.97	81.02
Led3D	Depth+Normal	98.17	99.62	78.10	70.38	65.28	84.22
PointFace	XYZ	97.93	99.35	77.06	72.03	66.33	84.78
PointFace	XYZ+Normal	98.52	99.46	80.67	73.69	67.00	87.18
MQFNet	Normal	97.31	99.95	80.97	73.61	61.67	86.55
LMFNet	Normal	99.01	99.95	83.08	78.61	75.60	87.49
LMFNet	Depth+Normal	**99.21**	99.95	84.40	79.05	**76.31**	88.01
our Point-MAE	XYZ	98.63	99.78	**85.74**	80.02	63.83	89.59
our Point-MAE	XYZ+Normal	99.06	**99.95**	85.49	**84.57**	71.00	**91.17**

Table 1. Rank-one recognition accuracy on Lock3DFace

- Additionally, we introduced a Patch-based super-resolution method to learn more intricate facial features.
- We fine-tuned the pre-trained model for 3D face recognition tasks and achieved the state-of-the-art across multiple datasets.

5 Results and analysis

We are conducting fine-tuning and testing on the Lock3DFace facial dataset, which has five subsets including expression (FE), natural (NU), occlusion (OC), pose (PS), and time-pulse (TM).In this protocol we randomly select 340 individuals for fine-tuning and 169 individuals for testing. Our evaluation criterion utilizes the Rank-one recognition rate. Table 1 shows the comparison with other 3D face recognition methods.

Our approach achieved the best performance on all three subsets and demonstrated a significant improvement in the overall recognition rate. In comparison to PointFace, which also utilizes XYZ and normal vector information as input, our method exhibited an almost 4% increase in overall performance, surpassing it on all subsets. Additionally, when compared to the second-best performing LMFNet, our approach showed superior performance by a considerable margin.

We also visualized the facial key regions segmented by the DoN operator, in Fig. 3 revealing our ability to successfully segment most of the facial features. Additionally, we presented visualizations of the results for random masking reconstruction and super-resolution. In the random masking reconstruction, colored points represent the reconstructed points, demonstrating that our method effectively reconstructs masked regions. In comparison to random masking reconstruction, the results obtained from super-resolution showcased superior reconstruction quality, especially in fine details.

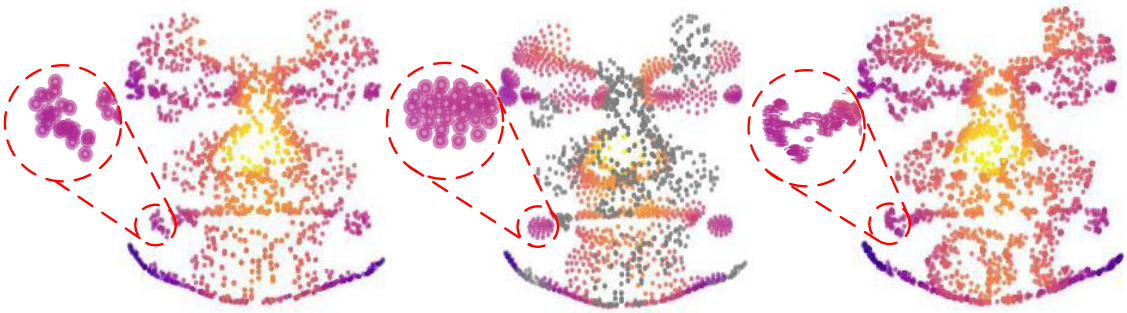

Figure 3. An example of the facial key region, its random masking reconstruction, and super-resolution

6 Conclusion and future work

In this paper, we introduce Point-MAE, a masked autoencoder (MAE) designed for 3D point cloud learning. We have redesigned and improved Point-MAE to endow it with the capability to address fine-grained classification tasks, specifically focusing on enhancements for 3D facial recognition. By pretraining on a hybrid facial dataset and incorporating facial key region extraction along with patch-based super-resolution, the network has been empowered to extract more discriminative facial features. We conducted fine-tuning and testing on the Lock3DFace dataset, achieving state-of-the-art results in terms of the rank-one recognition rate. Additionally, we conducted visualization experiments to demonstrate the effectiveness of our approach. In the future, we plan to explore methods to make the model more lightweight to enable real-time predictions.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[3] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7(2):187–199, Apr 2021.

[4] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv:2111.06377*, 2021.

[5] Changyuan Jiang, Shisong Lin, Wei Chen, Feng Liu, and Linlin Shen. Pointface: Point set based feature learning for 3d face recognition. In *2021 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–8, 2021.

[6] Guodong Mu, Di Huang, Guosheng Hu, Jia Sun, and Yunhong Wang. Led3D: A Lightweight and Efficient Deep Approach to Recognizing Low-Quality 3D Faces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[7] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning, 2022.

[8] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[10] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[11] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9653–9663, 2022.

[12] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19313–19322, 2022.

[13] Ziyu Zhang, Feipeng Da, and Yi Yu. Learning directly from synthetic point clouds for "in-the-wild" 3d face recognition. *Pattern Recognition*, 123:108394, 2022.

[14] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16259–16268, October 2021.

[15] Panzi Zhao, Yue Ming, Xuyang Meng, and Hui Yu. Lmfnet: A lightweight multiscale fusion network with hierarchical structure for low-quality 3-d face recognition. *IEEE Transactions on Human-Machine Systems*, 53(1):239–252, 2023.